Simulating FogDirector Application Management

Stefano Forti[1], Alessandro Pagiaro, Antonio Brogi

Department of Computer Science, University of Pisa, Italy

Abstract

Achieving a correct and effective management of Fog computing applications is a non-trivial task to accomplish, which includes considering specific application requirements as well as dynamic infrastructure characteristics. CISCO FogDirector is a tool that can be used to manage the entire life-cycle of IoT applications over Fog infrastructures by relying on a RESTful API. In this paper, we present a prototype simulation environment, FogDirSim, compliant with FogDirector API. FogDirSim permits comparing different application management policies according to a set of well-defined performance indicators (viz., uptime, energy consumption, resource usage, type of alerts) and by considering probabilistic variations of the applications workload and failures of the underlying infrastructure. A lifelike example is used to validate the prototype and to show its usefulness in selecting the best management policy.

Keywords: Fog computing, Application Management, CISCO Fog Director, RESTful API

1. Introduction

Fog computing architectures are characterised by the seamless exploitation of heterogeneous devices and communication protocols [1, 2]. Particularly, the presence of many resource-constrained devices in Fog networks calls for *resource-aware* application management more than in Cloud scenarios, where

[1]Corresponding author. Email: stefano.forti@di.unipi.it

elasticity and scalability are less critical [3, 4]. Furthermore, in addition to hardware failures, Fog devices can incur in the loss of Internet access due to their mobility or to edge network failures [5]. Last but not least, many Fog computing applications – such as e-health [6] or autonomous vehicles [7] – have stringent Quality of Service (QoS) requirements to meet due to their business-, mission- and sometimes life-critical nature.

Overall, managing applications in highly distributed Fog computing scenarios requires to decide on which node each application should be installed and run at each moment in time, depending on different infrastructure conditions and always considering application requirements in terms of hardware (e.g., RAM, CPU) and QoS (e.g., uptime, energy consumption). This process normally involves human decision-makers (i.e., application operators) that design and implement suitable management scripts to guarantee the fulfilment of all such requirements throughout the application life-cycle, viz. from application deployment to retirement.

Unfortunately, bad or poor management choices can lead to unsatisfactory service QoS, waste of electrical power or money, and – worst of all – customer (and, therefore, financial) losses due to bad service placement, overloading of Fog devices and service downtime, respectively. For instance, application operators have to decide where to suitably migrate applications whenever their deployment nodes undergo failures or overloading situations due to environmental factors, or when to increase the resources allocated to an application whenever it has to deal with a workload higher than expected, or again how many replicas of a certain application are needed to guarantee a satisfactory uptime. Finally, they might also consider purchasing new hardware to extend their deployment infrastructure if the current setup cannot suitably satisfy the application needs.

In general, designing, tuning and enacting correct and effective application management policies in Fog computing scenarios is an open research problem, which calls for suitable tooling and support [8, 9]. Indeed, before adopting new management policies in production environments, an effective loop for application operators would include:

1. writing a script to implement their management policies for deploying application artefacts and handling deployment alerts and faults at runtime,
2. experimenting and assessing the actual behaviour of their script in a simulated environment capable of reproducing all possible infrastruc-

ture and workload conditions,
3. obtaining an estimate of the Key Performance Indicators (KPIs) achieved by the considered management script within the simulated environment so to assess its effectiveness,
4. refining their script based on the obtained results, if needed, and possibly simulating it again by repeating steps 1–4.

To this end, research has proposed some first prototype simulators that permit to run and compare management policies (e.g., iFogSim [10], YAFS [11]). However, the major shortcomings that these prototypes share concern:

(A) the high level of abstraction with respect to industrial Fog computing solutions and APIs for managing Fog applications (like CISCO FogDirector), and

(B) the fact that few considers the possibility for Fog devices to fail or temporarily disconnect from the network.

Whilst (A) introduces an additional layer of complexity for application operators that must first translate – when possible – their infrastructure and application models, and naturally their management scripts, into the input accepted by the prototype simulators, (B) results in failing to capture one of the main aspect of Fog computing and highly distributed infrastructures in general, i.e. failures and device churn.

Among the first works inspired to industry-based solutions, Forti et al. [12] proposed an operational semantics of all basic management functionalities of CISCO FogDirector. A proof-of-concept implementation, FogDirMime, based on such semantics was then proposed to permit experimenting with abstract program specifications of CISCO FogDirector management scripts and to collect some customised performance indicators against probabilistic infrastructure variations. Still, FogDirMime only constitutes the core API of a simulation environment of Fog application management with CISCO FogDirector, and it shows important limitations. More precisely, it is not able to directly input actual CISCO FogDirector management scripts, nor it provides the possibility to set configuration parameters for the simulation to take place. Last but not least, FogDirMime proof-of-concept leaves the users alone with the task of deciding which KPIs to compute and how to compute them.

In this article, pursuing the (mostly formal) modelling effort of FogDirMime [12], we will describe the design and implementation of FogDirSim, a prototype discrete-event simulation environment which is fully compliant to (a feature-complete subset of) CISCO FogDirector RESTful API for application and infrastructure management. FogDirSim is a simulator that permits comparing different application management policies according to a set of KPIs. Based on monitored aggregate historical data, the prototype considers probabilistic variations and failures in the underlying infrastructure, which can happen independently from the considered management. In short:

> *This work aims at contributing to supporting the design and assessment of correct and effective management policies for Fog computing applications, by providing a (discrete-event) simulation environment based on the model and API of CISCO FogDirector, and capable of predicting management KPIs against probabilistically varying deployment conditions.*

As we will show over a lifelike motivating example, FogDirSim can be fruitfully exploited by application operators to *a priori* validate, assess and compare in a simulated environment different management policies for their applications. Differently from existing approaches, application operators can directly simulate the very same management scripts that they can run in production against an actual instance of CISCO FogDirector.

In addition to testing the correctness of their management against varying infrastructure conditions, FogDirSim can probabilistically predict:

- application uptime and downtime,
- the different types of alerts raised,
- energy consumption due to management choices,
- the convergence speed of different management policies,
- their robustness against varying device failure rates, and
- their capacity to adapt to changing application workloads.

The rest of this paper is organised as follows. After briefly describing CISCO FogDirector functionalities (Sect. 2), we describe a motivating scenario from the field of smart building application over lifelike infrastructures (Sect. 3). Then, we describe the implementation and functionalities of our simulator, FogDirSim (Sect. 4), and – after discussing its validation – we use it to solve the motivating example (Sect. 5). Finally, after reviewing some related work (Sect. 6), we conclude and point to possible directions for future work (Sect. 7).

2. Background: CISCO FogDirector

In 2016, CISCO released CISCO IOx, a first Fog platform that brings together capabilities from the networking operating system Cisco IOS, and from the Linux operating system. Such convergence permits to get together networking management and the possibility to run Linux-based applications within CISCO IOx, which enables developers to easily create a wide variety of IoT applications. Within the IOx framework, CISCO provides CISCO FogDirector [13], a single control panel to manage, monitor and troubleshoot Fog applications and infrastructure devices.

First, CISCO FogDirector enables application operators to achieve consistent management throughout the lifecycle of Fog applications. Indeed, it permits to start and stop applications, manage their versioning and updates, view and monitor the health state of deployed applications by possibly raising suitable alerts, backup and restore applications in case of failures, change deployment configurations and allocated resource profiles.

Second, CISCO FogDirector enables managing Fog infrastructures by continuously monitoring installed devices (with respect to consumed RAM and CPU, instantaneously and over time), whilst abstracting from more technical networking aspects. It also permits tagging devices and searching for them to perform informed management decisions at runtime.

Last but not least, CISCO FogDirector permits rapid adoption from new users as well as integration with existing or custom management systems. To this end, CISCO FogDirector provides a GUI to offer an intuitive user experience that fits in with operational and administrative processes and practices. CISCO FogDirector also exposes a RESTful API to perform all aforementioned operations. Such API enables the interaction with existing management scripts, and it can be exploited to implement new automated

management systems, which implement customised management policies to orchestrate applications and infrastructure devices.

3. Motivating Scenario

Consider a holiday resort with 150 bungalows dispersed over a large area. The resort currently has 20 access points deployed to offer Internet connectivity to its guests, and those access points are CISCO FogDirector-enabled. Particularly, the available infrastructure is composed of three types of devices with the specifications listed in Table 1, which also reports the average μ and the variance σ of the free CPU and RAM they feature, based on historical monitoring data. For instance, devices of type L have a total of 2500 CPU units and 1024 MB of RAM but, on average, there are 1700 free CPU units (with a variance of 500 units) and 850 MB of RAM available (with a variance of 450 MB).

		Total Resources		**CPU Usage**		**RAM Usage**	
Type	Number	**CPU**	**RAM**	μ	σ	μ	σ
L	5	2500	1024	1700	500	850	450
M	5	1500	1024	1000	400	750	450
S	10	1200	512	800	400	300	200

Table 1: Resort infrastructure.

The resort manager plans to install a *Smart Ambient* application capable of managing and properly coordinating IoT domotics systems installed at each bungalow (i.e. air-conditioning, autonomous cleaning, room lighting, fire alarm) according to commands issued and goals specified by resort guests. Sensors and actuators needed for these purposes can directly connect to CISCO FogDirector devices via various technologies.

An instance of the application – requiring on average 100 CPU units and 32 MB of free RAM – must be deployed for each room so to enable all smart ambient services. Thus, the resort manager has to deploy 150 instances to provide the new services to its customers, and she is willing to invest in extending the available infrastructure with more devices of type S so to suitably support those instances, i.e. to reach an overall uptime of at least 80%, with no alerts. Sizing of the infrastructure might also depend on the policy she will adopt for deploying and managing the application instances. To this end, she selected four candidate management policies to choose from:

- *Random-fit* which places application instances that are to be deployed anew, or migrated due to alerts, on a *random node* that can accommodate them,
- *First-fit* which places application instances that are to be deployed anew, or migrated due to alerts, on the *first node* in the list of the available devices that can accommodate them,
- *Largest-fit* which places application instances that are to be deployed anew, or migrated due to alerts, on the *node with currently more free resources* that can accommodate them, and
- *Informed Largest-Fit* which places application instances that are to be deployed anew, or migrated due to alerts, on the *node with historically more free resources* that can accommodate them, based on aggregate monitoring data (inspired by FogTorchΠ [14]).

Listing 1 shows the code of the first three policies described before and being considered by the manager. The fourth policy, i.e. the *Informed Largest-Fit*, only differs from the *Largest-Fit* because it monitors the infrastructure for a period of time and sorts the devices list based on the average free resources computed during that period, before performing its choice.
Naturally, the resort manager aims at selecting a policy that can rapidly deploy all application instances, whilst minimising overall energy consumption and alert rates, and maximising the uptime of all deployed application instances. Hence, the first questions she aims at answering are:

Q1. *Which is the best management policy among the candidate ones, with respect to overall convergence speed, applications uptime and monthly energy consumption?*

Q2. *In case the policy chosen in Q1 does not guarantee at least* 80% *of uptime without deployment alerts, how many devices of type S should she buy and install to reach such a guarantee? How much will the monthly energy consumption increase? How likely are different types of alert to happen in case the infrastructure is extended accordingly?*

As the new smart service becomes more popular, the hotel manager realises that she can offer a paid *premium replica* to resort customers, which guarantees them improved uptime in case of alerts or devices failures. Then, she would like to answer other questions:

```
def random_fit():
    _, devices =fd.get_devices()
    r =random.randint(0, len(devices["data"]) -1)
    return devices["data"][r]["ipAddress"], devices["data"][r]["deviceId"]

def first_fit(cpu, mem):
    _, devices =fd.get_devices()
    for dev in devices["data"]:
        if dev["cpu"]["available"] >=cpu and dev["memory"]["available"] >=mem:
            return dev["ipAddress"], dev["deviceId"]
    return None

def largest_fit(cpu, mem):
    _, devices =fd.get_devices()
    devices =[dev for dev in devices["data"] if dev["cpu"]["available"] >=cpu
                                                and dev["memory"]["available"] >=mem]
    devices.sort(reverse=True, key=(lambda dev: (dev["cpu"]["available"],
                                                 dev["memory"]["available"]) ))
    while len(devices) ==0:
        _, devices =fd.get_devices()
        devices =[dev for dev in devices["data"] if dev["cpu"]["available"] >=cpu
                                                and dev["memory"]["available"] >=mem]
        devices.sort(reverse=True, key=(lambda dev: (dev["cpu"]["available"],
                                                     dev["memory"]["available"]) ))
    largestFit =devices[0]
    return largestFit["ipAddress"], largestFit["deviceId"]
```

Listing 1: Example management policies.

Q3. *How many premium replicas can she offer with the infrastructure obtained as per the answer to Q2, when employing the policy chosen in Q1?*

Q4. *How many new devices of type S should she purchase and have installed at the resort so to suitably support a premium replica for each application instance for each bungalow (viz.,* 300 *instances overall)? How much will this impact on energy consumption and on alerts?*

Then, the resort manager might have some considerations related to device failures. Indeed, she might want to answer the following:

Q5. *Assuming that devices can fail (e.g., crash with probability 0.5% and successfully reboot with probability* 40%*), how much will this affect the chosen management policy? How many new devices of type S should be added to keep the probability of uptime without alerts around* 80%*? How likely are different types of alert to happen in case the infrastructure is extended accordingly?*

Finally, the resort manager would like to assess what happens when deployed application instances undergo a higher workload phase, with a probability of 0.2. Last but not least, she might want to answer the following:

Q6. *Assuming that devices can fail as before and that 20% of deployed*

applications are subject to an intensive workload, how much will this affect the chosen management policy? How many new devices of type S should be added to keep the probability of uptime without alerts around 80%*? How likely are different types of alerts to happen in case the infrastructure is extended accordingly?*

In Section 5, after discussing the validation of our prototype simulator of CISCO FogDirector management, we will use it to answer the questions raised by the hotel manager.

4. Architecture and Implementation of FogDirSim

In this section, we describe the architecture and the implementation[2] of our prototype simulation environment for CISCO FogDirector application management, FogDirSim. The prototype permits to directly simulate the execution of CISCO FogDirector management scripts, based on the well-established and industry-supported RESTful API exposed by the actual CISCO tool[3].

4.1. Bird's Eye View

As aforementioned, CISCO FogDirector exposes a RESTful API to manage both infrastructure devices and applications running on them. As sketched in Figure 1, FogDirSim exposes the same API of CISCO FogDirector and it enables its users to run their management scripts against a probabilistic modelling of the available infrastructure, based on monitored data on both device resource availability and failures. The probabilistic modelling exploited by FogDirSim is therefore capable of capturing the possibility that infrastructure conditions change independently from the considered application management policy. For instance, this can happen as a consequence of increasing or decreasing devices workload, hardware or network failures and concurrent management operations issued by other clients.
By relying on FogDirSim, application operators can validate, assess and compare different management policies for their applications, before enacting them in production environments.

[2] FogDirSim is publicly available at `https://github.com/di-unipi-socc/FogDirSim`.

[3] All information about CISCO FogDirector functioning is taken from the online documentation [15] and from the official user guide [16].

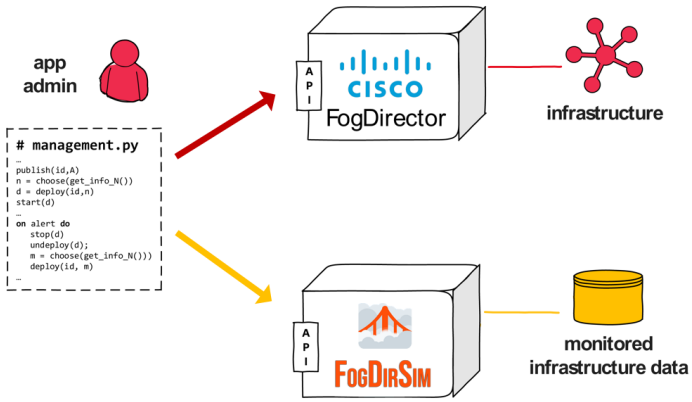


Figure 1: Bird's-eye view of FogDirSim.

On one hand, based on a set of well-defined key performance indicators (KPIs) that are predicted by the simulation environment, application operators can actually fix and refine their management policies. For instance, they can select the best management script among a set of candidates, by simulating them for a statistically significant number of times and by choosing the one script that can guarantee the best application(s) performance (e.g., higher uptime, lower energy consumption) on average. On the other hand, they can also experiment with their management, whilst varying the infrastructure capacity and/or the number (and resource profiles) of deployed applications. This enables FogDirSim users to evaluate, for free and beforehand, the impact of new investments or business-related strategic moves by means of *what-if analyses* [17]. For instance, by simulating different scenarios, one can best tune the sizing of a given infrastructure so to support a specific uptime for all the applications she will have to manage.
As we will detail later in this section, for any simulated management script, FogDirSim can

- *quantitatively* predict the probability of applications uptime and downtime, the probability of deployed applications to raise different types of alerts, the energy consumption due to management choices, and

- *qualitatively* estimate its convergence speed, its robustness against varying device failure rates, and its capacity to adapt to changing application workloads.

Being designed in a modular fashion, FogDirSim can be further extended to support computation of other quantitative or qualitative metrics, e.g. financial operational costs. In the next sections, we will detailedly describe the model and architecture of our prototype.

4.2. Architecture Overview

FogDirSim is organised into a microservice-based architecture, composed of a set of independently deployable services interacting via RESTful APIs. As shown in Figure 2, FogDirSim includes four main microservices, which are described hereinafter.

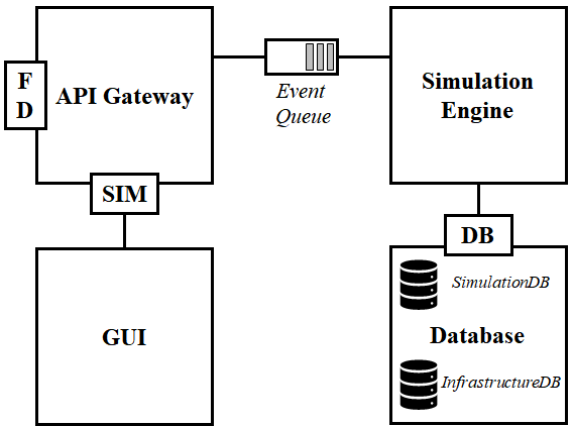


Figure 2: Microservice based architecture of FogDirSim

API Gateway It is implemented with the Python Flask micro-framework, and it exposes an API (FD) which is compliant[4] with a feature-complete subset of the actual CISCO FogDirector API (FD). The API Gateway[5] can, therefore, receive HTTP requests from existing application/infrastructure management scripts and suitably respond to them. In addition

[4]As further discussed in Section 5, FogDirSim has been validated against the official CISCO FogDirector sandbox. However, due to limitations in the testbed functionalities, it has not been possible to fully reverse-engineer all functionalities of the tool. For those cases, we referred to the Fog Director Reference Guide [16] and we completed the API accordingly.

[5]The implemented API is described at `https://github.com/di-unipi-socc/FogDirSim/wiki/API-Documentation`

to the CISCO FogDirector API, this micro-service exposes also a custom API (SIM) to retrieve data about the current simulation. The API Gateway microservice transmits its requests to the Simulation Engine microservice via an event queue.

Database The Database microservice manages two non-relational databases implemented with MongoDB. The first – *InfrastructureDB* – contains all the information about the monitored infrastructure devices and the probabilistic distributions of their available resources (viz., CPU, RAM). The second – *SimulationDB* – keeps track of the current state of the simulated infrastructure resources and application deployments.

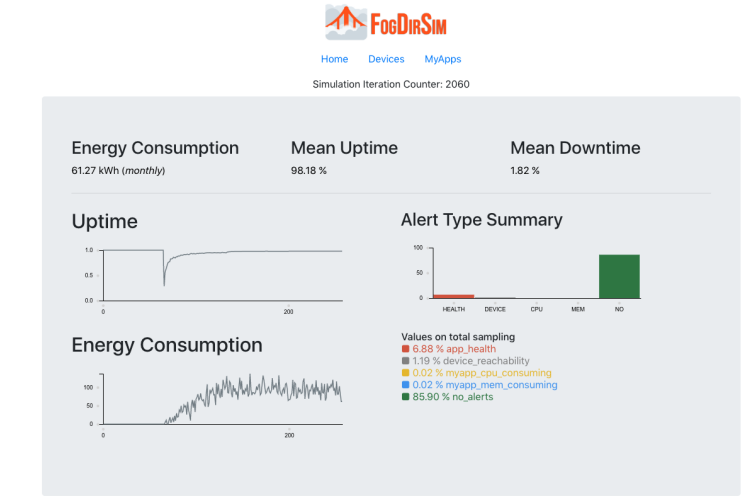
Simulation Engine It is a discrete-event simulator, implemented in Python. It executes a simulation loop that (*i*) samples a particular state of the infrastructure and of the application's workload, according to the probability distributions in the Database microservice, (*ii*) updates statistics on devices and applications status and checks for alerts triggered in the sampled state, (*iii*) pops the next client request to be simulated from the event queue and executes it.

GUI It is a Web-based GUI (Fig. 3) which shows the KPIs related to the currently running simulation. Particularly, the *Homepage* shows system-wise aggregate simulation results (Fig. 3(a)), the Devices tab details per device statistics (Fig. 3(b)), and the *Apps* tab details information on each deployed appplication (Fig. 3(c)). It is also in charge of aggregating data and of computing some of the results output by the simulator.

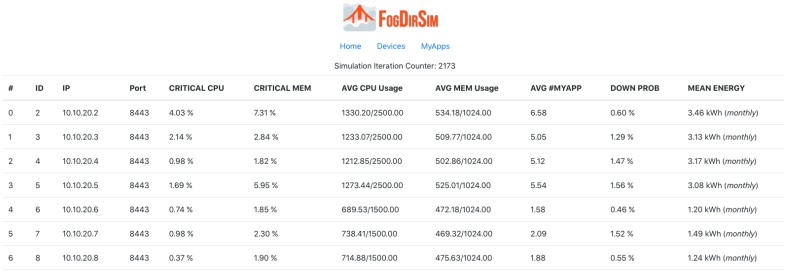
In the following, we focus on describing the implementation of the Simulation Engine microservice, by describing the underlying model and the KPIs it can compute in the current release.

4.3. Simulation Model and Implementation

The Simulation Engine is the core of FogDirSim, acting as a discrete-event simulation (DES) controller [18] for our prototype. As mentioned before, it implements a loop that pops operations to be executed from the event queue and runs them against a probabilistic varying model of the infrastructure. By simulating CISCO FogDirector behaviour, it permits to perform devices and applications management, as well as monitoring and troubleshooting of both

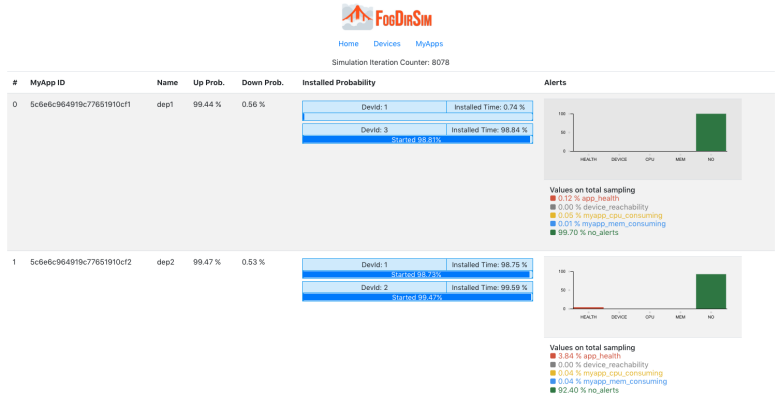


(a) Homepage



#	ID	IP	Port	CRITICAL CPU	CRITICAL MEM	AVG CPU Usage	AVG MEM Usage	AVG #MYAPP	DOWN PROB	MEAN ENERGY
0	2	10.10.20.2	8443	4.03 %	7.31 %	1330.20/2500.00	534.18/1024.00	6.58	0.60 %	3.46 kWh (*monthly*)
1	3	10.10.20.3	8443	2.14 %	2.84 %	1233.07/2500.00	509.77/1024.00	5.05	1.29 %	3.13 kWh (*monthly*)
2	4	10.10.20.4	8443	0.98 %	1.82 %	1212.85/2500.00	502.86/1024.00	5.12	1.47 %	3.17 kWh (*monthly*)
3	5	10.10.20.5	8443	1.69 %	5.95 %	1273.44/2500.00	525.01/1024.00	5.54	1.56 %	3.08 kWh (*monthly*)
4	6	10.10.20.6	8443	0.74 %	1.85 %	689.53/1500.00	472.18/1024.00	1.58	0.46 %	1.20 kWh (*monthly*)
5	7	10.10.20.7	8443	0.98 %	2.30 %	738.41/1500.00	469.32/1024.00	2.09	1.52 %	1.49 kWh (*monthly*)
6	8	10.10.20.8	8443	0.37 %	1.90 %	714.88/1500.00	475.63/1024.00	1.88	0.55 %	1.24 kWh (*monthly*)

(b) Devices



(c) Apps

Figure 3: FogDirSim GUI.

the infrastructure and the deployed applications. In addition, FogDirSim is capable of estimating KPIs related to the adopted management policy. In this section, we will describe all internals of the *Simulation Engine* – with a focus on the description of the (concurrent) finite state automata associated to each system entity – and show how they simulate (and sometimes approximate) CISCO FogDirector functioning.

4.3.1. Device Management

The finite state machine in Figure 4 shows all possible state transitions that a device can undergo in response to device management requests to CISCO FogDirector API.

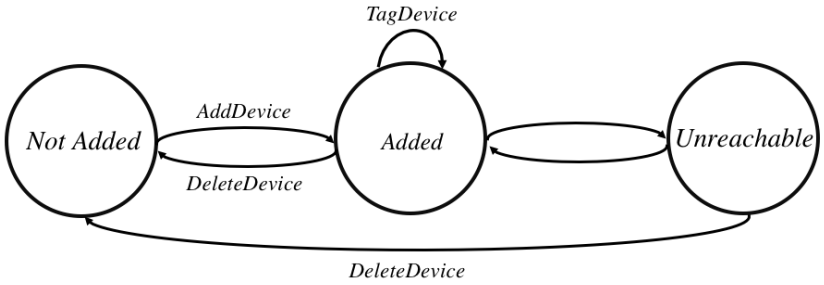


Figure 4: Device lifecycle.

In order to be managed by CISCO FogDirector, all devices must first be added to the currently running instance of the management tool. The transition ($Not\ Added \rightarrow Added$) models how a device can be added for management within an instance of CISCO FogDirector. To do so, a suitable API request has to be issued, specifying the IP address, the port and the authentication credentials of the device to be added. Conversely, the transition ($Added \rightarrow Not\ Added$) models the possibility of deleting a device from the infrastructure currently managed by an instance of CISCO FogDirector. Naturally, the device to be removed should be among the previously added ones.

CISCO FogDirector relies on a heartbeat mechanism to periodically check for device reachability. Devices can, therefore, become unreachable due to network or hardware failures. Analogously, a temporarily unreachable device can become again reachable whenever the failure is solved. Such possibilities are modelled by the unlabelled ($Added \leftrightarrow Unreachable$) transitions of the finite state machine in Figure 4. Unreachable devices can also be deleted

from the currently managed infrastructure, as modelled by the transition ($Unreachable \rightarrow NotAdded$). Finally, the added device can be tagged by assigning a symbolic label to it. This permits to retrieve (one or a set of) devices by means of a tag, thus easing device management.

Every IOx devices can run applications in a virtualised environment. When an IOx device is added to CISCO FogDirector, it shares information about its currently available hardware resources. These values refer to virtualised resources which can be allocated to application deployments. Resources are allocated at deployment time and cannot be changed afterwards unless uninstalling and installing the application again. Whenever deployed applications exceed their allocated resources, CISCO FogDirector triggers a suitable alert which can be programmatically retrieved and, possibly, processed by taking appropriate management decisions.

FogDirSim simulates the device management according to this description by exposing all the API functionalities needed to add, delete and tag devices. Additionally, to simulate the possibility of a device being unreachable, due to network or hardware failures, FogDirSim can input a failure probability ϕ and a recovery probability ρ for each infrastructure device. When the Simulation Engine samples a new infrastructure state, devices can fail (or recover) according to their specified failure (and recovery) rate following a simple Susceptible-Infectious-Susceptible (SIS) model [19].

4.3.2. Application management

CISCO FogDirector defines six states in which applications can be throughout their lifetime. The finite state machine of Figure 5 shows the transitions from one state to another. Dashed states are those in which application artefacts are not yet stored on their deployment device(s).

In order to be managed by CISCO FogDirector, applications first have to be uploaded to the current CISCO FogDirector instance. The transition ($Init \rightarrow Unpublished$) models how an application is uploaded to CISCO FogDirector. To do so, an API request has to be issued, containing the application package descriptor. Conversely, the transition ($Unpublished \rightarrow Init$) models how an application package descriptor can be removed from CISCO FogDirector. After uploading an application to CISCO FogDirector, we can publish it via the ($Unpublished \rightarrow Published$) transition, so to be able to deploy it later on. Naturally, published applications can be unpublished by issuing the operation modelled by the transition ($Published \rightarrow Unpublished$).

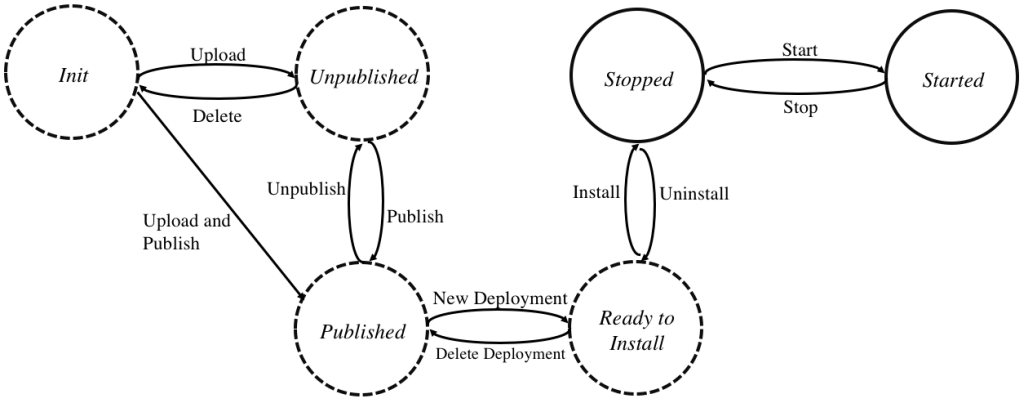


Figure 5: Application lifecycle.

Via a specific request to CISCO FogDirector API, it is also possible to upload and publish an application at a time, as modelled by the ($Init \rightarrow Published$) transition.
Once they are published, applications can be installed (viz., deployed) to a device by issuing a *New Deployment* request to CISCO FogDirector, along with the application identifier, a symbolic deployment identifier and the application deployment type (i.e., Docker, Linux container, VM). This operation is modelled by the transition ($Published \rightarrow Ready\ to\ Install$), while its dual is modelled by the opposite transition ($Ready\ to\ Install \rightarrow Published$).
A new deployment (i.e. a `myapp`) can group one or more instances (i.e. `jobs`) of a specific published application. This permits to group, for instance, all the replicas of the same service under a common identifier and, therefore, management policy. The result of creating a new deployment is a deployment identifier to be used later on (e.g. to delete a deployment).
Newly created `myapps` can be actually installed to target devices by following the transition ($Ready\ to\ Install \rightarrow Stopped$). As a result, the application is uploaded on these target devices and it is ready to be started, by triggering the ($Stopped \rightarrow Started$) transition via a suitable API request. Application deployments can be stopped again as per the ($Started \rightarrow Stopped$) transition, and uninstalled as per the ($Stopped \rightarrow Ready\ to\ Install$).

FogDirSim is able to simulate the CISCO FogDirector API for application management by supporting all the previously described API operations. Application can be published (or unpublished) through suitable requests that

add (or remove) application packages to (or from) the *applications* collection in the *SimulationDB*. Published applications can be deployed to managed devices by first creating a new deployment (`myapp`) via a request that contains the application identifier and a symbolic name for the deployment. Afterwards, application instances (`jobs`) can be installed to one or more devices by specifying a reference `myapp`, the deployment devices and the number of resources to be allocated[6] to each `job`.

Installation requests succeed if the currently sampled infrastructure state features enough resources to support the application instance at the chosen deployment device. In case it is not possible to deploy a `job`, a 400 HTTP error is returned by FogDirSim. Naturally, after installing a `myapp`, it can be started and stopped through suitable requests.

A priori estimating the workload and (consequently) the amount of resources needed by a deployed application to run smoothly is a non-trivial task to accomplish. To this end, FogDirSim introduces the possibility of specifying different application *stress profiles* with respect to the allocated resources. *Stress profiles* are modelled as probabilistic samplings that determine the specific amount of RAM and CPU required by each deployed application within its allocated amount (i.e., the current application workload). Currently[7], FogDirSim considers three possible *stress profiles*:

- *Quiet*, the stress profile that consumes a small amount of resources with respect to the amount defined in the application package, i.e., 70% of the resources declared in the application descriptor, with a standard deviation equal to 5%,
- *Normal*, the default stress profile that consumes an average amount of resources with respect to the amount defined in the application package, i.e., 80% of the resources declared in the application descriptor, with a standard deviation of 10%,
- *Intensive*, the stress profile that models an application with a currently very high workload, i.e., 90% of the resources declared in the application

[6]Despite Fog Director API permits to install multiple `jobs` related to the same `myapp` on a single device, our simulator follows the best practice enforced by Fog Director's GUI that only permit to deploy `jobs` of the same `myapp` to distinct devices.

[7]Embedded *stress profiles* profiles are a customisable parameter of the Simulation Engine. Indeed, users can tune them from a configuration file according to their needs.

Alert Type	Description
`status`	The application has a state mismatch (e.g., it is expected to be running but it is uninstalled) due to external management operations, or an install/uninstall operation failed on a device.
`App Health`	Whenever the application is running on a corrupted device or has some other issue with its health.
`App Configuration`	The application is modified outside CISCO FogDirector.
`Device Reachability`	The device on which the application is installed is not reachable.
`Memory Consumption`	During the last hour, the application has consumed more than 95% of the memory that is configured on its deployment device.
`CPU Consumption`	During the last hour, the application has consumed more than 95% of the CPU that is configured on its deployment device.
`Disk Consumption`	During the last hour, the application has consumed more than 95% of the Disk that is configured on its deployment device.
`DeviceClockSync`	The time of the device clock is ahead of the time of CISCO FogDirector.
`Read/Write Rate`	Whenever the application reads or writes data faster than the maximum rate threshold that is configured for the device.

Table 2: Alert's types and descriptions

descriptor, with a standard deviation equal to 10%.

As we will explain next, FogDirSim users are able to detect problems that may arise from the high application resource usage by programmatically retrieving the generated alerts, as in the actual CISCO FogDirector.

4.3.3. Monitoring, Alerting and KPIs

CISCO FogDirector implements a monitoring system that permits to retrieve information about the current state of the managed infrastructure and applications. Such information can be used by management scripts to make informed management decisions at runtime. The system, also, generates alerts whenever it detects a problem, an error or an unexpected behaviour in application management and infrastructure devices. Table 2 describes the main alerts that can be triggered by CISCO FogDirector, classified according to their type.

FogDirSim simulates the monitoring of the managed devices and applications by sampling both available resources and applications status profiles. The Simulation Engine of FogDirSim tracks the system state and generates alerts whenever a criticality is detected. The current version of FogDirSim only

considers a subset of all possible alerts that can be raised by CISCO FogDirector. We will detail in the next paragraphs the simulated monitoring and alerting system of FogDirSim for both devices and applications.

- **Devices Monitoring -** At every cycle the Simulator Engine samples the available resources for every device (i.e., according to the CPU/RAM device distributions and by subtracting the allocated application's resources). Such values are used in the current run and, moreover, averaged with all previous samplings to compute the average resources availability for every managed device. Furthermore, for all devices the simulation keeps track of the average number of application it hosts, of the probability that it is in a critical resource state (i.e. CPU or memory allocated is more than the available resources given the current sampling) and the failure state (i.e. when the device is not reachable, resulting from the probabilistic sampling given by the failure probability).

- **Applications monitoring -** For every application instance (i.e., a `job`), the Simulation Engine samples the resources usage (as per its stress profile) and checks if the allocated resources on the deployment devices are used more than the 95%. Whenever this happens, a `Memory Consumption`/`CPU Consumption` alert is triggered by the simulation. FogDirSim also checks for device failures. If the deployment device is in a failure state, a `Device Reachability` alert is triggered[8].

  Overall, FogDirSim is capable of simulating all types of alerts that can be triggered in response to bad management policies or resource allocation choices, and varying infrastructure or workload conditions, viz. `App Health`, `Device Reachability`, `Memory Consumption` and `CPU Consumption`. Based on this, FogDirSim computes the probability that each deployed application undergoes different alert types, by averaging them on the number of simulation cycles.

In addition to the metrics described up to now, FogDirSim is capable of computing two additional KPIs that CISCO FogDirector does not consider but

[8]Since CISCO FogDirector fires alerts from the application viewpoint, a device with resource deficiency triggers an application alert for every application it hosts.

which are useful to evaluate and assess different management policies. They are the application up- and downtime and the devices energy consumption of managed devices.

- **Uptime and Downtime -** As discussed before, CISCO FogDirector permits to group different application instances (i.e., `jobs`) under the hood of some deployment (i.e., `myapps`). FogDirSim extends the CISCO FogDirector API so to permit to specify a minimum number of `jobs` that must be up and running in order to consider a given deployment up and running. A `job` is considered up and running wherever it is *started* and its device is *reachable*.

  At each simulation cycle, FogDirSim checks the number of `jobs` that are up and running for each `myapp` and compares it with the minimum value the user specified[9] so to determine whether the `myapp` can be considered up or down. The overall average of `myapp` uptime percentage constitutes a measure of the overall system uptime, under specific management.

- **Energy Consumption -** Whilst device power consumption is often modelled as a function which linearly depends on CPU (or memory) utilisation (e.g., [20], [21]), FogDirSim permits to its users to input arbitrary energy consumption functions for each available device.

  Based on such functions, FogDirSim estimates per device and overall energy consumption of a CISCO FogDirector-managed infrastructure under the effects of a particular management policy. Whenever no energy function is specified, FogDirSim leverages the default one[10] shown in Listing 2. Energy consumption functions should return the expected instantaneous energy consumption[11] in Watt.

[9] In order to maintain the compatibility with the original CISCO FogDirector API, the minimum number of `jobs` is an optional parameter. In case it is not specified, FogDirSim consider an application up and running only if all the application `jobs` are up and running.

[10] Default behaviour is based on CISCO 800 Series Integrated Service Router - `https://www.cisco.com/c/en/us/products/collateral/routers/800-series-routers/data_sheet_c78-519930.html`

[11] Naturally, FogDirSim can be extended to account for other operational costs (e.g. financial) based on similar computations.

```
def default_function(cpu_usage, mem_usage):
    if cpu_usage <425:
        return 5
    if cpu_usage <850:
        return 10
    if cpu_usage <1275:
        return 18
    return 25
```

Listing 2: Default energy consumption function.

At each simulation cycle, FogDirSim computes the energy consumption of the entire managed system by computing the energy consumed by all devices, and by subtracting from such value the energy consumed by the sampled infrastructure alone, i.e., without managed applications. Based on the average of such values, the simulator estimates the monthly system energy consumption due to application management in kWh.

5. Experiments

In this section we briefly discuss the validation of FogDirSim (Section 5.1) and we use the prototype to solve the smart resort motivating example of Section 3 (Section 5.2).

5.1. Prototype Validation

The FogDirSim prototype described before has been validated against an actual instance of CISCO FogDirector, which is made available by CISCO in a sandboxed environment[12]. The sandbox permits to instantiate two virtual devices running IOx (on top of which it is possible to deploy applications) and one virtual device running CISCO FogDirector.

For each API operation available in FogDirSim, tests were written to compare its behaviour with that of the sandboxed instance of CISCO FogDirector. By doing so, results from all the following operations of FogDirSim have been successfully compared against those obtained in the CISCO sandbox:

- adding, removing, tagging and getting a device(s),
- creating and retrieving (device and deployment) tag(s),

[12]CISCO DevNet Sandbox – `https://developer.cisco.com/site/sandbox/`

- uploading and deleting local applications,
- publishing and unpublishing applications,
- creating and deleting `myapps`,
- retrieving generated alerts[13], and
- users login and logout.

Overall, the test suite[14] consists of 43 actual test requests, which were all successfully passed.

5.2. Motivating Example Continued

To answer her questions, the resort manager decides to run various $10,000$ epochs[15] simulations in FogDirSim. First, she aims at selecting the best management policy among the candidate ones and to understand whether such policy can guarantee a good uptime with no alerts (i.e., at least 80%) for 150 application instances. Hence, she simulates all policies on the available infrastructure made from 20 Cisco IOx devices (see Table 1), launching and managing 150 application instances. Throughout this section, but where differently specified, we assume that application instance undergo a *Normal* stress profile.

Fig. 6 shows how the overall system uptime evolves throughout the first $10,000$ simulation epochs. Despite converging more slowly – due to the involved monitoring phase it performs – the *Informed Largest-Fit* policy proves to achieve the highest average uptime probability for all the deployed instances. Also, when considering such uptime probability multiplied by the probability of not having applications under alerts, the *Informed Largest-Fit* policy outperforms the *First-Fit*, *Largest-Fit* and the *Random-Fit*) policies, as plotted in Fig. 7. Clearly, *Informed Largest-Fit* is the management policy of choice for the resort manager (what answers question **Q1**) but, unfortunately, it barely reaches a probability of 75% uptime without alerts. The

[13]The sandbox only permits to raise `Application Status` alerts, which we also used to derive the JSON format of all other alert messages described in CISCO FogDirector manual [16].

[14]The tests suite is publicly available at `https://github.com/di-unipi-socc/FogDirSim/tree/master/tests`.

[15]Each epoch corresponds to one cycle of simulation.

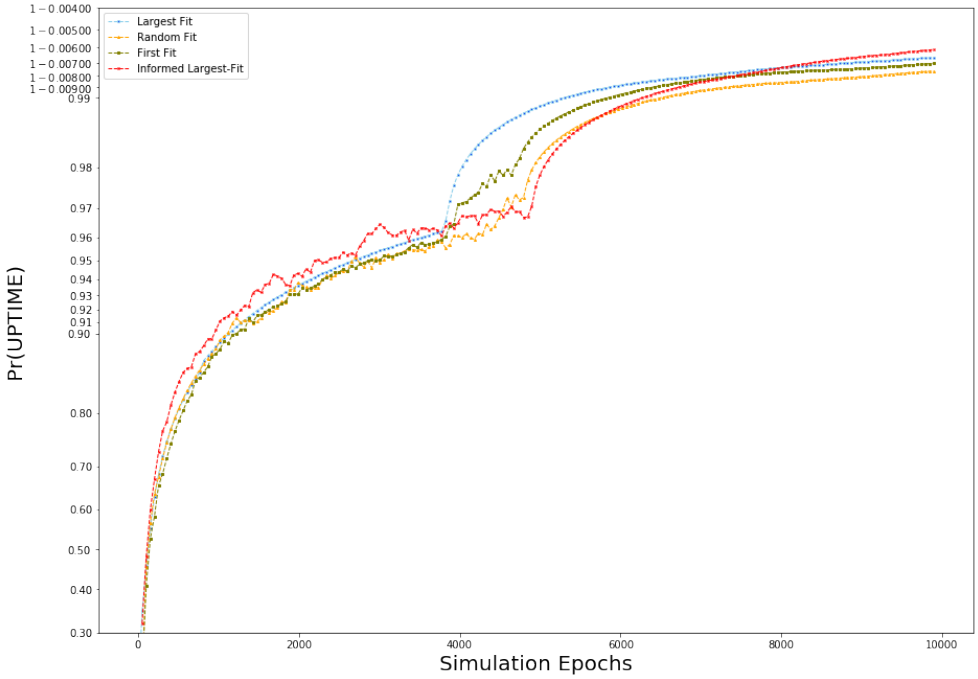


Figure 6: Average uptime over 10,000 simulation epochs (20 devices, 150 applications).

overall management energy consumption of such management settles around 100 kWh per month.

Then, with the goal of answering question **Q2**, the hotel manager starts increasing the number of available nodes of type S, still running the *Informed Largest-Fit* policy and attempting the deployment of 150 applications. She soon realises that 80% of uptime without alerts can be reached by purchasing 15 new devices to complement the infrastructure up to 35 nodes overall. Fig. 8 shows how the average uptime without alerts and management energy consumption evolve at varying the number of application instances from 150 to 290. In the simulated scenario, by adding 15 devices of type S, with 150 application instances running, FogDirSim estimates the probability that an application raises a `App Health` alert at around 15%. Other alert types are not raised in this scenario, as they depend on infrastructure failures or application workload variations, which are not considered in this first step. The overall management energy consumption of such management slightly increases up to 103 kWh per month. Overall, this answers question **Q2**.

It is worth noting that this setting can also accommodate 40 *premium*

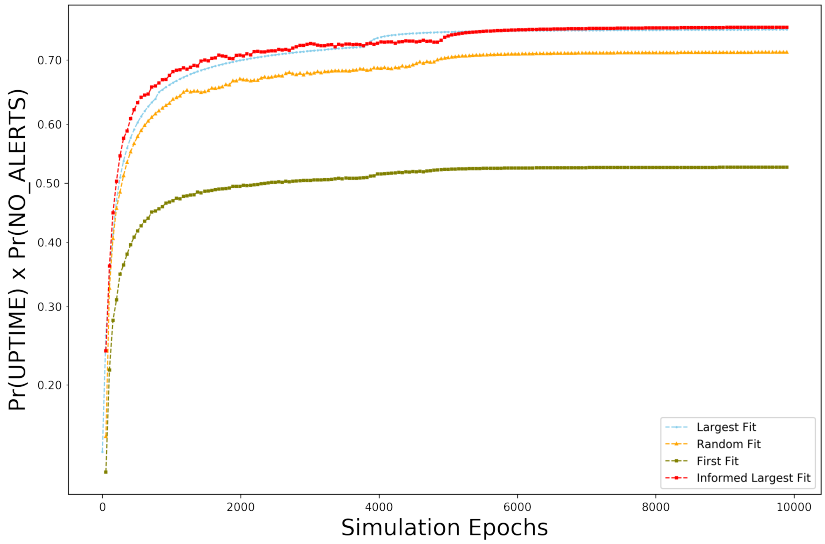


Figure 7: Average uptime with no alerts over 10,000 simulation epochs (20 devices, 150 applications).

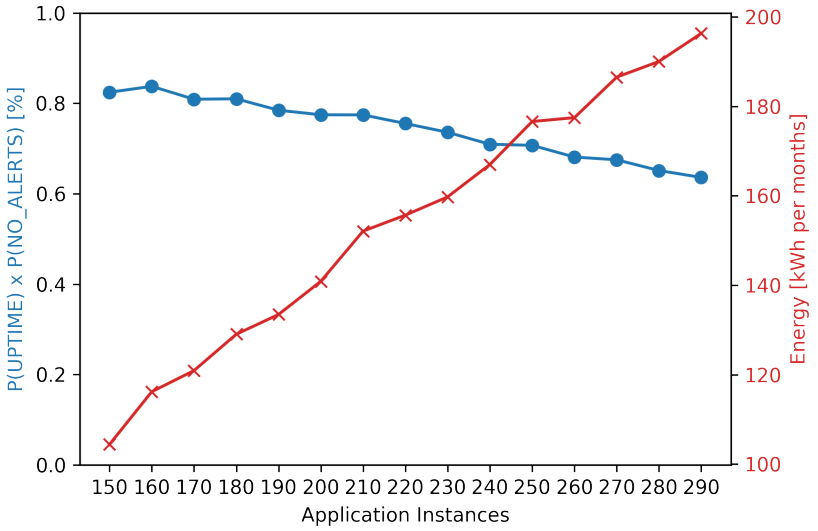


Figure 8: Average uptime with no alerts and energy consumption when varying the number of application instances (after 10,000 epochs) with 35 devices.

replicas (i.e., up to 190 overall) of the application with suitable uptime without alerts (i.e., 80%) with an estimated monthly management energy con-

sumption of 134 kWh (i.e., +34 kWh), which answers **Q3**. Deploying 190 application instances does not substantially changes the probability of raising a `App_Health` alert, which settles around 16%. Indeed, when trying to deploy more than 190 applications mainly the uptime of the system undergoes a considerable decrease, due to the repeated attempts to migrate applications unsuccessfully from one node to another.

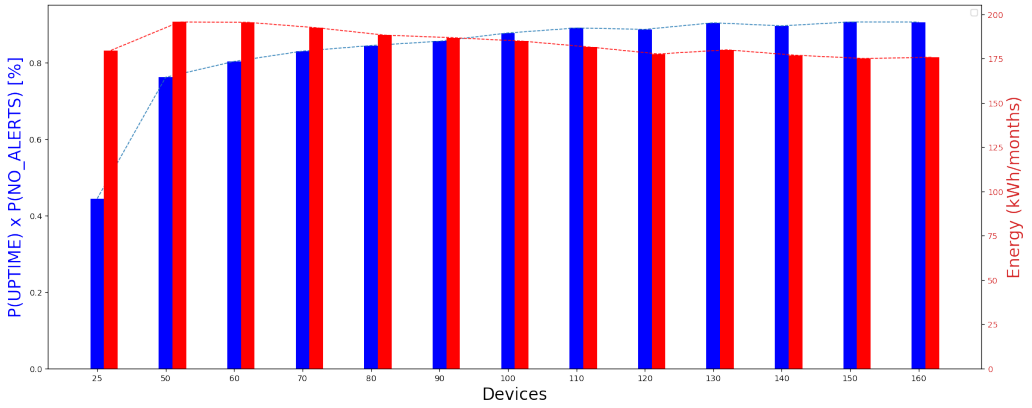


Figure 9: Average uptime with no alerts and energy consumption when varying the number of available devices with 300 deployments (10,000 epochs).

Since – according to simulation results – the 35 devices infrastructure cannot support more than 290 application instances, as the available system resources are insufficient, the resort manager investigates the possibility of further extending the available infrastructure with devices of type S, so to answer question **Q4**. Fig. 9 shows how the probability of uptime without alerts and the management energy consumption change as the number of devices varies within 25 and 160.

When incrementing the number of devices, the total amount of management energy consumption slightly decreases since devices are less overloaded (and thus consume less power), whilst uptime with no alerts increases. With 70 devices (+50 devices of type S), the uptime is greater than the 80% and, requires approximately 192kWh of management power consumption per month. On the other hand, extending the infrastructure by introducing more devices leads to consuming more device idle power, which is not considered in management power consumption. For instance, adding the 50 devices of type S and considering an average of 800 CPU units used, results in a monthly

increment of 125 $ per month[16] overall. Extending the infrastructure to 70 devices shows a reduced probability of `App Health` (around 9%). All these considerations answer question **Q4**.

Subsequently, in order to answer question **Q5**, the hotel manager sets a failure rate of 0.5% and a recovery rate of 40% for all the 70 devices that she plans to install. As shown in Fig. 10, considering device failures in the settings with 70 devices, considerably reduces the probability of system uptime without alerts at around 70%. By running further simulations and increasing the number of devices of type S, the hotel manager can realise that she will need to consider extending the infrastructure to 110 devices so to reach the set goal of 80%. Fig. 11 shows the probability of different alert types in this scenario. It is worth noting that even though `App Health` alerts become less likely (i.e., 7%), new alerts of type `Device Reachability`, due to failures, reach a probability of 1%.

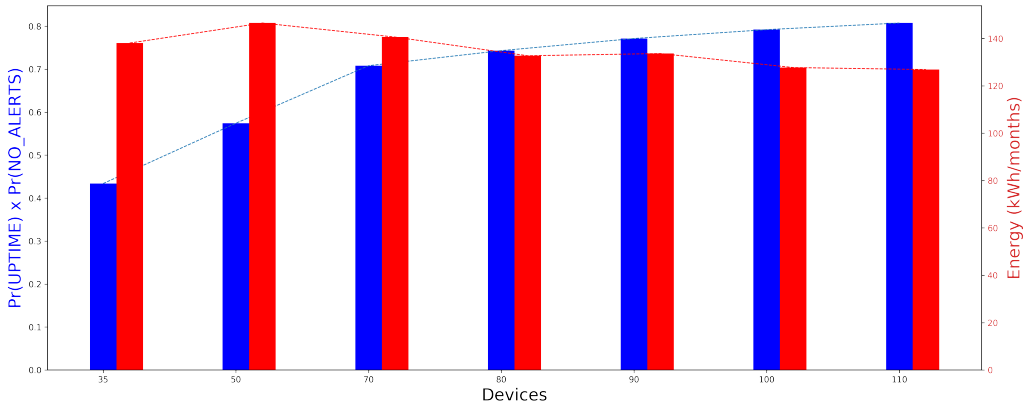


Figure 10: Average uptime with no alerts and energy consumption when varying the number of available devices with 300 deployments and device failures (10,000 epochs).

Finally, the hotel manager includes the possibility for a part of the application to be (re-)started with an *Intensive* stress profile, representing a higher workload, so to answer question**Q6**. As a result, the previous extension to 110 devices (+90 of type S), the uptime with no alerts decreases below 80% whilst overall management power consumption increases above 160 kWh per month. The hotel manager soon realises that there is a need to buy another 10 devices of type S, so to have an infrastructure made from 120 devices,

[16]We assume that 1 KWh costs 0.25 $.

which can support an uptime without alerts of 80%, with a management power consumption of around 140 kWh (+250 $ for idle consumption), and the probability of different alert types of Fig. 13.

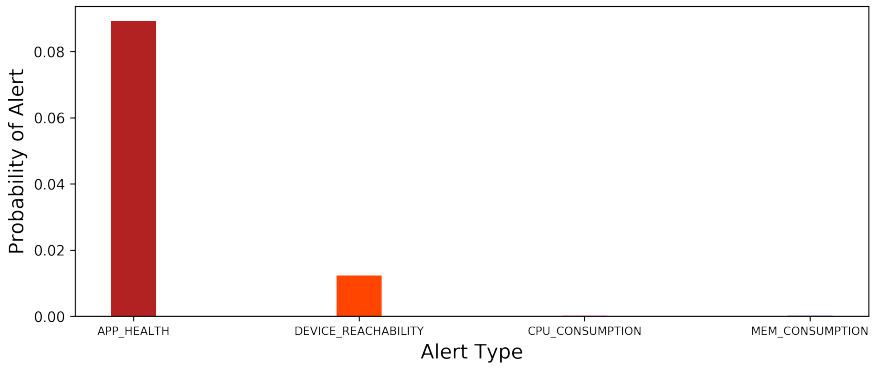


Figure 11: Alert types (after 10,000 epochs) with 110 possibly failing devices and 300 application deployments.

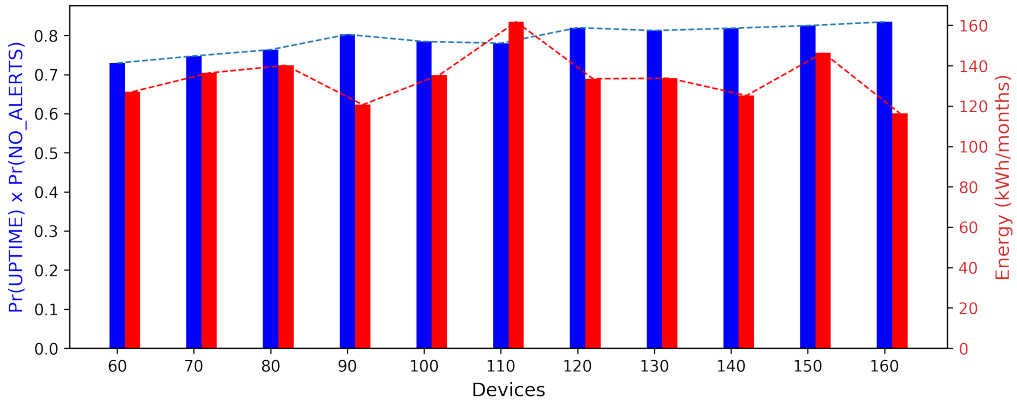


Figure 12: Average uptime with no alerts and energy consumption when varying the number of available devices with 300 deployments, device failures (10,000 epochs) and intensive stress profiles for 20% of the applications.

Such final scenario – requiring the purchase of 100 additional devices of type S – can suitably support 300 application instances (150 primary and 150 replicas), in presence of failures and applications with a high-stress profile, what answers also question **Q6** by the hotel manager.

To conclude, Table 3 summarises all the questions of the hotel manager and the answers she got from performing what-if analyses with FogDirSim.

Question	Answer
Q1. *Which is the best management policy among the candidate ones, with respect to overall convergence speed, applications uptime and monthly energy consumption?*	The best management policy is the *Informed Largest-Fit* as it shows the highest average uptime probability (without alerts) and fastest convergence speed with respect to all others. It has an estimated monthly management energy consumption of 100 kWh.
Q2. *In case the policy chosen in Q1 does not guarantee at least* 80% *of uptime without deployment alerts, how many devices of type S should she buy and install to reach such a guarantee? How much will the monthly energy consumption increase? How likely are different types of alert to happen in case the infrastructure is extended accordingly?*	To guarantee a probability of uptime without alerts of at least 80% the infrastructure should be extended with 15 new devices of type S. The estimated monthly management energy consumption grows up to around 103 kWh. `App Health` alerts are raised with a 15% probability.
Q3. *How many premium replicas can she offer with the infrastructure obtained as per the answer to Q2, when employing the policy chosen in Q1?*	The infrastructure extended with 15 new devices of type S can accommodate at most 290 application instances. However, to keep their probability of uptime without alerts at 80%, the hotel manager can offer at most 40 premium replicas (190 instances overall). The estimated monthly management energy consumption grows up to around 134 kWh. `App Health` alerts are raised with a 16% probability.
Q4. *How many new devices of type S should she purchase and have installed at the resort so to suitably support a premium replica for each application instance for each bungalow (viz.,* 300 *instances overall)? How much will this impact on energy consumption and on alerts?*	To guarantee a probability of uptime without alerts of at least 80% for 300 application instances, the infrastructure should be extended with 50 new devices of type S. The estimated monthly energy consumption grows up to around 192 kWh. `App Health` alerts are raised with a 9% probability.
Q5. *Assuming that devices can fail (e.g., crash with probability 0.5% and successfully reboot with probability* 40%*), how much will this affect the chosen management policy? How many new devices of type S should be added to keep the probability of uptime without alerts around* 80%*? How likely are different types of alert to happen in case the infrastructure is extended accordingly?*	To guarantee a probability of uptime without alerts of at least 80% for 300 application instances, in presence of failures, the infrastructure should be extended with 90 new devices of type S. The estimated monthly management energy consumption settles around 125 kWh. `App Health` and `Device Reachability` alerts are raised with a 7% and 1% probability, respectively.
Q6. *Assuming that devices can fail as before and that 20% of deployed applications are subject to an intensive workload, how much will this affect the chosen management policy? How many new devices of type S should be added to keep the probability of uptime without alerts around* 80%*? How likely are different types of alerts to happen in case the infrastructure is extended accordingly?*	To guarantee a probability of uptime without alerts of at least 80% for 300 application instances, in presence of failures and intensive workload, the infrastructure should be extended with 100 new devices of type S. The estimated monthly management energy consumption settles around 140 kWh. `App Health`, `Device Reachability`, `CPU Consumption` and `Mem Consumption` alerts are raised with a 6%, 1%, 3%, 4% probability, respectively.

Table 3: Summary of the experiments.

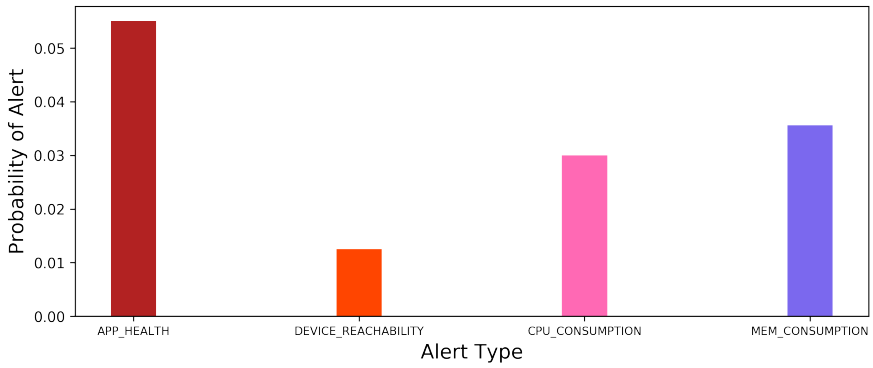


Figure 13: Alert types (after 10,000 epochs) with 120 possibly failing devices and 300 application deployments with a probability of *Intensive* stress profile of 20%.

6. Related Work

To the best of our knowledge, a few different simulation environments of Fog computing scenarios have been designed and prototyped in the last years. In the next paragraphs, we describe those which are more related to simulating Fog application management and we analyse their specificity.

iFogSim. Based on CloudSim [22], iFogSim [10] is one of the most promising tools for simulating a Fog computing environment. It models IoT, Fog and Cloud resources and measures the impact of resource management techniques on latency, network congestion, energy consumption and monitoring cost. Its architecture is composed of several layers. The first layer is the Sensors Layer, responsible to simulate data generation according to customised patterns. The second layer is the Fog Devices Layer, which simulates the Fog and Cloud resources. The devices are organised hierarchically and only vertical connection can be done. Finally, the Monitoring Layer is the core of the simulator and it manages resources of the environment in order to meet the QoS constraints with the available resources. The current version of iFogSim provides a static application placement policy. Every application is modelled as a collection of modules that can be statically installed over devices. The simulation model adopted by iFogSim is a Discrete-Event Simulation (DES).

EdgeCloudSim. As iFogSim, EdgeCloudSim [23] is an extention of CloudSim. In addition to CloudSim device model, it features the possibility of simulating (possibly parallel) execution of application tasks.

*FogTorch*Π. FogTorchΠ [9] proposes a simple, yet general, model of multi-service IoT applications and Fog infrastructures. The authors prove the NP-hardness of the problem and find a heuristic backtracking search algorithm to solve it. They, moreover, extend the prototype in [24] by introducing a Monte Carlo simulation so to consider variations in the QoS of communication links. FogTorchΠ can be used to determine and compare eligible deployments of an application to a given infrastructure considering the QoS, the cost of deploying IoT applications to considered infrastructures [14] and the context.

YAFS. Very recently, and promisingly, Lera et al. [11] proposed YAFS (*Yet Another Fog Simulator*), a discrete-event simulation library that relies on complex network theory to support the design of IoT applications, embedding different placement and routing strategies. Similarly to FogDirSim, YAFS models potential failures of the available devices, but not the possibility for them to recover from failed states. Also, their modelling accounts for the mobility of users and the associated workload, and considers network topology information, mostly focussing on communication latency and bandwidth. Being a library, YAFS does not feature a set of predefined KPI but gathers the main events in raw format, leaving its users to decide how to output aggregate data. A simulator GUI is not currently available but it can be realised by relying on Python libraries.

Other approaches. Other solutions, that include a mix of simulation, emulation and the use of a real testbed, are proposed in the literature. We report, in the follows paragraph two of them: a solution from Ficco et al. [25] and EmuFog [26]. A simulation environment is not always able to catch all the features of the environment. In order to avoid this bias, Ficco et al. [25] has proposed a mixed model in which some experimental scenario is simulated, while the edge and fog nodes under test are emulated or executed in a real environment. It introduces, of course, a different cost to the simulation since the IoT devices have to be bought and installed. The simulation part is used to reproduce the behaviour of the external system, instead, the specific devices phenomena are analyzed using a combination of emulation and real devices analysis. Similar to the previous work, EmuFog [26] tries to delete the simulation simplification that may not always be true in a dynamic infrastructure. It uses a network emulator, MaxiNet [27], to emulating the infrastructure. It permits to load the infrastructure from a file and connect the devices using ad undirected graph.

All simulation environments described up to now rely upon custom APIs to express application management policies, i.e. they require application operators to convert their (actual) management scripts into an abstract representation, compliant with the API and internal model of the simulator. Doing so clearly introduces an additional layer of complexity and might also introduce difficulties in interpreting the simulation output, requiring to compare the application and infrastructure models of the actual management tool with the ones of the simulator. A first effort towards reducing the gap between simulation and real-world tools has been carried out by Forti et al. [12] in their FogDirMime.

FogDirMime. In [12], Forti et al. proposed a simple operational semantics to model CISCO FogDirector basic functionalities, i.e. publish, deploy, configure, start, monitor, stop, undeploy and retire application instances. They implemented a first, partial, proof-of-concept library, *FogDirMime*, which, using a very high-level API can simulate CISCO FogDirector functioning and permits writing custom simulation scripts to compare different application management policies. The operational semantics of CISCO FogDirector given in [12] has been used as a first formal guideline to drive the design and implementation of FogDirSim. Still, FogDirMime prototype shows important limitations. In addition to modelling a much smaller (and high-level) subset of CISCO FogDirector functionalities with respect to FogDirSim, FogDirMime is not able to directly input actual management scripts, nor it provides the possibility to set configuration parameters for the simulation to take place. Last but not least, FogDirMime proof-of-concept leaves the users alone with the tasks of writing their own simulation engine and of deciding which KPIs to compute and how to compute them, and it does not provide any GUI.

7. Concluding Remarks

In this paper, we illustrated FogDirSim, a novel simulation environment of CISCO FogDirector management system. We then discussed validation of the prototype and showed its usability and usefulness over a lifelike experiment from the world of smart-buildings and domotics.

To the best of our knowledge, FogDirSim is the first prototype capable of simulating actual application management policies, written by relying on the RESTful API of (and therefore fully compliant with) an industrial Fog computing management platform (viz., CISCO FogDirector). Our prototype

constitutes a first – feature-complete and extensible – framework to validate, assess and compare different management policies against a probabilistic description of the available (or target) Fog infrastructures.

Indeed, it tames the scale and complexity of Fog infrastructure, and it permits simulating different CISCO FogDirector application management scripts so to:

- validate their correct functioning within different simulated infrastructure workload conditions and against (network or hardware) failures, and
- predict KPIs (viz., uptime, energy consumption, resource usage, type of alerts) of the effectiveness of the management policies they implement.

Tools like FogDirSim are expected to be extremely useful to application operators in Fog computing scenarios. Actually, application operators can freely exploit our prototype to improve and refine their management scripts before using them in production environments by following a write-simulate-refine feedback loop.

For instance, FogDirSim can be used to help in deciding when and where to migrate a certain application, how to best handle failures and node workload variations, how many application replicas are needed to achieve a suitable uptime, how to reduce energy consumption due to a bad management policy. Also, it enables application operators evaluating – for free and beforehand – the impact of infrastructural upgrades or expected changes by performing *what-if analyses* over simulated scenarios. For instance, employing amortised costs modelling, it would be possible to complement FogDirSim analysis with an analysis of the costs related to the purchase of new (and possibly diverse) devices and to their future maintenance so to take the best strategic decision in the considered business scenario.

Overall, FogDirSim contributes to supporting the design of correct and effective management policies for Fog computing applications, whilst reducing the overhead required to application operators by other approaches to translate management scripts according to a particular simulation model and API.
In our future work, it would be interesting to:

- refine the simulation model so to account for the relative duration of simulated operations, which would permit to more accurately estimate

the period of stay of a deployment in each different state and, consequently, the output KPIs,

- improve the probabilistic modelling of infrastructure variations so to introduce some form of correlation among consecutively sampled events (e.g., by relying on Markov chain models as in [28]), or to be able to simulate a specified series of those variations as well as time-related KPIs (e.g., application response time) so to offer support for time-sensitive applications (e.g., healthcare emergency),

- refine the implemented API against a production-ready instance of Fog Director over which it is possible to run actual IOx applications, and possibly getting from CISCO more complete documentation of the RESTful API of the tool,

- test and assess the prototype against an actual use case infrastructure and applications set, which are currently under development at our University, and use it to compare other existing placement and management policies [29].

Acknowledgements

This work has been partly supported by the project "*DECLWARE: Declarative methodologies of application design and deployment*" (PRA_2018_66) funded by University of Pisa, Italy, and by the project "*GIÒ: a Fog computing testbed for research & education*", funded by the Department of Computer Science of the University of Pisa, Italy.

References

[1] F. Bonomi, R. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for internet of things and analytics, in: Big data and internet of things: A roadmap for smart environments, Springer, 2014, pp. 169–186.

[2] J. Dizdarević, F. Carpio, A. Jukan, X. Masip-Bruin, A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration, ACM Computing Surveys (CSUR) 51 (2019) 116.

[3] C. S. R. Prabhu, Fog application management, in: Fog Computing, Deep Learning and Big Data Analytics-Research Directions, Springer, 2019, pp. 21–24.

[4] H. Shahinzadeh, J. Moradi, G. B. Gharehpetian, H. Nafisi, M. Abedi, Iot architecture for smart grids, in: 2019 International Conference on Protection and Automation of Power System (IPAPS), IEEE, pp. 22–30.

[5] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture (2019).

[6] A. A. Mutlag, M. K. A. Ghani, N. Arunkumar, M. A. Mohamed, O. Mohd, Enabling technologies for fog computing in healthcare iot systems, Future Generation Computer Systems 90 (2019) 62–78.

[7] C. Yu, B. Lin, P. Guo, W. Zhang, S. Li, R. He, Deployment and dimensioning of fog computing-based internet of vehicle infrastructure for autonomous driving, IEEE Internet of Things Journal (2018).

[8] A. N. Toosi, R. M. Q. Chi, R. Buyya, Management and Orchestration of Network Slices in 5G, Fog, Edge, and Clouds, Wiley, pp. 79–101.

[9] A. Brogi, S. Forti, QoS-aware Deployment of IoT Applications Through the Fog, IEEE Internet of Things Journal 4 (2017) 1185–1192.

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

[11] I. Lera, C. Guerrero, C. Juiz, YAFS: A simulator for IoT scenarios in Fog computing, IEEE Access 7 (2019) 91745–91758.

[12] S. Forti, A. Brogi, A. Ibrahim, Mimicking FogDirector Application Management, Software-Intensive Cyber-Physical Systems 2–3 (2019).

[13] CISCO Fog Director, `https://www.cisco.com/c/en/us/products/cloud-systems-management/fog-director/index.html`, 2019. Accessed: June 19.

[14] A. Brogi, S. Forti, A. Ibrahim, Deploying Fog Applications: How Much Does It Cost, By the Way?, in: Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018), SciTePress, 2018, pp. 68–77.

[15] Cisco Fog Director REST API, `https://developer.cisco.com/docs/iox/#!fog-director-api-documentation/cisco-fog-director-rest-api`, 2019. Accessed: June 18.

[16] CISCO, Cisco Fog Director Reference Guide (v. 1.5), 2017.

[17] S. Rizzi, What-if analysis, in: Encyclopedia of Database Systems, Springer, 2009, pp. 3525–3529.

[18] B. P. Zeigler, T. G. Kim, H. Praehofer, Theory of modeling and simulation, Academic press, 2000.

[19] A. Gray, D. Greenhalgh, L. Hu, X. Mao, J. Pan, A stochastic differential equation sis epidemic model, SIAM Journal on Applied Mathematics 71 (2011) 876–902.

[20] S. Rivoire, P. Ranganathan, C. Kozyrakis, A comparison of high-level full-system power models., HotPower 8 (2008) 32–39.

[21] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, W. Dai, Energy efficient task allocation and energy scheduling in green energy powered edge computing, Future Generation Computer Systems 95 (2019) 89–99.

[22] R. Buyya, R. Ranjan, R. N. Calheiros, Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities, in: High Performance Computing & Simulation, 2009. HPCS'09. International Conference on, IEEE, pp. 1–11.

[23] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: An environment for performance evaluation of edge computing systems, Transactions on Emerging Telecommunications Technologies 29 (2018) e3493.

[24] A. Brogi, S. Forti, A. Ibrahim, Optimising QoS-Assurance, Resource Usage and Cost of Fog Application Deployments, in: V. Muñoz, D. Ferguson, M. Helfert, C. Pahl (Eds.), Cloud Computing and Services Science,

CLOSER 2018 Revised Selected Papers, volume 1073 of *Communications in Computer and Information Science (CCIS)*, Springer, 2019, pp. 168–189.

[25] M. Ficco, C. Esposito, Y. Xiang, F. Palmieri, Pseudo-Dynamic Testing of Realistic Edge-Fog Cloud Ecosystems, IEEE Communications Magazine 55 (2017) 98–104.

[26] R. Mayer, L. Graser, H. Gupta, E. Saurez, U. Ramachandran, Emufog: extensible and scalable emulation of large-scale fog computing infrastructures, in: Fog World Congress (FWC), IEEE, pp. 1–6.

[27] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, H. Karl, Maxinet: Distributed emulation of software-defined networks, in: Networking Conference, 2014 IFIP, IEEE, pp. 1–9.

[28] T. Ahmad, D. Truscan, I. Porres, Identifying worst-case user scenarios for performance testing of web applications using markov-chain workload models, Future Generation Computer Systems 87 (2018) 910–920.

[29] A. Brogi, S. Forti, C. Guerrero, I. Lera, How to Place Your Apps in the Fog - State of the Art and Open Challenges, Software: Practice and Experience In Press. (2019).