Fogbed: A Rapid-Prototyping Emulation Environment for Fog Computing

Antonio Coutinho*†, Fabíola Greve†, Cássio Prazeres†, and João Cardoso*
*State University of Feira de Santana, Department of Technology
†Federal University of Bahia, Computer Science Department
Email: {acoutinho, jcardoso}@uefs.br, {fabiola, prazeres}@ufba.br

***Abstract*—The fog computing paradigm extends cloud resources near data sources to overcome limitations of cloud-based IoT centralized architectures. Its involve the running of services and applications on the nodes between the IoT devices and the cloud. The prototyping and testing of these distributed software is challenging due to the fact that not only the fog service has to be tested but also its integration with management systems. Despite recent advances in fog platforms, there exists no readily available testbed which can help researchers to design and test real world fog applications. To this purpose, network simulators and cloud middleware are adapted to enable the investigation of fog solutions. This paper presents Fogbed, a framework and toolset integration for rapid prototyping of fog components in virtualized environments. Using a desktop approach, Fogbed enables the deployment of fog nodes as software containers under different network configurations. Its design meets the requirements of low cost, flexible setup and compatibility with real world technologies. Unlike current approaches, the proposed framework allows for the testing of fog components with third-party systems through standard interfaces. A scheme to observe the behavior of the environment is provided. A case study is presented to demonstrate a fog service analysis. In addition, future developments and research directions are discussed.**

I. Introduction

A cloud-based centralized model has become the default approach for the internet of things (IoT). Current proposals [1], [2], [3], [4] are promoting a major shift towards a decentralized architecture to overcome limitations of cloud IoT platforms such as mobility, location awareness and low latency.

Fog computing is an emergent paradigm that extends cloud services to the edge of the network in an integrated way with network devices such as switches, routers and IoT gateways[1]. The concept of fog computing is defined in [1] as a virtualized platform which offers services between end devices and the conventional data centers of cloud computing. This definition implies a series of challenges that make the fog environment a non-trivial extension of the cloud.

Figure 1 presents the fog system-level horizontal architecture [5]. It involves the running of services and applications concerning distributed components on the nodes between edge devices and the cloud. They are distributed hierarchically in a network infrastructure with at least three layers.

At the edge of the network there are different IoT devices. Retrieving data from each sensor device is often challenging due to sensor power and communication constraints. An IoT gateway or smart device can aggregate and translate data from sensors before sending it onward by supporting different communication protocols.

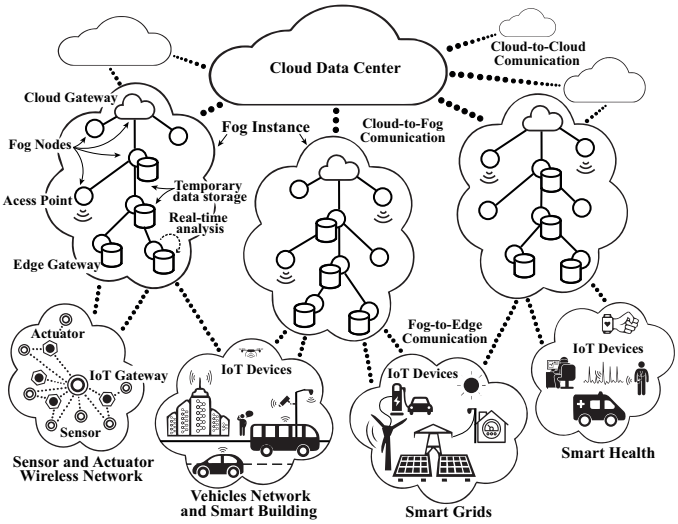

Fig. 1: The fog computing architecture.

The key components that can provide processing and storage capabilities at the edge of the network are called fog nodes [6]. These fog nodes are involved in the data processing of different IoT applications. It must favor the access of IoT devices to the network and cloud resources. Due to the heterogeneity of fog environments, the fog nodes may present different capabilities, interfaces and hardware configurations. Depending on the application, they can support advanced tasks including data caching, geo-localization, service discovery, real-time analysis, load balancing, resource management and security.

Fog instances can offer virtualized resources to support the extension of the cloud services to the edge of the network. They are also responsible for periodically filtering and sending information to the cloud. In the fog horizontal architecture, cloud gateways can act as proxy for larger cloud providers.

The management software that globally coordinates the fog infrastructure is distributed in a number of services working together. Its main purpose is to provide acceptable levels of latency while optimizing the performance of fog applications.

Very recently, fundamental works [7], [5], [8], [9] have proposed a reference architecture, design goals and components of a fog platform. In such an environment, the creation of fog services involves steps such as [10]: the development of the service functions; its integration with the system that performs

978-1-5386-3180-5/18/$31.00 ©2018 IEEE

life cycle management and orchestration; the implementation and test of management interfaces; the implementation and validation of service-specific management components.

A current problem in this area is the lack of supporting tools to prototype and test services in fog scenarios. As far as we know, there are no readily available fog testbeds that can help researchers to design and verify distributed algorithms on a truly IoT scale. At present, simulators and cloud middleware are adapted to allow the experimental evaluation of fog solutions in specific scenarios and restricted conditions.

In the development of fog components, supporting tools can reduce deployment time, save costs and improve the quality of the offered services. These tools should allow testing a fog service such as by sending sensor data through it and also validating its interaction with a management system.

This paper presents Fogbed [11], a framework and toolset integration for rapid prototyping of fog components in virtualized environments using a desktop approach. Its design meets the postulated requirements of low cost, flexible setup and compatibility with real world technologies. The components are based on Mininet [12] network emulator with Docker [13] container instances as fog virtual nodes.

In addition, an example to demonstrate how Fogbed is employed to analyse fog service latency is presented. The implemented application is a healthcare prevention and monitoring system. The proposed solution is a fog service that employs a distributed top-*k* monitoring query processing approach [14]. A preliminary comparison using this method in different scenarios is provided.

The rest of the paper is organized as follows. Section II presents related work. Section III addresses Fogbed requirements and technological choices. Section IV illustrates the emulator architecture. Section V presents a case study example. The conclusions and future works are presented in Section VI.

II. Related Work

As the pioneer in the fog computing field, Cisco solutions finds itself in a better position compared to other commercial fog products [8]. The Cisco IOx [15] is a network operational system included in routers and switches that makes processing and storage available to applications hosted in virtual machines (VMs). Distributed devices with IOx support can function as the compute environment for fog applications. The IOx also supports open source systems based on Linux instances running in a hypervisor. Together with IOx Fog Director [15], the Cisco IOx Sandbox enables development and testing of fog applications using a desktop simulation approach.

Existing work has focused on the designing and implementing of an adequate fog computing platform, which can serve as a model development environment for prototyping. In November 2015, the OpenFog consortium united leading IT-companies and universities in order to develop a standard fog architecture. The first version of OpenFog reference architecture [9] was released in February 2017.

In the absence of ready platforms, an alternative is to adapt current testbeds to support experimental evaluation of fog solutions. The largest IoT testbeds such as [16] are equipped with thousands of IoT devices geographically distributed. However, since they are not intended for fog applications, a major effort is required to configure and deploy experiments.

In [17] the performance of object store solutions in meeting the criteria expected for a fog infrastructure were evaluated. The experiments were conducted using the Yahoo Cloud System Benchmark (YCSB) on top of the Grid'5000 testbed.

Also, there are open source technologies that can be used to create small fog testbeds. For example, Soft-IoT [18] is a IoT framework that supports fog and edge computing. In [5] a face recognition application uses a prototyping fog platform consisting of two nodes implemented with OpenStack and connected to the Amazon EC2 service.

Although desirable, the use of a real world testbed is expensive, difficult to access, time-consuming and does not provide a controllable environment. Therefore, low cost-effective alternatives should be considered before a costly experiment so as to eliminate ineffective algorithms, policies and strategies.

In [19] OMNET++ and open source simulator environments that allow for the extension of their functionality are compared. OMNET++ is not a network simulator itself but an extendable library and framework for building network simulators. There are extensions for real-time simulation, network emulation, system integration, and several functions that can be used for modeling a fog environment. As far as we know, extensible solutions such as OMNET++ do not have a specific framework implementation for the fog computing model.

Several works such as [20], [21] evaluate fog model proprieties and algorithms through numerical simulations. However, this strategy does not address the problems relating to its practical implementation. To this purpose, the adaptation of existing open source frameworks or simulators is currently a common approach in the fog computing research area.

In [22], [23], [24] specific simulators to evaluate fog computing frameworks were designed. In [25] a hybrid PaaS for IoT application provisioning in fog environments was proposed. MicroPCF, a lightweight distribution of Cloud Foundry was extended to implement an IoT healthcare application.

A case study on smart traffic management in [8] extended the CloudSim [26] with fog computing capabilities to improve the performance of the application in terms of response time and bandwidth consumption. This work was the basis for iFogSim [27], a simulator that supports edge and fog resources under different scenarios in a sense-process-actuate model [27]. Also, it allows for the evaluation of resource management policies by focusing on their impact on latency, energy consumption, network congestion and operational costs.

In summary, fog applications continue to be developed as proof-of-concept, implemented in limited environments and applied to specific scenarios. This remains a default approach, given the challenges involved in the implementation of fog environments. Therefore, research concerning fog computing platforms may help to integrate cloud and IoT technologies and accelerate the development of fog applications.

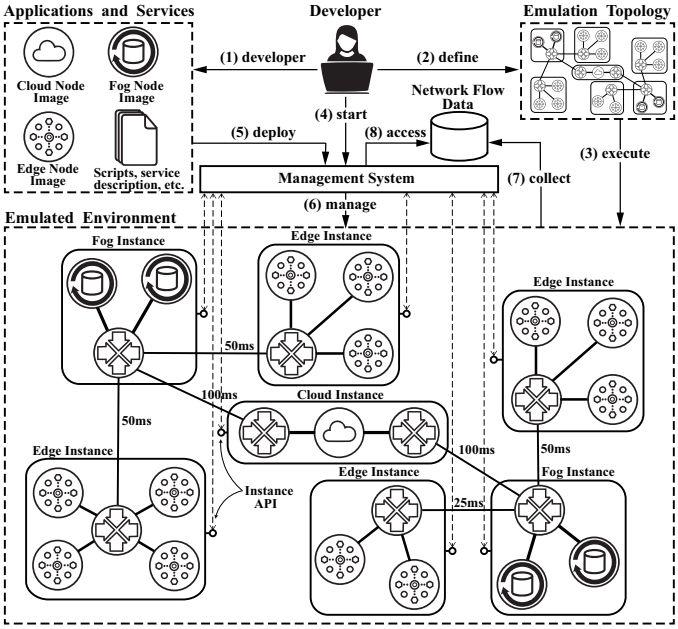

Fig. 2: Fogbed emulation workflow.

III. Fogbed Requirements and Technologies

The Fogbed architecture is based on solutions that meet the following requirements: (1) low cost deployment; (2) fast and flexible setup; (3) and support to perform real world protocols and services. In its development, proprietary solutions were not considered viable alternatives due to their high cost. Open source frameworks can be considered as cost-effective. However, some solutions do not model fog environments and IoT, where services can be deployed both on edge and cloud resources. Moreover, the use of simulators makes it difficult to support real world IoT protocols and services.

Therefore, in the remainder of this section, open source solutions that enable the deployment of the main fog components are described. They allow for the testing of real world technologies in a repeatable and controllable environment.

A. Mininet Network Emulator

Mininet [12] is a network emulator that uses the Linux kernel to create virtual hosts, switches and links. Unlike the simulators, it allows for the deployment of protocols and applications in the same way as real world testbeds. Mininet also offers native support to OpenFlow [28] for flexible custom routing and software-defined networking (SDN) [12].

Mininet virtual hosts that can run standard Linux software by using Linux namespaces. Each virtual host runs inside a separate network namespace with its own set of network interfaces, IP addresses, and a routing table. A virtual host connects to one or more virtual switches through virtual network links. The topology and link properties such as bandwidth, loss rate and delay can be configured with an extensible Python API. However, Mininet virtual hosts share the host machine file system by default. This makes it troublesome to use virtual hosts as fog nodes, since they should not use the same configuration or data files in distributed applications.

B. Docker Container Platform

Docker [13] is a software platform that allows the creation of containers. A container is a lightweight, stand-alone and executable package that includes everything needed to run an application such as executable code, runtime environments, system tools, system libraries and configuration settings.

Applications running inside a Docker container and using operating-system level virtualization on Linux are similar to traditional VM instances. However, unlike VM images, containers do not bundle a full operating system.

Docker isolates the container resources using similar technologies as Mininet does for its virtual hosts, based on Linux cgroups and kernel namespaces. They both use virtual ethernet interface pairs to connect the virtual hosts to the virtual switches. Replacing Mininet virtual hosts with Docker containers is technically feasible as demonstred in [29].

IV. Fogbed Architecture

Fogbed extends the Mininet framework to allow the use of Docker containers as virtual nodes. It provides capabilities to build cloud and fog testbeds. The Fogbed API enables adding, connecting and removing containers dynamically from the network topology. These features allow for the emulation of real-world cloud and fog infrastructures in which it is possible to start and stop compute instances at any point in time. Also, it is possible to change at runtime resource limitations for a container, such as CPU time and memory available.

A Fogbed emulation can be created by deploying virtual nodes and virtual instances into an virtual network environment running on a host machine. The flexible setup is achieved by using preconfigured container images. Each container image comprises part of a distributed application, required services and protocols. An example of a fog environment emulation is shown in Figure 2, where three types of container images were used to instantiate different virtual nodes.

1) Cloud container image: Using this type of container, cloud gateways to support virtual resources for IoT applications are emulated. These virtual cloud nodes can form a virtual cloud instance (VCI) using open frameworks or act as proxies for cloud services located in remote data centers.

2) Fog container image: Using this type of container, fog nodes capable of processing and storing data collected by the IoT gateways are emulated. These virtual fog nodes can form a virtual fog instance (VFI) to support virtualized resources for IoT devices. Also, it is responsible for analysing, filtering and sending information to a VCI.

3) Edge container image: Using this type of container, smart IoT devices or IoT gateways are emulated. In a virtual edge node, specific protocols and services to allow for communication with real sensor and actuator devices can be installed. It is possible to use simulated sensors as data sources. These virtual edge nodes can form a virtual edge instance (VEI) to emulate a sensor network and send sensing data to a VFI.

Each virtual instance involves virtual switches and virtual nodes. The management system is responsible for uploading and starting the instances in the emulated environment.

The defined topology determines the virtual connections between virtual nodes and virtual instances. The communication between a virtual instance and the management system is performed through a well-defined instance API.

A. Fogbed Workflow

Figure 2 depicts the high-level workflow of a developer using the environment. First, the developer provides the container images that will be instantiated to setup the emulation (1). Each container includes everything needed to run an application such as executable code, scripts, service descriptions and configuration settings. Second, the developer defines a topology on which he wants to test the application (2) and executes the emulator with this topology definition (3). If SDN is required, an SDN controller should be started. After the platform has been initiated, the developer starts the management system (4). The management system connects to the emulated environment by using the provided instance API. The application is deployed on the platform by the management system (5) which starts the required process and services in each virtual node. Then, the application can run inside the platform (6). The network flow statistics can be collected and stored for future analysis (7). Furthermore, the developer and the management system can access arbitrary monitoring data generated by the platform (8).

B. Design Components

The general architecture, essential components of the system, as well as the relationships between those components are shown in Figure 3. In the host machine, the Mininet is the core of the system and implements the emulation environment. The Fogbed extends the Mininet functionalities to support Docker containers as virtual nodes. At the top of the Fogbed are the framework APIs provided for developers.

The topology API interacts with the Fogbed to load and execute topology definitions and create its virtual nodes, switches, instances and connections. The instance API is flexible and allows the system to be extended with different standard interfaces. Each interface can be used by a third-party system to manage applications and services. The resource API allows the developer to apply limitation models that define the available resources for each virtual instance such as CPU time and memory capacity. In addition, the interactive command line interface (CLI) allows developers to interact with the emulated components, change configurations, view log files or run arbitrary commands while the Mininet platform executes the application and services.

The emulated environment contains virtual nodes, virtual switches and virtual connections. The Mininet process instantiates the virtual nodes from pre-created Docker container images. A container image can be instantiated more than once in an emulated environment. For example, Figure 2 shows an edge container image being used to initiate different VEIs, each capable of generating IoT sensor data.

The virtual instance is an abstraction that allows for the management of related set of virtual nodes and virtual switches

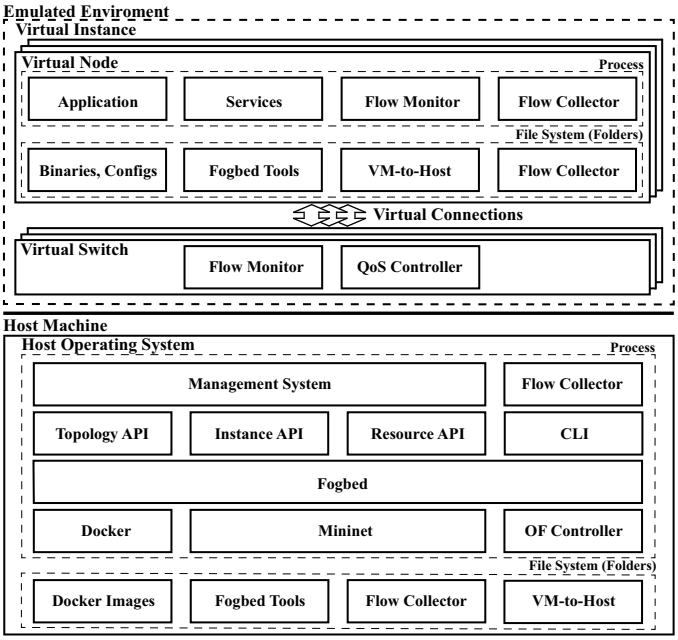

Fig. 3: An overview of the Fogbed architecture.

as a single entity. A distributed fog application and its services run in one or more virtual nodes inside a virtual instance. It is assumed that the management system has full control over which applications are executed on each virtual instance. However, it is not concerned with the internal execution and communication details of the virtual instance. In the virtual switches, settings for flow monitoring and QoS controls are configured from the host machine. The controller process generates routing and QoS settings to enable virtual connections between virtual instances.

The network traffic monitoring can be implemented by running flow monitors for each network interface on virtual nodes and virtual switches. The run-time data from these processes are saved in the folders of the virtual node and host machine. Each virtual node contains a VM-to-Host folder which is connected to the same folder on the host machine. This facilitates the data exchange activities between the virtual nodes and the host machine.

C. Topology API

Using the concept of virtual instances, different components can be emulated, including its links and properties. For example, a virtual node with constrained resources attached to a virtual switch can represent a virtual gateway for IoT application testing. In an upper layer, a set of virtual switches that connect multiple virtual nodes with sufficient resources to run applications can represent a virtual fog node. Depending on the hardware configuration of the host machine, small clouds can be emulated. With the use of network address translation, the virtual instance is able to communicate with external devices on the Internet or act as proxies for cloud services located in remote data centers.

The hierarchical organization of the fog architecture favors intelligent network routing based on SDN. The switches in

the emulated network are configured by standard SDN controllers as part of the Mininet emulation environment. Mininet supports several SDN controllers and can work with external OpenFlow controllers. With a high-level programmatic interface, sophisticated network protocols and forwarding setups can be implemented by developers. Also, these and other custom controller implementations provided by third-party solutions can be integrated with the management system.

An address scheme based on an invalid subnet allows for the support of millions of devices with IP addresses, which is sufficient to evaluate large-scale solutions. An arbitrary number of virtual switches can be added between virtual instances. Listing 1 shows an example script for defining a network topology where three instances are initiated. The topology API is based on the Mininet Python API. With this interface, developers can use executable scripts to generate different topologies in a simple and sample-based way.

```
1  e1= net.addEdgeInstance("Edge gateway")
2  f1= net.addFogInstance("Fog device")
3  c1= net.addCloudInstance("Cloud gateway")
4  s1 = net.addSwitch("switch from Edge to Fog")
5  s2 = net.addSwitch("switch from Fog to Cloud")
6  net.addLink(e1, s1, delay="10ms", loss=2)
7  net.addLink(f1, s1, delay="50ms")
8  net.addLink(f1, s2, delay="50ms")
9  net.addLink(c1, s2, delay="100ms")
10 r1 = ResModelA(max_cu=20, max_mu=40, max_su=60)
11 r1.assignInstance(e1)
12 r2 = ResModelB(max_cu=200, max_mu=150, max_su=200)
13 r2.assignInstance(f1)
14 r3 = ResModelC(max_cu=1000, max_mu=1500, max_su=2000)
15 r3.assignInstance(c1)
16 api1 = EdgeApi(port=8001)
17 api1.connectInstance(e1)
18 api1.start()
19 api2 = FogApi(port=8002)
20 api2.connectInstance(f1)
21 api2.start()
22 api3 = CloudApi(port=8003)
23 api3.connectInstance(c1)
24 api3.start()
25 net.start()
```

Listing 1: An example of Fogbed topology script.

D. Instance API

The management system needs to interact with the emulated environment to control virtual nodes within virtual instances. The instance API provides an infrastructure as a service (IaaS) semantics to manage virtual nodes in an adaptable way. It is designed as an abstract interface to allow for the integration and testing of third-party systems. The developers can implement their own management interfaces on top of a virtual instance API. The default approach adds one specific instance API to each type of virtual instance. These instance APIs are assigned to virtual instances in the topology scripts (Listing 1, lines 16–23). With this flexible conception, the execution of different management strategies for each virtual instance in the emulated environment is possible.

E. Resource API

Fog environments are characterized by the existence of fog nodes with different capacities and resources. While cloud services provide virtually infinite compute resources to their clients, fog nodes and IoT gateways offer limited computing, memory, and storage resources to applications. These different scenarios must be considered by a management system when offloading and resource allocation decisions are taken.

To emulate such resource limitations, the emulation architecture offers the concept of resource models assigned to each virtual instance (Listing 1, lines 10–15). These models are invoked when containers are allocated or released. They are used to compute CPU time (*max_cu*), available memory (*max_mu*) and storage (*max_su*) limits for each virtual instance. Therefore, it is possible for a virtual instance to reject allocation requests from the management system if there are no free resources available. The resource API allows developers to create specific testing scenarios involving instances with different capacities in the emulated environment.

V. Fogbed Using Example

A healthcare prevention and monitoring service can take advantage of fog computing to reduce the response time for data queries and critical events [30]. This application involves searching for and selecting a desired sensor or a group of sensors, where data are generated dynamically in a vast number of sources, geographically distributed across a region or an entire country.

A. The Case Study

A public health-care doctor wants to predict in real time which k users are more likely to suffer a heart attack. The goal is monitor the health of critical users, provide proactive assistance and set up emergency alerts. In order to be useful to users and health workers, the immediate outputs of the system need to be accurate and prioritize the worst cases.

Based on m health parameters established by the medical community, the doctor can implement a priority function $health_e(d_{i,t})$ that waits for the health data input $d_{i,t} = (d_{i,t}[1], d_{i,t}[2], ..., d_{i,t}[m])$ of the user i in the time t and generates as output a priority score $ps_{i,t}$. This output value represents the user health condition in relation to the criteria adopted by the medical community in the time t. Therefore, it can be used to prioritize the patient care. Different $health_e$ functions can be used depending on the established criteria, diseases and individuals that one wishes to monitor.

Using a cloud system application, this function can be sent in the query $q_{i,t}(health_e, k)$ to be installed on different fog nodes or smart devices with compute capability to execute the query. In this way, the values of the sensors or devices are monitored locally in real-time according to the defined criterion. The k parameter associated with the function defines which users will be prioritized by the service.

B. Fog Service Solution

In this example, the performance of a fog service is analysed with Fogbed. For this purpose, a top-*k* monitoring query processing method is compared in different experimental scenarios. The employed method is similar to the distributed strategy defined in [14] to monitor weather information events

in real-time. However, [14] aims to monitor top-k queries in 2-tier distributed systems while the proposed method focuses on top-k queries in a n-tier fog horizontal architecture.

Also, as in [14], the proposed fog service is based on a publish-subscribe scheme and maintains a cache with top-k values collected from edge devices to reduce its latency. The method was developed in Python and the sensors were simulated from datasets with priority score values. It is assumed all edge devices have health functions already installed.

C. Environment Configuration

We consider three factors for the experimental investigation: (*i*) the paradigm, (*ii*) the number of fog instances and (*iii*) the number of edge instances per fog instance. The analysed scenarios are composed of an arrangement of these factors. The service latency (in seconds) to complete a rank-aware top-k query was considered as the response variable.

TABLE I: The emulation environment for each testing scene.

Test ID	k	Paradigm	#VCIs	#VFIs	#VEIs	#Devices
F1E8	10	fog	1	1	8	4 millions
F1E16	10	fog	1	1	16	8 millions
F2E8	10	fog	1	2	8	4 millions
F2E16	10	fog	1	2	16	8 millions
F4E8	10	fog	1	4	8	4 millions
F4E16	10	fog	1	4	16	8 millions
C1E8	10	cloud	1	1	8	4 millions
C1E16	10	cloud	1	1	16	8 millions
C2E8	10	cloud	1	2	8	4 millions
C2E16	10	cloud	1	2	16	8 millions
C4E8	10	cloud	1	4	8	4 millions
C4E16	10	cloud	1	4	16	8 millions

The configuration of each testing scene is shown in Table I. In the cloud paradigm, all sensing data are sent and processed directly in the cloud instance. In these emulated scenes the fog instances act only as a gateway to edge devices. Also, in none of the scenarios an edge instance processes top-k queries. Its function is restricted to the caching of the last priority score values calculated from the user smart health devices below it in the fog hierarchy. Each edge instance simulated 500 thousand health-care users or smart health devices.

After each top-k request to an edge instance, its cached data values are sent to the nearest fog instance or, depending on the paradigm, to the cloud instance for top-k query processing. In the fog paradigm, after the lower-level fog instance processes the edge device values, only the top-k priority scores are sent to the next-level fog instance or, at the last-level, to the cloud.

D. Execution and Preliminary Results

The experiments were conducted in a 1.6 GHz Intel Core i5 processor equipped with 8GB RAM and 256GB SSD. Synthetic uniform datasets were used for all tests. A preliminary comparison of the testing scenarios is presented in Figure 5. The emulation environment testing scene configured in F2E8 and G2E8 is shown in Figure 4.

The response time achieved with fog paradigm scenes is better than the response time obtained with cloud paradigm scenes. Using the fog paradigm, the latency was 52.03% lower

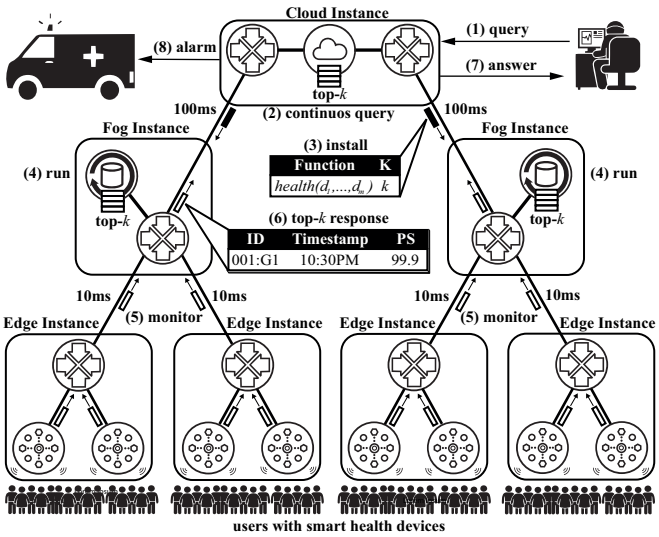

Fig. 4: A real-time healthcare monitoring fog service.

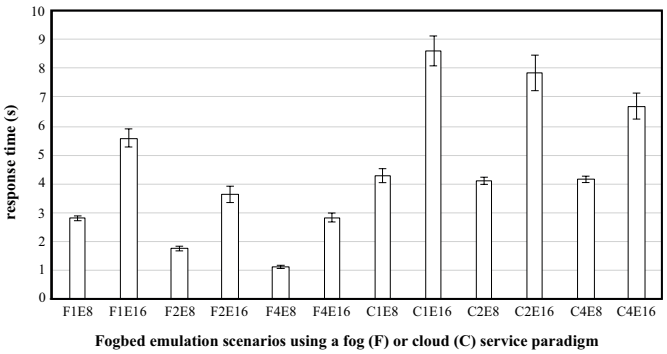

Fig. 5: Evaluation of service latency in different scenarios.

on average than in the cloud paradigm. Considering only fog paradigm scenes, it was observed that, by doubling the number of fog instances in a balanced way, the latency was 33.02% lower on average. Also, by doubling the number of edge instances, and consequently the number of monitored devices, the latency had an increase of 121,87% in average.

VI. Conclusion and Future Work

This paper proposed Fogbed which is a fog emulation system based on Mininet and Docker open source software. Its current components were designed for rapid prototype and test fog services in a realistic environment running on a desktop.

Using container images, it is possible to offer virtual fog nodes with computing resources in the emulated network. The proposed architecture remains faithful to the fog model by enabling the developers to define fog instances that treat virtual nodes and switches as a single entity.

The use of the same pre-created container images to start more than one instance makes the environment setup flexible. New types of container images can be deployed to allow for the testing of different IoT solutions inside a fog architecture. In a simple procedure, it is possible to build fog environments with multiple IoT data sources from a few container images.

Fogbed allows developers to use their tested applications in real world environments with minimal changes. The provision of standard interfaces allows for the testing of third-party systems for management functions such as resource management, virtualization, and service orchestration.

In the current stage, Fogbed meets its main goal which is the emulation of the fog layer for rapid prototyping a test of fog components. It has some restrictions with respect to the emulation of the cloud and edge layers. For example, some desktop computers may not have processors with advanced virtualization functions to support the installation of the current cloud middleware. To work around this limitation, the virtual instance can be configured as a cloud gateway to enable a remote connection to real cloud data centers.

Fogbed uses datasets to simulate the generation of data from each device and sensor aggregated in virtual IoT gateways. However, through wireless interfaces installed on the host machine, virtual edge instances can connect and gather data from real IoT devices at real time. An advance in this regard is the addition of virtualized WiFi stations and access points based on Mininet-WiFi fork and its extension to support IoT-class wireless networking (e.g., Bluetooth, Zigbee, etc.).

The main future goals are to enable scalable fog emulation and the testing of reliable distributed algorithms in fog scenarios. Aspects such as security, fault tolerance and reliable management can be investigated by developing functionalities to allow the insertion of simulated attacks and problems in the emulated environment. Finally, further research concerning a scalable architecture and its components can allow for the use of Fogbed for realistic and reproducible fog experiments.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing-a key technology towards 5g," *ETSI White Paper*, vol. 11, 2015.

[4] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[5] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*. IEEE, 2015, pp. 73–78.

[6] E. M. Tordera, X. Masip-Bruin, J. Garcia-Alminana, A. Jukan, G.-J. Ren, J. Zhu, and J. Farre, "What is a fog node a tutorial on current concepts towards a common definition," *arXiv preprint arXiv:1611.09193*, 2016.

[7] F. Bonomi, R. Milito, P. Natarajan, and 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.

[8] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," *arXiv preprint arXiv:1601.02752*, 2016.

[9] OpenFog, "Reference architecture for fog computing," Available: http://www.openfogconsortium.org/ra/, 2017, accessed: 20 set. 2017.

[10] C. C. Byers, "Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, 2017.

[11] Fogbed, "Project page," Available: https://github.com/fogbed/fogbed, 2017, accessed: 15 out. 2017.

[12] R. L. S. de Oliveira, A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. IEEE, 2014, pp. 1–6.

[13] Docker, "project home page," Available: https://www.docker.com, 2017, accessed: 20 set. 2017.

[14] K. Udomlamlert, T. Hara, and S. Nishio, "Subscription-based data aggregation techniques for top-k monitoring queries," *World Wide Web*, vol. 20, no. 2, pp. 237–265, 2017.

[15] Cisco DevNet, "Iox and fog director developer page," Available: https://developer.cisco.com/site/iox/, 2017, accessed: 20 set. 2017.

[16] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "Fit iot-lab: A large scale open experimental iot testbed," in *Internet of Things (WF-IoT), 2015 IEEE World Forum on*. IEEE, 2015, pp. 459–464.

[17] B. Confais, A. Lebre, and B. Parrein, "Performance analysis of object store systems in a fog/edge computing infrastructures," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 294–301.

[18] C. Prazeres and M. Serrano, "Soft-iot: Self-organizing fog of things," in *Advanced Information Networking and Applications Workshops (WAINA), 2016 30th International Conference on*. IEEE, 2016, pp. 803–808.

[19] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.

[20] G. Lee, W. Saad, and M. Bennis, "An online secretary framework for fog network formation with minimal latency," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[21] X. Chen and J. Zhang, "When d2d meets cloud: Hybrid mobile task offloadings in fog computing," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[22] H.-J. Hong, J.-C. Chuang, and C.-H. Hsu, "Animation rendering on multimedia fog computing platforms," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 336–343.

[23] M. Etemad, M. Aazam, and M. St-Hilaire, "Using devs for modeling and simulating a fog computing environment," in *Computing, Networking and Communications (ICNC), 2017 International Conference on*. IEEE, 2017, pp. 849–854.

[24] C. Sonmez, A. Ozgovde, and C. Ersoy, "Performance evaluation of single-tier and two-tier cloudlet assisted applications," in *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*. IEEE, 2017, pp. 302–307.

[25] O. Bibani, C. Mouradian, S. Yangui, R. H. Glitho, W. Gaaloul, N. B. Hadj-Alouane, M. Morrow, and P. Polakos, "A demo of iot healthcare application provisioning in hybrid cloud/fog environment," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 472–475.

[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[27] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments," *arXiv preprint arXiv:1606.02007*, 2016.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[29] M. Peuster, H. Karl, and S. Van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," *arXiv preprint arXiv:1606.05995*, 2016.

[30] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 356–363.