A Distributed Environment for Scalable Fog Computing Emulation

Heitor Rodrigues Santos Silva, Antonio Coutinho
Department of Technology
State University of Feira de Santana
Email: hr.heitor@hotmail.com, acoutinho@uefs.com.br

***Abstract*—The fog computing model reformulates the way cloud services act in the network, making it work on a widely distributed level to overcome challenges of cloud-based Internet of Things (IoT) platforms. The deployment of IoT solutions employing a fog architecture requires a decentralized and scalable computing infrastructure, which places networking, compute and storage resources in a hierarchy of levels arranged between the data source and the cloud. Despite recent advances in fog platforms, there exists no readily available testbed which can help researchers to design and evaluate fog applications on a truly IoT scale. The current fog prototyping tools are adapted from cloud middleware or network simulators to enable the evaluation of fog solutions in limited environments. This paper presents a framework and toolset integration that uses the Fogbed emulator to enable fog distributed testbeds in virtualized environments. Unlike current approaches, the proposed framework allows for the deployment and testing of fog components in a scalable way. Its design is compatible with real world technologies and meets the requirements of low cost, flexible setup and supports third-party systems through standard interfaces. A case study example is presented to demonstrate the emulation of fog distributed components and services using a cluster approach. In addition, future developments and research directions are discussed.**

***Keywords*—Fog, cloud, edge, SDN, emulator, network, container.**

I. Introduction

In the last decade cloud computing came to be known as the standard way to build applications and services that needed resources on demand, making it easier than ever to store and process data in a centralized fashion, and it fits well with the early stage requirements of the Internet of Things (IoT), where many edge devices with low computing power can send their data to be stored and processed at a data center.

Nowadays, several initiatives support a paradigm shift towards a decentralized architecture to overcome limitations of cloud based IoT platforms. Current proposals [1], [2], [3], [4] can be summarized through their similar strategies: use of edge-located small clouds with virtualization support, distribution of computational resources on large scale, provision of IoT capabilities based on cloud service delivery models, and the offer of exclusive services using information that is present on the local network or in its proximity.

In [1] the concept of fog computing is defined as a virtualized platform which offers services between end devices and data centers of cloud computing. It spreads the concept of edge computing [4] in a scalable and integrated way with network devices such as switches, routers and IoT gateways.

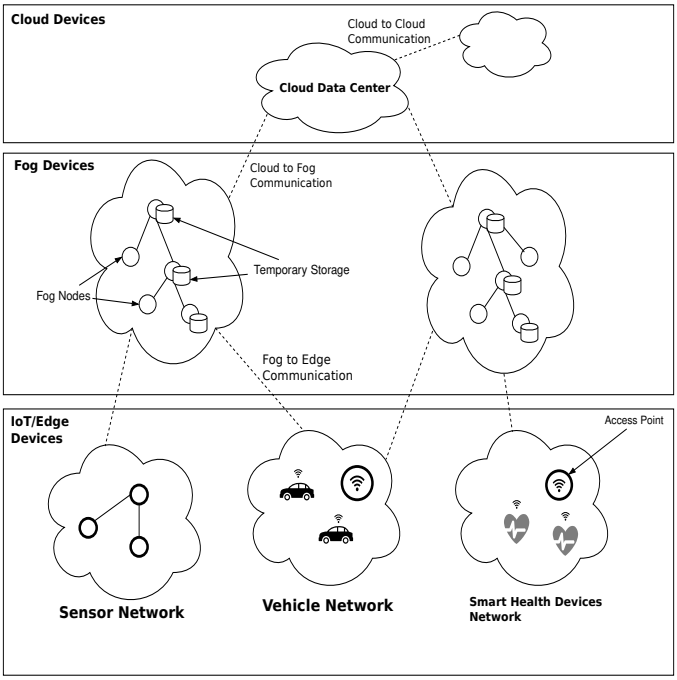

Fig. 1: Fog computing architecture.

The distributed approach taken by fog computing reduces the amount of data going through the core network by capturing and processing part of the data close to the edge. Its low response time brings real-time analysis to data coming from edge devices by distributing the process across the network [5].

The fog system architecture is distributed hierarchically in a n-tier network infrastructure with at least three layers. Figure 1 presents a fog system vision with three levels: the cloud level, the fog level, and the IoT/edge level.

At the edge of the network sensor and actuator devices installed in parks, industries, buildings, vehicles and streets can provide valuable streams of data for monitoring, control, analysis and prediction applications running on nearby edge gateway nodes. Its location depends on factors such as energy saving, response time or application latency requirements. In some applications, embedded smart IoT devices can receive actuation commands or send sensing data without the need for dedicated gateways. Also, they are capable of acting as edge gateways to other IoT or end-user devices.

The fog level is formed by one or more fog domains,

controlled by the same or different providers. In the system hierarchy the fog layer makes the transmitted data actionable, meaningful and valuable by filtering data at different levels of the infrastructure. It involves the running of services and applications concerning distributed components on the fog nodes between the IoT devices and the cloud data centers.

The services that manage and support virtually unlimited resources for IoT applications are executed at the cloud level. These applications can use both cloud and fog resources to provide intelligent and cutting edge solutions to end users. This can be achieved by efficient solutions that distribute task execution demands across optimal, well located and available fog nodes. Its main goal is to provide tolerable levels of latency while optimizing the performance of fog applications.

Real world fog environments are expected to support millions of IoT and end-user devices. They may involve a large number of applications, fog domains and fog nodes. Also, several applications may have a large number of components. Thus, fog systems need to be operational on large scales and they should be able to scale down and up in an elastic fashion.

A current problem in this area is the lack of supporting environments to prototype and test fog services, components and applications over a large scale [6] [7]. Currently there are no readily available fog testbeds that can help researchers to design and verify distributed algorithms on a truly IoT scale. At this moment, network simulators and cloud middleware are adapted to allow the experimental evaluation of fog solutions in limited environments, specific scenarios and restricted conditions. However, testing and validating an architecture with few devices and nodes does not guarantee it performs properly over a large scale environment, with regard to the quality and performance of the delivered solution.

In a previous work [7], a framework named Fogbed was presented in order to allow for the emulation of fog components. Its functionalities provide developers with an environment to prototype fog solutions close to reality, running locally on a single desktop.

This paper presents a framework and toolset integration that extends Fogbed to enable the deployment and testing of real world fog components in a scalable way. It is made possible by distributing the emulated environment across a cluster of host machines in a local network.

In addition to some implementation details of the framework, examples of network topologies and architectures scripts are presented to demonstrate how the current version of the scalable Fogbed works and how it can be used to collect valuable data from experiments.

The remainder of the paper is organized as follows. Section II introduces some basic concepts needed to understand this project, Section III presents related work. Section IV addresses requirements and technological choices, Section V describes the architecture of the emulator, Section VI show an usage example and some results obtained running it and Section VII concludes with final considerations and possibilities for the future.

II. Concepts

In this section some of the basic concepts around the project proposed in this paper are described.

A. Cloud Computing

As described by the American National Institute of Standards and Technology (NIST) [8], cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The cloud has three service models, Software as a Service (SaaS), which provides software running on a cloud infrastructure for an end user, Platform as a Service (PaaS), which provides to the user the capability of deploying his own applications on a cloud infrastructure, and Infrastructure as a Service (IaaS), which provides computing resources to the user, like processing, storage and networks so that he can run arbitrary software on it.

Cloud services are usually implemented as large centralized data centers and some of the biggest commercial cloud providers are Google (Google Cloud Platform), Amazon (AWS), Microsoft (Azure) and IBM (IBM Cloud).

B. Fog Computing

Fog Computing is a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing data centers, typically, but not exclusively, located at the edge of network, as defined in [1].

The defining characteristics of the Fog are: low-latency and location awareness, wide-spread geographical distribution, mobility, very large number of nodes, predominant role of wireless access, strong presence of streaming and real time applications and heterogeneity.

Fog still is a new concept in computing, the OpenFog Consortium released the first paper describing an architecture for it in 2016, so many practical details about how it's going to work are unknown and many of the papers about the subject try to demonstrate how some of the entities are going to interact with each other.

C. Network Emulator

An emulator is a software or hardware system capable of behaving like some other system. Emulators are capable of replacing the original system with little performance gap, different of simulators that try to model every step done by a system, without the intention of replacing them, for close analysis of how the system works, usually with much worse performance than the original.

Network emulators try to mimic the components that are part of computer networks (switches, routers, hosts, etc) with the possibility of measuring and constraining traffic, bandwidth, latency and loss rate for a user defined topology.

D. Container Platform

As described in [9], a container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A container image is a lightweight, stand-alone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

The container images become containers at runtime when they are ran by a container platform, like Docker [10]. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware.

Being lightweight and providing isolation from the host machine, containers make great network hosts for a network emulation where each of them have a certain role in the topology defined by the user.

E. Generic Routing Encapsulation (GRE)

In order for machines to communicate over the internet infrastructure, the packets sent from these machines need to contain the IP protocol header, thus becoming an IP packet.

GRE is a tunnel made to encapsulate packets from other OSI layer 3 (Network layer) protocols in IP packets, they can be transferred over an IP Network, creating a point-to-point connection like that of a virtual private network (VPN).

F. Software Defined Networking (SDN)

Software defined networking (SDN) advocates separating the data plane and the control plane, making network switches in the data plane simple packet forwarding devices and leaving a logically centralized software program to control the behavior of the entire network. SDN introduces new possibilities for network management and configuration methods, as described in [11].

In SDN the network devices (e.g. switches) communicate with some central unit called Controller that knows how to make decisions for various kinds of traffic and is capable of pushing rules to them, increasing the flexibility of network management.

III. Related Work

Fundamental works [5], [12], [13], [14] have proposed a reference architecture, design goals and components of a fog platform. In [15] is discussed important architectural requirements and techniques to deploy fog-enabled IoT networks in the context of exemplary use cases. In [6] a structured classification of the present architectures, algorithms, issues and research directions for fog systems are presented.

Existing work has focused on the design of an adequate fog computing architecture, which can serve as a development model for different large scale platforms. In November 2015, the OpenFog consortium [16] united leading IT-companies and universities in order to develop an open standard architecture for fog computing. The first version of OpenFog reference architecture [14] was released in February 2017.

However, to validate a fog-based architecture, it is necessary to deploy its components in a suitable environment to assess aspects such as safety, performance and scalability.

Cisco was one of the earliest companies to provide an open testbed environment for fog applications through packaged labs called IOx Sandboxes [17]. The main components of the Cisco IOx application environment are: (*i*) the Cisco IOx [18], a network operational system that makes processing and storage available to a wide variety of IoT applications hosted in virtual machines (VMs); (*ii*) the Fog Director [19], which provides centralized services and RESTful APIs to manage, integrate, monitor, and troubleshoot Cisco IOx applications and devices remotely over the network using a desktop approach; (*iii*) SDK and development tools, a set of software packages used by third-party developers to build and integrate applications. The Cisco IOx Application Framework (CAF) also supports open source systems and applications based on Linux instances running in a hypervisor.

The Cisco IOx Sandbox enables the development and testing of fog applications using a computer with internet access to run simulations. A developer can reserve and remotely access the simulated environment to install, deploy and execute IOx applications.

The IOx platform running inside a sandbox is a proper tool for promoting the teaching and learning of Cisco IoT proprietary technologies. However, the testing of solutions with a few devices and nodes does not guarantee it performs properly over a large scale environment, in terms of quality and performance of the delivered solution. In this regard, a widely distributed computing infrastructure that places networking, compute and storage resources in a hierarchy of levels arranged between the data source and the cloud is required.

In the absence of large scale fog testbeds, an alternative is to adapt current IoT testbeds to support experimental evaluation of fog solutions. The largest IoT testbeds such as [20] and [21] are provided with thousands of IoT devices geographically distributed. The authorized users can run applications over a large number of resources to evaluate low level protocols, analysis techniques and services on a very large scale. However, since they are not aimed at fog applications, a major effort is required to configure and deploy fog experiments.

Despite the relevance of the scalability criterion, the fog solutions found in the literature and involving real technologies were evaluated in small scale scenarios. Other research works which employ fog computing in large scenarios were evaluated through numerical simulations such as [22], [23], [24].

Although extremely useful, a large-scale simulation does not address the problems relating to its practical implementation. Besides the scalability, proposed algorithms need to operate in real world conditions with realistic system parameters. There are initiatives and open source technologies that can be used to create real world fog testbeds. For example, Soft-IoT [25] is an IoT framework that supports fog and edge computing.

In [26] the development and orchestration of edge distributed microservices uses a prototyping fog platform consisting of four clouds implemented with OpenStack [27].

However, real world fog systems have been validated at a small scale with no guarantees that they perform well at a large scale, thus leaving it as a challenge to address. This remains a default approach, given the challenges involved in running and operating real world fog experimental evaluations over a large scale. Although desirable, the use of a real world testbed facility is expensive, time-consuming and in some cases difficult to access, configure and operate.

Therefore, low-cost effective alternatives should be considered. Large scale realistic simulations remain a tractable option enabling the comparison of different algorithms, as well as eliminating ineffective policies and strategies. To this end, the adaptation of existing open source frameworks or simulators is a common approach in the fog research area.

In [28] and [29] fog-based mechanisms for intelligent transport systems (ITS) that aim to manage traffic congestion in vehicular ad hoc networks (VANETs) are presented. Both simulation environments were performed using the OMNET++ which is not a network simulator itself but an extendable library and framework for building network simulators.

Specific simulators to evaluate fog computing frameworks were designed in [30], [31], [32]. In [13] a case study on smart traffic management extended the CloudSim [33] with fog capabilities to improve the performance of the applications in terms of bandwidth consumption and response time. This work was the basis for iFogSim [34], a simulator that supports edge and fog resources in different scenarios. It allows for the evaluation of resource management policies by analysing their influence on latency, network congestion, energy consumption and operational costs. In [35] MyiFogSim extended iFogSim to support mobility through the migration of VMs between cloudlets. Nonetheless, the above mentioned environments do not address real world and large scale simulations.

In [36] a fog emulator called EmuFog was proposed, it is built on top of MaxiNet [37] and makes use of Docker [10] to provide network hosts as containers. EmuFog has as it's main feature a fog node placement algorithm, where it takes into account arbitrary latency costs given by the developer to the connections between hosts and switches, and distributes fog nodes across the network in order to get the best edge device coverage. After deciding on the placement of fog nodes, the network created is passed on to MaxiNet, that is responsible for all the heavy lifting of initializing every device and emulating the network. As it imposes the way fog nodes should be distributed, it takes away some of the freedom from the user, and it doesn't provide an interface to query MaxiNet about network specific information.

In summary, fog solutions continue to be developed as proof-of-concept, implemented in limited environments, specific scenarios and restricted conditions. Therefore, current research concerning real world and scalable fog computing testbeds may help to validate fog components, algorithms and applications. These pretested and validated architectures will help service providers achieve their goals, while minimizing the risks involved in moving to the fog computing model.

IV. Solution Requirements and Technologies

The present architecture is based on solutions that meet the following requirements: (1) low cost; (2) fast and flexible setup; (3) support to perform real world protocols and services; and (4) scalable deployment. In the remainder of this section, open source solutions that enable the deployment of the main fog components in accordance with these requirements are described. They allow for the testing of real world technologies in a repeatable and controllable environment.

A. Docker Container Platform

Docker [10] is a software platform that uses a Linux kernel technology called containerization. A container is a 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 are similar to traditional VM instances. However, containers do not bundle a full operating system. The Docker engine shares the host kernel and isolates required resources using a lightweight approach based on Linux cgroups and kernel namespaces.

Developers are increasingly using containers because they are flexible, portable and scalable. Even complex applications can be containerized and transmitted over the internet in a small file size compared with VM images. Thus, it is possible to build a container image locally, deploy it to the cloud or automatically distribute and run replicas in the network.

B. MaxiNet Framework

MaxiNet [37] is a framework that extends the Mininet [38] network emulator to deploy the virtual network environment across a cluster of physical machines. Mininet is the most common tool to emulate software-defined networks (SDN) of several hundred virtual nodes. According to their authors, MaxiNet makes it possible to emulate a large SDN of several thousand virtual nodes on a handful of physical machines.

The MaxiNet environment runs on a pool of host machines called workers. Each worker machine runs a Mininet instance and emulates a part of the whole virtual network. These Mininet instances can be configured and interconnected using generic routing encapsulation (GRE) tunnels across different workers. The MaxiNet API is invoked at a specialized worker called the frontend. The frontend allocates the virtual nodes onto the workers and maintains a list of which virtual node resides on which worker. Therefore, it is possible to access and manage the virtual environment in a centralized approach using the built-in MaxiNet command line interface (CLI).

MaxiNet workers can be started using preconfigured VM images or by installing Mininet from scratch on non-virtualized machines when performance is essential. Also, it can be configured to run Docker containers instead of the default Mininet virtual nodes. Alternatively, replacing Mininet

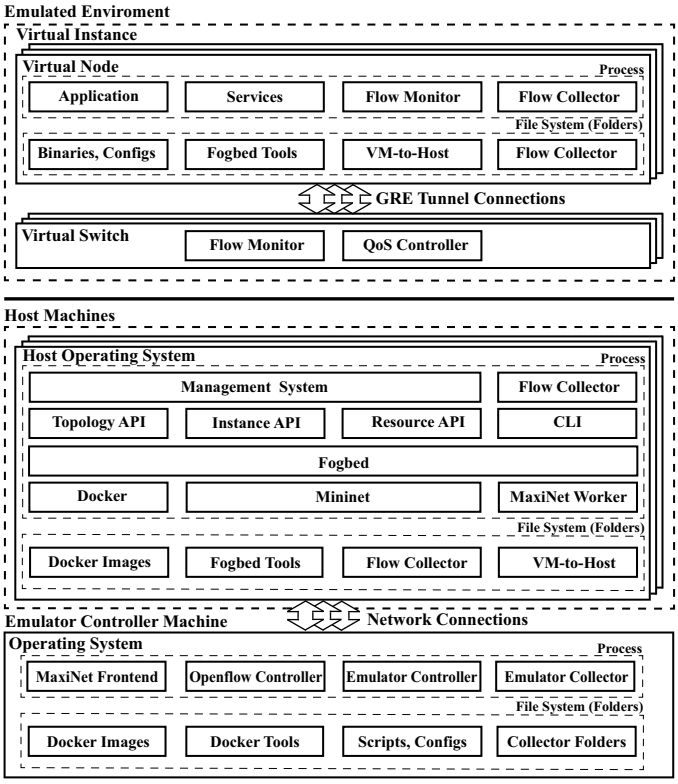

Fig. 2: An overview of the Fogbed distributed architecture. [39]

instances with other Mininet-based emulator is technically feasible.

C. Fogbed Framework

Fogbed [7] is a framework that extends the Mininet emulator to create fog testbeds in virtualized environments. Using a desktop approach, Fogbed enables the deployment of virtual fog nodes as Docker containers under different network configurations. The Fogbed API provides functionality to add, connect and remove 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 environment can be created by deploying virtual nodes, virtual switches, virtual connections and virtual instances in a virtual network environment running on a host machine. The flexible setup is achieved by using preconfigured Docker container images. Each container image comprises part of a distributed application, required services and protocols. Different types of container images can be used to instantiate virtual nodes.

The virtual instance is an abstraction that allows for the management of a related set of virtual nodes and virtual switches as a single entity. A fog application and its services run in one or more virtual nodes inside a virtual instance. Virtual instances differ by the type of resource model applied to them, it could be an over provisioning model, where the

```
1
2  topo = FogTopo()
3
4  f1 = net.addVirtualInstance("Fog")
5  c1 = net.addVirtualInstance("Cloud")
6  e1 = net.addVirtualInstance("Edge")
7
8  r1 = FogResourceModel(max_cu=20, max_mu=40)
9  r2 = CloudResourceModel(max_cu=200, max_mu=150)
10 r3 = EdgeResourceModel(max_cu=1000, max_mu=1500)
11
12 e1.assignResourceModel(r3)
13 c1.assignResourceModel(r2)
14 f1.assignResourceModel(r1)
15
16 d1 = c1.addDocker("d1", ip='10.0.0.251',
17 dimage="ubuntu:trusty")
18 d2 = f1.addDocker("d2", ip='10.0.0.252',
19 dimage="ubuntu:trusty")
20 d3 = e1.addDocker("d3", ip='10.0.0.253',
21 dimage="ubuntu:trusty")
22
23 s1 = net.addSwitch("s1")
24 s2 = net.addSwitch("s2")
25
26 net.addLink(e1, s1, delay="10ms", loss=2)
27 net.addLink(f1, s1, delay="50ms")
28 net.addLink(f1, s2, delay="50ms")
29 net.addLink(c1, s2, delay="100ms")
30
31 exp = FogbedDistributedExperiment(topo, switch=OVSSwitch)
32 exp.start()
33 try:
34   print exp.get_node("Cloud.d1").cmd("ifconfig")
35   print exp.get_node(d2).cmd("ifconfig")
36
37   print exp.get_node(d1).cmd("ping -c 5 10.0.0.252")
38   print exp.get_node("Fog.d2").cmd("ping -c 5 10.0.0.251")
39 finally:
40   exp.stop()
```

Listing 1: An example of a Fogbed topology script.

resource allocation mimics a cloud, or a limiting model, where the resource allocation is restricted to mimic low cost edge devices. The management system process is responsible for deploying and starting the instances in the Fogbed emulation environment. It first executes a script that defines the network topology between virtual instances. The communication between a virtual instance and the management system is performed through a well-defined instance API.

V. Fogbed Distributed Architecture

The aim of this work is to design an architecture that enables the deployment of the main components of a fog environment in a scalable way. Figure 2 shows the Fogbed architecture and system components. More details, source code and usage instructions can be obtained in [7] and [40]. Therefore, we focus on the main components to enable a scalable Fogbed emulation.

The scalability was achieved by partitioning an emulated fog environment into virtual Fogbed instances running on distributed host machines. The MaxiNet API is used to create a scalable cluster of Fogbed instances for deployment and test of real world fog components. To this end, the Mininet default instances were replaced with preconfigured Fogbed instances.

Before any experiment can be executed, each host machine has to start a MaxiNet worker service that waits for everything needed to run a local emulation such as topology scripts and configuration settings. After this stage, the host machine is

ready to be remotely allocated, configured and linked with other Fogbed instances by setting up GRE tunnel connections.

The emulator controller machine assumes the role of frontend. The emulator controller process running in the emulator controller machine uses the MaxiNet API to remotely start and control the execution of different Fogbed instances distributed in the local network. Inside each Fogbed instance, virtual nodes and their network infrastructure are emulated.

Listing 1 shows an example script for defining Fogbed and virtual instances. It uses the Fogbed topology API to define virtual instances and their connections with different link properties. The resource API enables each virtual instance to use a specific resource model (Listing 1, lines 8–14). The example adds one specific host instance to each type of virtual instance (Listing 1, lines 16–21) and then adds the links between the virtual instances (Listing 1, lines 23–29). To execute a specific experiment, it first needs to be instantiated and started, followed by the instructions it should run and, finally, stopping to clean up the allocated resources (Listing 1, lines 31–40).

Figure 3 and Figure 4 depict the high-level workflow of a developer using the environment. Figure 3 shows the distributed emulation environment whilst Figure 4 focuses on the details of starting a single Fogbed instance. First, the developer provides the container images that will be used to setup the emulation (1). Each container includes everything needed to run an application. Using the topology API, the developer defines a distributed environment on which he wants to test the application (2) and starts the emulator controller with the topology definition (3). The emulator controller process uses the MaxiNet API to create the required Fogbed instances as defined in the emulation topology script (4). After the distributed environment has been started, the emulator controller invokes the management system on each Fogbed instance to upload the necessary container images and settings (5). The local management system connects its emulated environment by using the provided instance API. The application is deployed on each Fogbed instance by the local management system which starts the required process and services in each local virtual node (6). Then, the application can be managed from the emulation controller machine using a remote CLI to access each Fogbed instance (7). The network flow statistics can be collected on each Fogbed instance and stored in the emulation controller machine for future analysis (8). Furthermore, the developer in the emulation controller machine can access arbitrary monitoring data generated by the platform (9). The VEI, VFI and VCI shown in Figure 4 refer to Virtual Edge Instance, Virtual Fog Instance and Virtual Cloud instance, respectively, those represent Virtual Instances using edge, fog and cloud resource models.

Inside a Fogbed instance, 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 VM-to-Host folder which is linked to the same folder on its host machine.

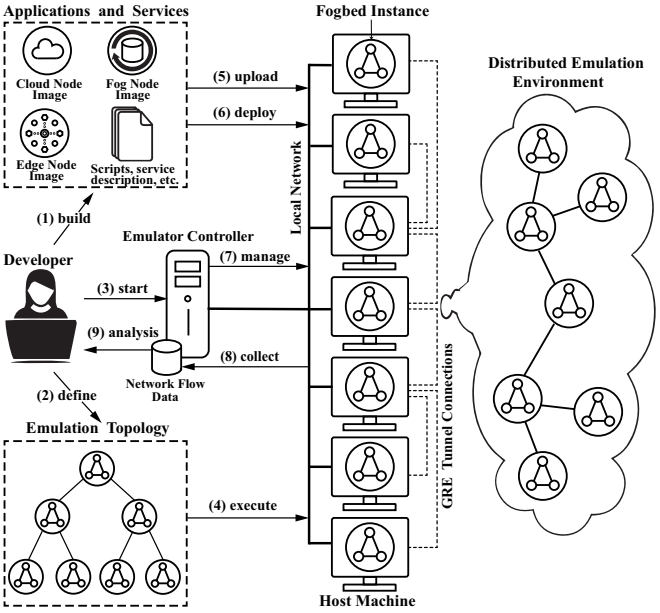

Fig. 3: The Fogbed distributed emulation workflow. [39]

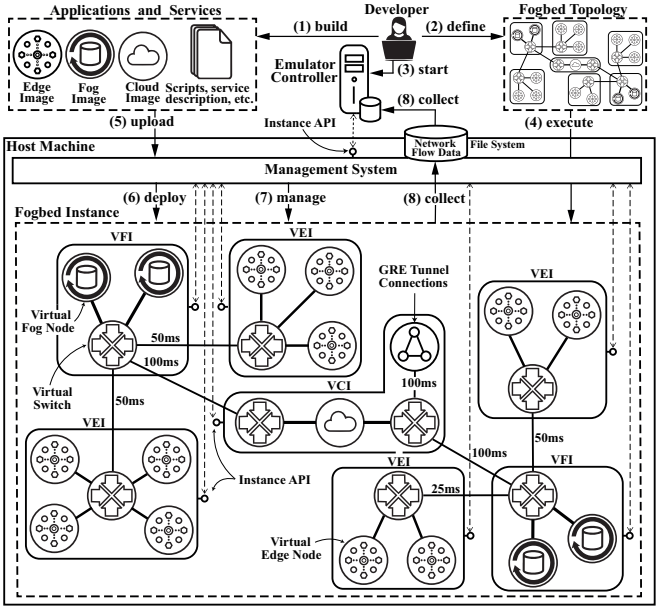

Fig. 4: The Fogbed instance emulation workflow. [39]

VI. Fogbed Using Example

The real-world user experiences with cloud-based healthcare are limited due to privacy issues, excessive networking latency and longer response time [41]. Fog computing is useful to develop healthcare solutions with intelligent and predictive capabilities. A fog-based healthcare system enables low latency, mobility support, location and privacy awareness [41] [42].

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 to monitor the health of critical users, provide proactive assistance and set up emergency alerts. In order to be useful to

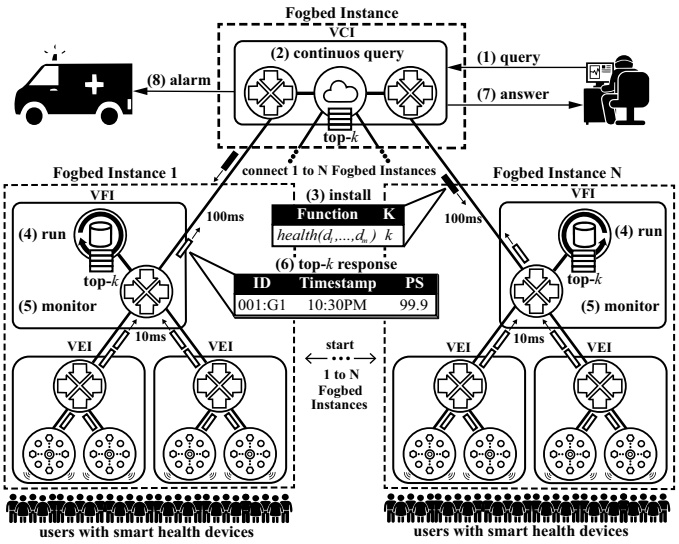

Fig. 5: A real-time healthcare monitoring fog service. [7]

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. 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, two different approaches that employ distributed top-k monitoring query processing methods are compared. Method A consists in requesting data from the cloud directly to each edge device, without the fog nodes preprocessing. Method B is based on a fog model where the cloud requests data to the fog nodes, and the fog nodes forwards the requests to the edge devices close to it and filter the data received before sending back to the cloud.

The methods A and B were implemented in Python and the edge devices were simulated from datasets with priority score values used by the top-k function. It is assumed all edge devices have health functions installed.

C. Execution and Data Collection

The main performance aspect considered for the experiment was the response time. The emulation scenario used in the example is shown in Figure 5. The experiments were conducted in a prototype cluster with eight machines as Fogbed instances one to eight (N=8) and two as emulator controller and SDN controller instance, each of the computers had Intel Pentium D 3.00GHz with 2GB RAM running Ubuntu Server 16.04.

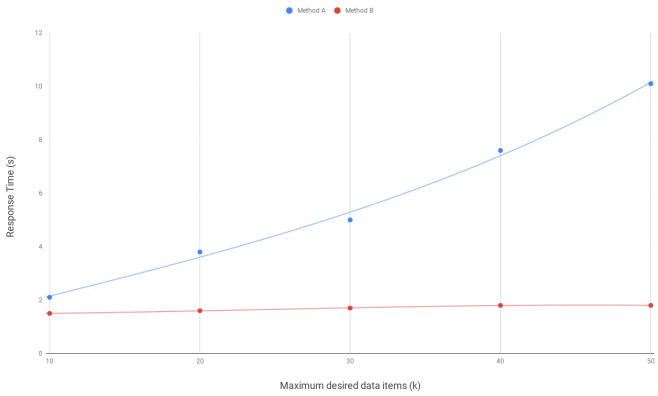

Fig. 6: Evaluation of response time.

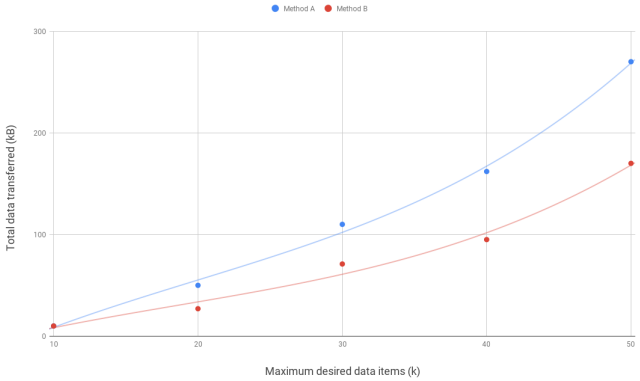

Fig. 7: Evaluation of total data transferred.

A scheme to observe the behavior of the environment is implemented with NetFlow [43] as the flow monitoring technology and NFDUMP [44] as the NetFlow collector and analysis tool. The flow records are sent to a specified *nfcapd* flow capture daemon using the command *fprobe* in virtual nodes and the command *ovs-vsctl* in virtual switches. The *nfdump* flow analysis tool is used to study the collected traffic.

D. Evaluation

In the experiments, each virtual edge instance (VEI) simulated 500 thousand smart health devices. The topology used consisted of 1 VCI, 2 VFIs and 2 VEIs for each VFI. For each value of k presented in the Figure 6 and Figure 7 graphs a query was triggered from the cloud instance 30 times before calculating the average response time and total data transferred, and comparisons were made of distributed top-k running on 8 workers. The response time achieved with Method B is better than the response time obtained with Method A. Figure 6 provide comparative results of the two methods. Figure 7 depict the total amount of transferred data for the two algorithms. This amount corresponds to the number of bytes transferred during query processing to the virtual cloud instance. In the case of data shown in Figure 7, Method

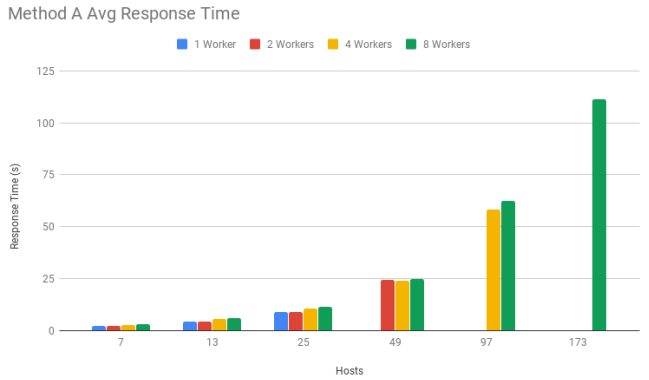

Fig. 8: Method A average response time for 1, 2, 4 and 8 workers.

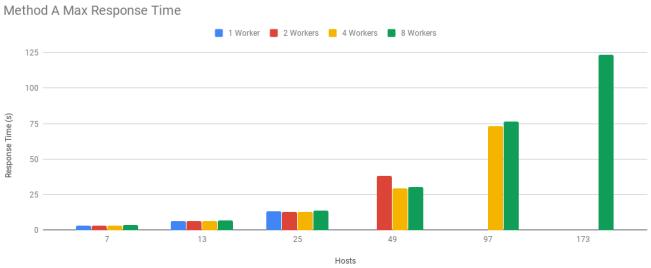

Fig. 9: Method A maximum response time for 1, 2, 4 and 8 workers.

B transfers less data than Method A, mostly due to the filtering done by fog instances before data can reach the cloud.

Figures 8, 9, 10 and 11 show comparisons between running the top-k experiment with 1, 2, 4 and 8 workers, and 7 (1 cloud, 2 fog and 4 edge instances), 13 (1 cloud, 4 fog and 8 edge instances), 25 (1 cloud, 8 fog and 16 edge instances), 49 (1 cloud, 16 fog and 32 edge instances), 97 (1 cloud, 32 fog and 64 edge instances) and 173 (1 cloud, 64 fog, 128 edge instances) virtual hosts for each value of N workers, keeping the value of k at 10. In general the more workers are available, more virtual hosts can be emulated without compromising the results. Using 1 worker the maximum number of hosts emulated before affecting the results was 25, using 2 workers the maximum was 49 and using 4 workers the maximum was 97, trying to emulate more than these maximum values resulted in emulation times longer then 30 minutes for these top-k experiments, mostly due to the time that takes to instantiate each virtual host inside Mininet for each worker. Adding more workers to the cluster virtual hosts can be distributed better without overloading a single worker with the task of instantiating too many virtual hosts.

VII. Conclusion and Future Work

This paper proposed a distributed fog emulation system based on open source technologies used by the research community and developed for use in scalable systems. Its current architecture was designed to prototype and test fog components in a realistic environment running on a cluster of machines. A more general discussion about the Fogbed environment can be found in [7] and the open-source code

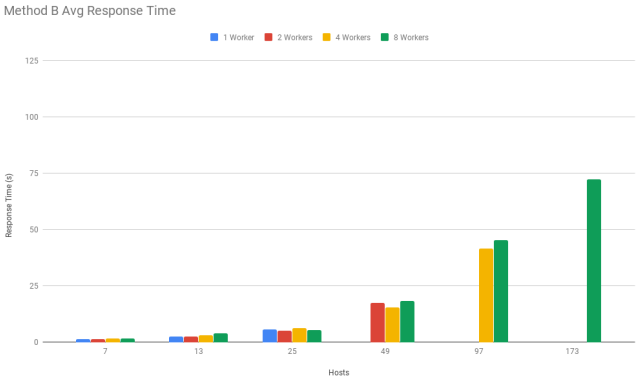

Fig. 10: Method B average response time for 1, 2, 4 and 8 workers.

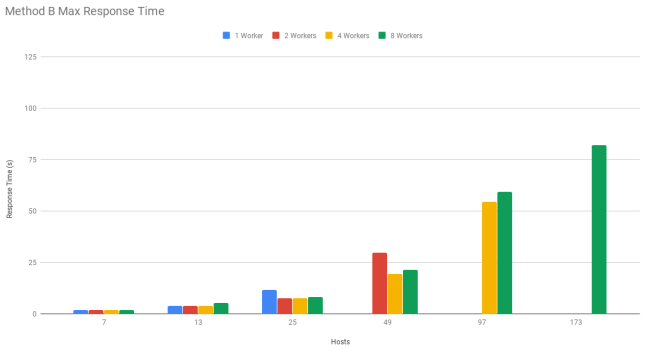

Fig. 11: Method B maximum response time for 1, 2, 4 and 8 workers.

can be found in [40]. The issues raised below are related to the implementation of a scalable Fogbed emulation.

The execution of a Fogbed emulated environment in a cluster of machines makes the proposed architecture scalable. For example, when the memory capacity of a host machine is filled by virtual instances, a new host machine can be added in the network. However, the centralized management controller limits the performance of the emulation. In simulations with many Fogbed instances the time to initiate the environment may increase to an impractical level. The scalability of the architecture can be improved by distributing and balancing the emulation controller functionalities between different hosts.

The use of the same pre-created container images to start more than one instance simplifies the configuration of experiments that deal with massive amounts of data streams such as big data applications. However, when emulating large networks with both high link bandwidths and high traffic volume the results can be distorted by limitations of the physical network or by the limited processing power of the hosts. In such scenarios it is still possible to emulate the network by using a technique called time dilation [37].

In order to improve the range of emulated devices and the allocation of virtual hosts in a cluster, the addition of WiFi devices from the Mininet-Wifi project, [45], and the use of container orchestration technologies, like Kubernetes, were considered. The wireless devices addition could have been done, but it would require major changes on Fogbed due to

all the new kinds of nodes and wireless drivers, and the use of orchestration would require adjustments on the way these technologies handle their container network topology, so it is referenced here for future work.

Unfortunately, the current emulation performance strongly depends on the virtual topology used, the amount of traffic and the software running at the emulated hosts. This makes it hard to give a general statement about performance factors. The case study was performed as proof-of-concept and future work may highlight important performance issues. Also, further research concerning the presented architecture 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] 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.

[6] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, 2017.

[7] A. Coutinho, F. Greve, C. Prazeres, and J. Cardoso, "Fogbed: A rapid-prototyping emulation environment for fog computing," in *Communications Workshops (ICC Workshops), 2018 IEEE International Conference on*. IEEE, 2018, pp. 1–7.

[8] NIST, "The nist definition of cloud computing," Available: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf, 2011, accessed: 30 oct. 2018.

[9] Docker, "What is a container," Available: https://www.docker.com/resources/what-container, 2018, accessed: 30 oct. 2018.

[10] ——, "project home page," Available: https://www.docker.com, 2018, accessed: 21 jun. 2018.

[11] N. F. Hyojoon Kim, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, 2013.

[12] 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.

[13] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[14] OpenFog, "Reference architecture for fog computing," Available: http://www.openfogconsortium.org/ra/, 2018, accessed: 20 jun. 2018.

[15] 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.

[16] OpenFog, "Consortium home page," Available: http://www.openfogconsortium.org/, 2017, accessed: 25 jun. 2018.

[17] Cisco System Inc., "IOx Sandbox," Available: https://developer.cisco.com/docs/sandbox/#iot, 2018, accessed: 10 jul. 2018.

[18] Cisco Systems Inc., "IOx Documentation," Available: https://developer.cisco.com/docs/iox/, 2018, accessed: 10 jul. 2018.

[19] Cisco System Inc., "Fog Director," Available: http://www.cisco.com/c/en/us/products/cloud-systems-management/fog-director/, 2018, accessed: 10 jul. 2018.

[20] 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.

[21] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis *et al.*, "Smartsantander: Iot experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, 2014.

[22] C. Fricker, F. Guillemin, P. Robert, and G. Thompson, "Analysis of an offloading scheme for data centers in the framework of fog computing," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 1, no. 4, p. 16, 2016.

[23] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support iot applications," *Iet Networks*, vol. 5, no. 2, pp. 23–29, 2016.

[24] L. Liu, Z. Chang, X. Guo, and T. Ristaniemi, "Multi-objective optimization for computation offloading in mobile-edge computing," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 2017, pp. 832–837.

[25] 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.

[26] M. Villari, A. Celesti, G. Tricomi, A. Galletta, and M. Fazio, "Deployment orchestration of microservices with geographical constraints for edge computing," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 2017, pp. 633–638.

[27] Openstack, "home page," Available: https://www.openstack.org, 2018, accessed: 25 jun. 2018.

[28] C. A. Brennand, F. D. da Cunha, G. Maia, E. Cerqueira, A. A. Loureiro, and L. A. Villas, "Fox: A traffic management system of computer-based vehicles fog," in *Computers and Communication (ISCC), 2016 IEEE Symposium on*. IEEE, 2016, pp. 982–987.

[29] C. A. Brennand, A. Boukerche, R. Meneguette, and L. A. Villas, "A novel urban traffic management mechanism based on fog," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 2017, pp. 377–382.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Myifogsim: A simulator for virtual machine migration in fog computing," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. ACM, 2017, pp. 47–52.

[36] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *2017 IEEE Fog World Congress*. IEEE, 10 2017, pp. 1–6.

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

[38] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[39] A. Coutinho, F. Greve, C. Prazeres, and H. Rodrigues, "Scalable fogbed for fog computing emulation," in *Symposium on Computers and Communications (ISCC), 2018 IEEE International Conference on*. IEEE, 2018, pp. 334–340.

[40] Fogbed, "Project page," Available: https://github.com/fogbed/fogbed, 2018, accessed: 20 jun. 2018.

[41] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, "Fog data: Enhancing telehealth big data through fog computing," in *Proceedings of the ASE BigData & SocialInformatics 2015*. ACM, 2015, p. 14.

[42] 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.

[43] B. Claise, Ed., "Cisco systems netflow services export version 9," Available: https://tools.ietf.org/html/rfc3954, 2004, accessed: 20 jul. 2018.

[44] Haag, P., "Nfdump-netflow processing tools," Available: http://nfdump.sourceforge.net, 2011, accessed: 20 jul. 2018.

[45] Fontes, R. R. and Rothenberg, C. R. E., "Mininet-wifi," Available: https://github.com/intrig-unicamp/mininet-wifi, 2017, accessed: 14 feb. 2019.