An Architecture for Fog Computing Emulation

Antonio A. T. R. Coutinho[1], **Fabíola Greve**[2], **Cássio Prazeres**[2]

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana, Bahia 44036–900

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia
Salvador, Bahia 40170–110

augusto@ecomp.uefs.br,{fabiola,cprazeres}@ufba.br

Abstract. *Fog computing is an emergent paradigm that integrates cloud computing services into the network on a widely distributed level. It extends cloud capabilities near data sources to overcome limitations of cloud-based IoT platforms such as mobility, location awareness and low latency. In spite of its increasing importance, there exist no readily available fog computing platforms which can help researchers to design and test real world fog applications. To this purpose, simulators and testbeds are adapted to enable the investigation of fog solutions. This paper presents a framework architecture to create fog virtualized environments. Its design meets the requirements of low cost, flexible setup and compatibility with real world technologies. Unlike current approaches, the proposed architecture allows for the testing of fog components with third-party systems through standard interfaces. A scheme to observe the behavior of the environment by monitoring network flow is provided. In addition, future developments and research directions are discussed.*

1. Introduction

A cloud-based model has become the default approach for Internet of things (IoT). Due to the limitations of cloud centralized architecture [Aazam et al. 2014], current IoT proposals are fomenting a major paradigm shift towards a decentralized model. Similar solutions in [Satyanarayanan et al. 2009], [Dinh et al. 2013], [Hu et al. 2015], [Bonomi et al. 2012] are bringing cloud computing capabilities near data sources to overcome limitations such as mobility, location awareness and low latency. We can summarize the initiatives through the following convergent points [Coutinho et al. 2016]: use of small clouds with virtualization support, distribution of computational resources on large scale, provision of IoT infrastructure as a service, and the offer of exclusive services using information that is only present on the local network or in its proximity.

In this scenario, fog computing is an emergent paradigm that extends cloud services to the edge of the network in an integrated way with switches, routers and IoT gateways[Bonomi et al. 2012]. The fog distributed approach reduces the amount of data transferred to the core network by capturing and processing the necessary data locally on each edge device; its low latency support brings real-time analysis to edge-based data by distributing the analytic process across the network[Bonomi et al. 2014].

The integration of the fog computing into the scope of IoT requires optimizing, deploying and spreading the concept of the cloud through a dense and geographically distributed platform. All the benefits of cloud should be preserved with fog,

including containerization, virtualization, orchestration, manageability and efficiency. Very recently, interesting works have appeared [Bonomi et al. 2014], [Yi et al. 2015a], [Dastjerdi et al. 2016], [OpenFog 2017a]; they propose a reference architecture, components, services, design goals and challenges of a fog platform. Nonetheless, as far as we know, there are no readily available fog computing platform that can help researchers to design and verify distributed algorithms on a truly IoT scale. Currently, simulation tools and testbeds are adapted to allow the experimental evaluation of fog solutions in specific scenarios and restricted conditions.

This paper presents a framework architecture to create fog testbeds in virtualized environments. Its design is based on solutions that meet the postulated requirements of low cost, flexible setup and compatibility with real world technologies. The components are based on Containernet [Containernet 2017], which allows Mininet [de Oliveira et al. 2014] to use Docker [Docker 2017] containers as virtual nodes. The main contribution is an architecture that enables the testing of fog components with third-party systems through standard interfaces. In addition, a scheme to observe the behavior of the environment is provided through network flow monitors.

The remainder of the paper is organized as follows. Section 2 introduces fog computing concepts. Section 3 presents related work. Section 4 addresses design requirements and technological choices. Section 5 illustrates the framework architecture. The conclusions and future works are presented in Section 6.

2. Fog Environment

The concept of fog computing is defined in [Bonomi et al. 2012] 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 computing paradigm. Their characteristics include edge location, location awareness, low latency, geographical distribution, support for large-scale sensor networks, very large number of nodes, support for mobility, real-time interactions, predominance of wireless access, heterogeneity, interoperability, federation and real-time data analysis.

There is a set of IoT solutions that can be supported by fog computing. The fog concept was introduced in [Bonomi et al. 2012] with applications for connected vehicle networks, smart grids and wireless sensor networks. Later, surveys in [Stojmenovic 2014] [Yi et al. 2015b] [Yi et al. 2015a] discussed scenarios that take advantage of fog computing such as smart health, smart building, smart cities, etc.

The fog computing model involves the running of applications concerning distributed components on the fog nodes between the IoT devices and the cloud data center. Fog instances can offer compute and storage resources to support the extension of the cloud services and real-time analysis to the edge of the network. The fog hierarchy makes the transmitted data actionable, meaningful and valuable by filtering information at different levels of the infrastructure.

Figure 1 presents the distributed physical components of a fog architecture [Yi et al. 2015a]. They are distributed hierarchically in a network infrastructure with at least three layers. At the edge of the network there are different IoT devices. The

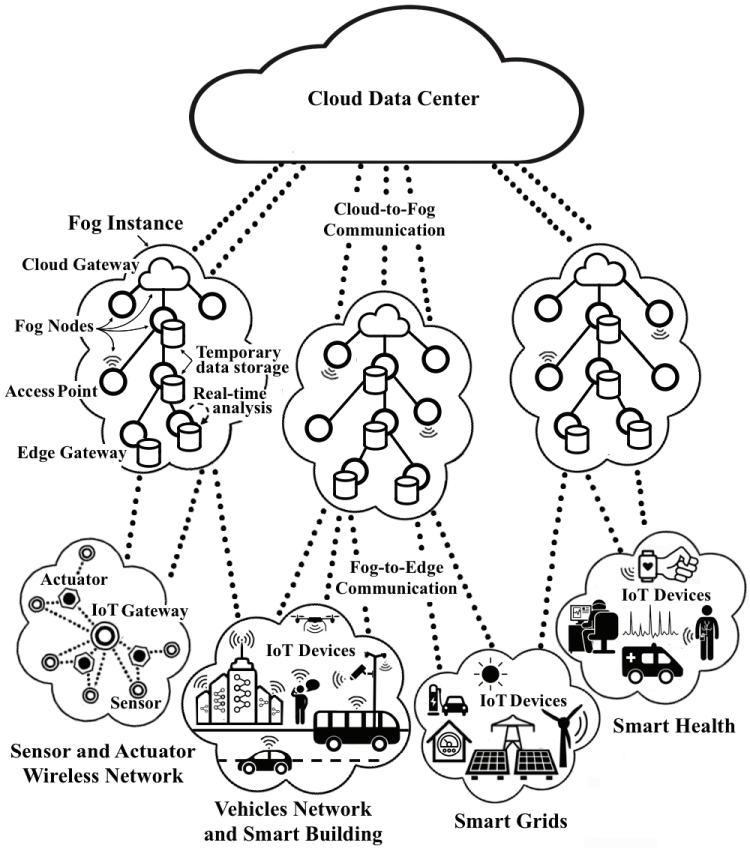

Figure 1. A Fog Architecture.

key components that can provide processing and storage capabilities at the edge of the network are called fog nodes [Tordera et al. 2016]. These fog nodes are involved in the data processing of different IoT applications. For example, sensor and actuator devices installed in parks, buildings and streets can provide valuable streams of data for monitoring, control, analysis and prediction applications. Its location depends on factors such as energy saving or application latency requirements.

Retrieving data from each sensor device is often challenging due to sensor power and communication constraints. An IoT gateway can aggregate and translate data from sensors before sending it onward by supporting different communication protocols. 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 devices.

The fog nodes must favor the access of IoT devices to the network, fog 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 temporary storage, preprocessing, data caching, geo-localization, service discovery, data analysis, load balancing, resource management and security. In addition, fog instances can provide access to virtualized resources through local fog nodes. They are also responsible for periodically filtering and sending information to the cloud. In the fog hierarchical organization, cloud gateways can act as proxy servers for larger cloud providers.

The services that support virtually unlimited resources for IoT applications are executed in the cloud. IoT applications can use both cloud and fog infrastructure to provide intelligent and cutting edge solutions to end users. The management software that globally coordinates the fog infrastructure are distributed in a number of services working together. The main purpose of these services is to provide acceptable levels of latency while optimizing the performance of fog applications. This can be achieved by efficient solutions that distribute task execution demand across optimal, well located and available fog nodes, using conventional and non-conventional algorithms.

3. Related Work

Fog computing has been steadily growing since the concept was adopted in 2012 [Bonomi et al. 2012]. As the pioneer in the fog field, Cisco solutions finds itself in a better position compared to other commercial fog products [Dastjerdi et al. 2016]. The Cisco IOx [Cisco IOx 2017] is a network operational system included in routers, switches and compute cards that makes processing and storage available to applications hosted in virtual machines (VMs). Distributed devices with IOx support in a network can function as the compute environment for fog applications. The IOx also supports open source environments based on Linux instances running in a hypervisor or virtual machine manager (VMM). The Cisco IOx Sandbox enables development and testing of fog computing applications using a desktop simulation approach. Together with IOx Fog Director [Fog Director 2017] it is possible to manage large-scale production deployments of IOx-enabled fog applications. In [DevNet 2017] a library with simulation tools, sample codes and templates to help deploy IoT applications for Cisco IOx platform is available. Examples of deployed IOx fog applications include basic services and protocols for IoT support written in Python, Java, Javascript and C.

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. Therefore, a standard fog architecture is necessary to allow integration between different solutions. In November 2015, the creation of the OpenFog consortium [OpenFog 2017a] united companies such as Cisco, Microsoft, Dell, Intel, ARM and universities such as Princeton in order to develop a standard fog architecture. The first version of OpenFog reference architecture [OpenFog 2017b] was released in February 2017.

In the absence of ready platforms, a viable alternative is to adapt IoT-specific testbeds to support experimental evaluation of fog solutions. The largest IoT testbed environments, such as [Adjih et al. 2015] and [Sanchez et al. 2014], are equipped with thousands of IoT devices geographically distributed. The authorized users can run applications over a large number of available resources to evaluate low level protocols, analysis techniques and services on a very large scale. However, since they are not intended for fog applications, a major effort is required to configure and deploy experiments.

Although desirable, in many cases the use of a real world testbed facility is expensive, time-consuming and does not provide a repeatable and controllable environment. Also, it has a selective access based on user research impact or political importance. Therefore, low cost-effective alternatives should be considered before a costly experiment so as to eliminate ineffective algorithms, policies and strategies. In [Yi et al. 2015a] a prototyping platform, consisting of two nodes connected to the Ama-

zon EC2 service was presented. The fog capabilities in each node were implemented with OpenStack[Openstack 2017]. A face recognition application was used to evaluate latency and bandwidth performance metrics in a scheme of VM offloading.

The adaptation of existing open source frameworks or simulators is currently a common approach in the fog computing area. Depending on research aims, the setup of testbeds to run specific applications can occur in two ways: adaptation of IoT simulators for fog experiments, or extension of cloud simulators for fog and IoT support.

NS-3 [NS-3 2017] and OMNET++[Varga and Hornig 2008] are examples of open source environments that allow for the extension of their functionality in a specific domain. OMNET++ is not a network simulator itself but an extendable library and framework for building network simulators. It provides a component architecture for simulation models such as Internet standard protocol, wireless sensor networks, etc. There are extensions for real-time simulation, network emulation, database integration, system integration, and several other functions that can be used for modeling a fog environment.

A case study on smart traffic management in [Dastjerdi et al. 2016] extended the CloudSim[Calheiros et al. 2011] 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 [Gupta et al. 2016], a simulator that supports edge and fog resources under different scenarios in a sense-process-actuate model [Gupta et al. 2016]. 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 computing 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.

4. Design Requirements and Technologies

In the development of a fog computing platform, the fog nodes and their network infrastructure are the main components to be deployed. In this work, the design of the fog architecture is based on solutions that meet the following requirements: 1) low cost deployment; 2) flexible setup; and 3) support to perform real world protocols and services.

Due to their high cost and restricted availability, proprietary solutions and large-scale IoT testbeds were not considered viable and practical alternatives to the current system architecture. Open source simulations 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. Extendible solutions such as OMNET++ do not have a specific framework implementation for the fog computing model. Mainstream network simulators such as NS-3 [NS-3 2017] face the same limitations. 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. An architecture that employs these solutions to create virtualized fog environments is presented in Section 5.

4.1. Mininet Network Emulator

Mininet [de Oliveira et al. 2014] is a network emulator that uses the Linux kernel to create an experimental network with virtual elements such as 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 [McKeown et al. 2008] for flexible custom routing and software-defined networking (SDN) [de Oliveira et al. 2014].

Mininet allows the creation of 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 network topology and link properties such as bandwidth, loss rate and delay can be configured with a extensible Python API. However, Mininet virtual hosts share the host machine file system by default. This makes it troublesome and challenging to use virtual hosts as fog nodes, since they should not use the same configuration or data files in distributed applications. Although there are ways to work around this limitation, a distinct filesystem for each virtual node can be maid by the integration of Mininet with containers.

4.2. Docker Container Platform

Docker [Docker 2017] is a software platform that allows the creation of containers. 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. 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.

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 (vEth) pairs to connect the virtual hosts to the virtual switches. Therefore, replacing Mininet virtual hosts with Docker containers is technically feasible.

4.3. Containernet Framework

Containernet [Containernet 2017] is a software project that extends the Mininet framework to allow the use of Docker containers as virtual nodes. In addiction, it introduces important capabilities to built networking and cloud and fog testbeds [Peuster et al. 2016].

Containernet framework API enables adding, connecting and removing containers dynamically from the network topology. These features allow emulate 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 single container, such as CPU time and memory available.

5. Framework Architecture

The aim of this work is to design an architecture that enables the deployment of the main components of a fog environment with third-party systems through standard interfaces. The flexible setup is achieved by deploying virtual nodes and virtual instances into an emulated fog environment running on a host machine.

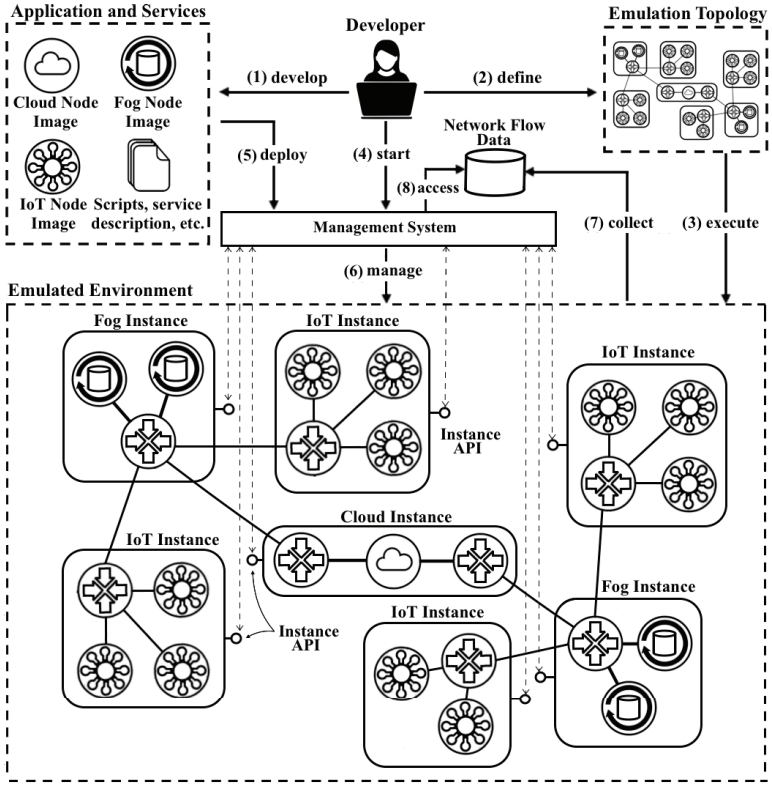

Figure 2. General emulation workflow. A example of a running emulation environment with seven virtual instances. Each virtual instance involves virtual switches and virtual nodes. The management system is responsible for uploading and starting the instances in the emulated environment. The defined topology determines the virtual connections configured 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 fog environment emulation can be created by using preconfigured container images. Each container image comprises part of a distributed application and its 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 image, cloud gateways to support virtualized resources for IoT applications are emulated. These virtual cloud nodes can form a virtual cloud instance (VCI) using open source technologies 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 filtering and sending information to a VCI.

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

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 platform 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).

5.1. Architectural 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 Containernet extends the Mininet functionalities to support Docker containers as virtual nodes. At the top of the Containernet are the framework APIs provided for developers.

The topology API interacts with the Containernet 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 IoT container image being used to initiate different VIoTIs, 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 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 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 in the emulated network.

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

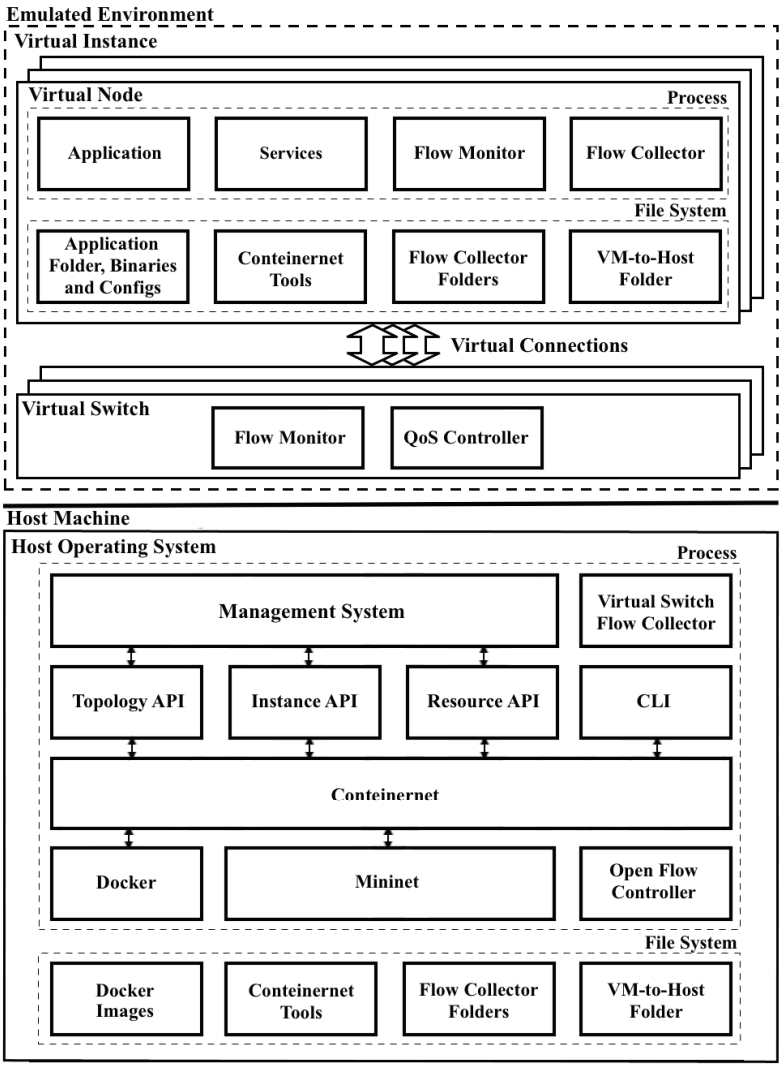

Figure 3. An overview of the architecture and principal components. The system design is customizable and provides APIs for its functionalities such as virtual instance management, container resource limitation or topology generation.

node contains a VM-to-Host folder which is connected to the same folder on the host machine. This facilitates data exchange between the virtual nodes and the host machine.

5.2. 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 the intelligent network routing based on SDN. The switches in the emulated network are configured by standard

```
1  # create and connect virtual instances to SDN switches with different link properties
2  g1= net.addIoTInstance("IoT gateway")
3  f1= net.addFogInstance("Fog device")
4  c1= net.addCloudInstance("Cloud gateway")
5  s1 = net.addSwitch("switch from IoT to Fog")
6  s2 = net.addSwitch("switch from Fog to Cloud")
7  net.addLink(g1, s1, delay="10ms", loss=2)
8  net.addLink(f1, s1, delay="50ms")
9  net.addLink(f1, s2, delay="50ms")
10 net.addLink(c1, s2, delay="100ms")
11 # assign resource models for each virtual instance (cpu, memory and storage)
12 r1 = ResModelA(max_cu=20, max_mu=40, max_su=60)
13 r1.assignInstance(g1)
14 r2 = ResModelB(max_cu=200, max_mu=150, max_su=200)
15 r2.assignInstance(f1)
16 r3 = ResModelC(max_cu=1000, max_mu=1500, max_su=2000)
17 r3.assignInstance(c1)
18 # instantiate, connect and start a particular interface for each virtual instance
19 api1 = IoTApi(port=8001)
20 api1.connectInstance(g1)
21 api1.start()
22 api2 = FogApi(port=8002)
23 api2.connectInstance(f1)
24 api2.start()
25 api3 = CloudApi(port=8003)
26 api3.connectInstance(c1)
27 api3.start()
28 # run the emulation
29 net.start()
```

Listing 1. An example of topology script which defines virtual instances of distinct types. They are connected with different link properties. Each virtual instance uses a specific resource model and connects to a particular instance API.

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.

5.3. Instance API

The management system needs to interact with the emulated environment to control virtual nodes within the 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 the integration and testing of third-part management systems.

The developers can implement their own management interfaces on top of a virtual instance API. The default approach is adding 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 19–27). With this flexible conception, it is possible the execution of different management strategies for each virtual instance in the emulated environment.

5.4. Resource API

Fog environments are characterized by the existence of network nodes with different capacities and resources. While cloud services provide virtually infinite compute resources to their clients, fog devices 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 12–17). These models are invoked when containers are allocated or released. They are used to compute CPU time, memory, and storage 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.

5.5. Flow Monitoring and Collection

A scheme to observe the behavior of the environment can be implemented by running network flow monitors and collectors. For implementation example, it is possible employs NetFlow [B. Claise, Ed. 2004] as the flow monitoring technology and NFDUMP [Haag, P. 2011] as the NetFlow collector and analysis tools. In this monitoring scheme, flow records can be 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* or other compatible flow analysis tool can be used to study the collected traffic.

The capture daemon running on a virtual node is responsible for collecting and saving flow statistics from the flow monitors of the virtual node. It is necessary to start one flow monitor process for each network interface. The traffic on the virtual nodes may overlap with the traffic monitored on virtual switches interfaces.

In the Containernet, virtual switches are not Docker containers. Just like in the Mininet, they are created in the host machine. Consequently, the flow monitor commands and capture daemons should be run in the host machine. A flow capture daemon running in the host machine is responsible for collecting and saving flow statistics from a specific virtual switch. Each flow capture daemon saves the flow records to a different directory. In this manner, is possible separate the collected data from different virtual switches.

6. Conclusion and Future Work

In this work, the design of a fog emulation testbed based on feasible deployment features of existing technologies was presented. The strengths of the proposed architecture are low cost, flexibility and support to perform real world protocols and technologies.

The current components were designed to prototype and test complex network services in a realistic environment running on low-cost machines. Such computers may not have processors with advanced virtualization functions to support the installation of the current cloud middleware. However, the proposed components can be extended to enable its connection to real cloud data centers.

The use of the same pre-created container images to start more than one container instances makes the environment setup process flexible. New types of container images

can be deployed to allow for the testing of different IoT solutions inside a fog architecture. Also, it makes easier the testing of solutions that deal with massive amounts of data streams such as big data applications. In a simple procedure, it is possible to build fog environments with multiple IoT data sources from a few container images.

The provision of standard interfaces allows for the testing of the third-party systems for functions such as resource management, virtualization, and service orchestration. For example, emerging systems with support to network function virtualization (NFV) and virtual network functions (VNF) can be connected with fog platforms to create complex network services (NS), which are controlled by a management and orchestration (MANO) system [Karl et al. 2016] [Sonkoly et al. 2015]. Also, it allows developers to use their tested applications into real world environments with minimal changes. In addition, the future progress in developing standard interfaces for fog computing can facilitate the integration of the third-party systems in different application scenarios.

Another important aspect of the architecture is the fidelity to the fog computing model. Using containers, it is possible to offer virtual fog nodes with computing resources in the emulated network. An advance in this regard is the use of containers not only in virtual nodes but also in the virtual switches created by Mininet. However, such capabilities require significant changes in Mininet software and as such were not considered. 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 principal future objective of this work is to enable the testing of distributed algorithms. Possible experiments may range from fog applications to the services related to the fog infrastructure itself. The provided features include a built-in scheme to observe the behavior of the environment by monitoring network flow inside the virtual nodes and switches. It provides insightful network traffic information of a running application so that researchers can evaluate their algorithms and network configurations in terms of system performance. It is possible to incorporate techniques for data visualization and real-time analysis in different levels of the network hierarchy to help developers debug problems and measure the performance of applications and services. 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 the presented architecture and its components can allow for the use of emulators for realistic and reproducible fog experiments.

References

Aazam, M., Khan, I., Alsaffar, A. A., and Huh, E.-N. (2014). Cloud of things: Integrating internet of things and cloud computing and the issues involved. In *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*, pages 414–419. IEEE.

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., et al. (2015). Fit iot-lab: A large scale open experimental iot testbed. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 459–464. IEEE.

B. Claise, Ed. (2004). Cisco systems netflow services export version 9. Available: https://tools.ietf.org/html/rfc3954. Accessed: 20 fev. 2017.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50.

Cisco IOx (2017). product page. Available: http://www.cisco.com/products/cloud-systems-management/iox/. Accessed: 20 fev. 2017.

Containernet (2017). project and source code. Available: https://github.com/containernet/containernet. Accessed: 20 fev. 2017.

Coutinho, A. A. T. R., Greve, F. G. P., and Carneiro, E. O. (2016). Computação em névoa: conceitos, aplicações e desafios. In *Minicursos - XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 266–315. SBC.

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

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

DevNet (2017). Cisco IOx developer tools and applications templates for download. Available: https://developer.cisco.com/site/iox/. Accessed: 20 fev. 2017.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.

Docker (2017). home page. Available: https://www.docker.com. Accessed: 20 fev. 2017.

Fog Director (2017). product page. Available: http://www.cisco.com/c/en/us/products/cloud-systems-management/fog-director/. Accessed: 20 fev. 2017.

Gupta, H., Dastjerdi, A. V., Ghosh, S. K., and Buyya, R. (2016). 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*.

Haag, P. (2011). Nfdump-netflow processing tools. Available: http://nfdump.sourceforge.net. Accessed: 20 fev. 2017.

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

Karl, H., Dräxler, S., Peuster, M., Galis, A., Bredel, M., Ramos, A., Martrat, J., Siddiqui, M. S., van Rossem, S., Tavernier, W., et al. (2016). Devops for network function vir-

tualisation: an architectural approach. *Transactions on Emerging Telecommunications Technologies*, 27(9):1206–1215.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

NS-3 (2017). Simulator home page. Available: https://www.nsnam.org. Accessed: 20 fev. 2017.

OpenFog (2017a). Consortium home page. Available: http://www.openfogconsortium.org/. Accessed: 20 fev. 2017.

OpenFog (2017b). Reference architecture for fog computing. Available: http://www.openfogconsortium.org/ra/. Accessed: 20 fev. 2017.

Openstack (2017). home page. Available: https://www.openstack.org. Accessed: 20 fev. 2017.

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

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

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4).

Sonkoly, B., Czentye, J., Szabo, R., Jocha, D., Elek, J., Sahhaf, S., Tavernier, W., and Risso, F. (2015). Multi-domain service orchestration over networks and clouds: a unified approach. *ACM SIGCOMM Computer Communication Review*, 45(4):377–378.

Stojmenovic, I. (2014). Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*, pages 117–122. IEEE.

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

Varga, A. and Hornig, R. (2008). 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*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

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

Yi, S., Li, C., and Li, Q. (2015b). A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM.