FogLedger: An Extensible Prototyping Environment for Integrated Edge/Fog and Blockchain Systems

***Abstract*—Recent research seeks to deploy responsive, safe, and decentralized Internet of Things (IoT) solutions by integrating edge/fog computing and distributed ledger technology (DLT) such as blockchain. However, the current prototyping tools are adapted to evaluate edge/fog solutions in limited environments with no DLT/blockchain support. This paper presents FogLedger, an extensible toolset integration to enable edge/fog-based DLT testbeds in virtualized environments. In order to facilitate the test of Edge/Fog-DLT solutions, FogLedger was developed as a Fogbed plugin component for emulating different DLT networks. An experiment to demonstrate the practical applicability of FogLedger is presented in different scenarios on a local network and on the cloud. In each scenarios, the behavior of an emulated Hyperledger Indy network is evaluated, focusing on the average time required to read and write transactions. Furthermore, future developments and research directions are discussed.**

I. Introduction

A cloud computing model [1] became the default approach to building internet of things (IoT) applications and services that needed resources on-demand, making it easier to store and process data in a centralized fashion. However, the real-world user experiences with the cloud computing model are limited due to excessive latency and longer response time [2].

In order to overcome the limitations of the cloud model, the fog computing paradigm [3] extends the services of the cloud closer to end-users through the use of intermediary resources embedded in the core network. In a complementary approach, edge computing[4] aims to decrease the response time of applications by processing data on servers or gateways located within the local network of the end-users or in their immediate vicinity. Operating in an integrated manner, fog and edge computing transfer a portion of the workload from the cloud to resources beyond remote data centers, thereby enhancing the overall scalability of services. This shift facilitates a reduction in latency and the volume of data transmitted to the cloud [5].

Besides increasing the performance of IoT applications, edge/fog models can also enhance security and help prevent data thefts [6]. However, edge/fog computing benefits are not enough to address all IoT security problems.

Recent years have witnessed IoT systems that approach distributed ledger technologies (DLT) [7], such as blockchain, to improve security and privacy [8] [9]. DLT/blockchain is a peer-to-peer (P2P) network that maintains a distributed and immutable history of transactions that can be validated and audited. It eliminates the need for a trusted third party and provides a secure, private, and decentralized way to share information between distributed applications [10] [11].

Recent research such as [12] [13] [9] argue that integrating edge/fog computing and DLT/blockchain technologies can deliver responsive, secure, interoperable, and decentralized IoT solutions. In [14], the benefits of merging edge/fog with DLT/blockchain to improve IoT systems are described.

However, integrating edge/fog with DLT/blockchain is not trivial. The DLT/blockchain network has a high computational cost with mechanisms such as smart contracts, encryption, and mining [15]. Therefore, edge/fog-based DLT (Edge/Fog-DLT) solutions demand enhanced cooperation between different components to overcome the mismatch between DLT/blockchain and IoT requirements [15].

A current issue in this area is the lack of supportive environments and tools for prototype Edge/Fog-DLT solutions [16] [17]. When developing Edge/Fog-DLT systems, prototyping tools can reduce deployment time, save costs, and improve the security of the offered services. These tools should allow the testing of Edge/Fog-DLT approaches, evaluate their performance, and validate the interactions between their components.

This paper presents FogLedger, a toolset integration to enable rapid prototyping of real-world Edge/Fog-DLT systems in an extensible and scalable way. The proposed architecture extends Fogbed [18] to provide developers with a virtual environment that can be deployed on a local network or in the cloud for prototyping Edge/Fog-DLT systems.

In order to facilitate the test of Edge/Fog-DLT solutions, FogLedger was developed as a Fogbed plugin component. This approach extends the emulation environment to incorporate different open-source DLT/blockchain, enabling prototyping and testing of Edge/Fog-DLT applications with flexible configuration, low cost, and real-world service support.

An example of how FogLedger is employed to prototype Edge/Fog-DLT systems is presented in different scenarios: the first scenario was conducted on a cluster of computers on a local network, the second and third scenario was on the Google Cloud Platform (GCP), and the fourth scenario was on the local network-GCP. In each scenario, the behavior of an emulated Hyperledger Indy [19] network is evaluated, focusing on the average time required to read and write transactions.

The rest of the paper is organized as follows. Section II presents related work. Section III addresses the technological choices and functional components of the proposed architecture. Section IV presents a prototyping example and results. The conclusions and future works are presented in Section V.

II. Related Work

In the literature surveys, recent proposals and new versions of edge/fog simulation and modeling tools indicate that the development of prototyping frameworks is still in progress and helpful in this area. Table I resumes the most cited fog simulation environments from the literature surveys.

TABLE I: The referenced fog simulations environments.

Year	Ref	Supported Models				
		Mobility	Cost	Energy	Failure	Blockchain
2016	[20]	Yes	Yes	No	No	No
2016	[21]	No	Yes	Yes	No	No
2017	[22]	Yes	Yes	Yes	No	No
2017	[23]	No	No	No	No	No
2017	[24]	No	Yes	No	Yes	No
2017	[25]	Yes	Yes	Yes	Yes	No
2018	[26]	Yes	Yes	No	No	No
2018	[27]	Yes	Yes	Yes	No	No
2019	[28]	No	No	No	No	No
2019	[29]	No	No	No	Yes	No
2019	[30]	No	Yes	Yes	No	No
2019	[31]	Yes	No	No	Yes	No
2020	[32]	No	No	Yes	Yes	No
2020	[33]	Yes	Yes	Yes	No	No
2021	[34]	No	No	No	No	Yes

Among the simulators presented in Table I, only FoBSim [34] supports blockchain (BC) technologies. The main purpose of FoBSim is to facilitate the test and validation of integrated fog-blockchain approaches. However, specific properties such as Merkle trees, digital signatures, and mining pools are not implemented in the current version of FoBSim [34]. Also, using FoBSim with real-world third-party DLT/blockchain technologies is not possible.

Despite the growing offer of simulation environments, the emulation approach has piqued the interest of researchers and developers of fog systems. The motivation for using the emulation model lies in the invariable fact that measurable differences will exist between the performance of the real-world system and the equivalent simulation model [35]. As models are approximations, these differences often lead to a lack of credibility that an emulation model tries to minimize.

Different emulation-based prototyping tools for fog computing have been proposed. MockFog [36] is a system for the emulation of fog computing infrastructure in arbitrary cloud environments. Edge machines are also deployed in the cloud and configured to closely resemble the real-world (or planned) fog infrastructure in an emulated fog environment. Fogify [37] is an interactive fog computing emulation framework that enables repeatable, measurable, and controllable IoT service modeling, deployment, and experimentation under realistic environment assumptions, faults, and uncertainties. The Fogbed emulator [18] uses different frameworks and open-source tools to create a virtual infrastructure that seeks to facilitate the deployment and testing of scalable fog computing solutions [38]. The Osmotic Computing (OC) [39] is an emulator with a similar architecture to Fogbed that addresses QoS requirements, resource provisioning, orchestration, and service level agreements (SLA). It is beneficial for comprehending the impact of processing power, workloads, and QoS requirements while preserving the service level agreements (SLAs).

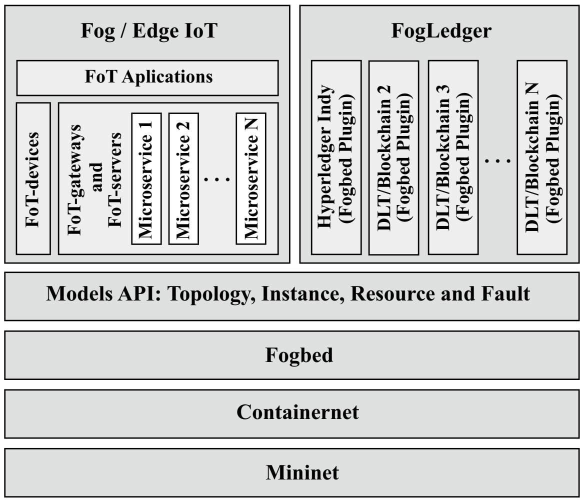
Fog / Edge IoT
FoT Aplications
FoT-devices
FoT-gateways and FoT-servers
Microservice 1
Microservice 2
Microservice N
FogLedger
Hyperledger Indy (Fogbed Plugin)
DLT/Blockchain 2 (Fogbed Plugin)
DLT/Blockchain 3 (Fogbed Plugin)
DLT/Blockchain N (Fogbed Plugin)
Models API: Topology, Instance, Resource and Fault
Fogbed
Containernet
Mininet

Fig. 1: The layers of emulation architecture.

In summary, as far as we know, no available emulation environment can help researchers design and verify Edge/Fog-DLT systems. Therefore, research concerning Edge/Fog-DLT prototyping platforms may help integrate technologies and accelerate the development of novel applications.

III. The Proposed Emulation Architecture

This section describes open-source solutions that allow real-world edge/fog applications and DLT/blockchain technologies to be emulated, integrated, and tested in a controllable environment. In addition, the proposed emulation architecture and its components at different levels are presented.

The components of the proposed architecture were founded on open-source and scalable technologies widely used by the research community and intended to meet the requirements of flexible configuration, low cost and compatibility with real world technologies [18]. Figure 1 shows the emulation proposed architecture, which employs the following tools and frameworks to create and deploy Edge/Fog-DLT virtual environments in an extensible and scalable way.

A. Emulation Technologies

A Docker container [40] is a software unit that includes everything needed to run an application, and the Docker platform uses it to create, deploy, and execute containerized applications and services. Docker lightweight container images can run quickly and be installed, copied, or moved reliably from one computing environment to another.

In the proposed architecture, containers package the main elements to emulate edge/fog and DLT/blockchain systems. However, other tools and frameworks described below were incorporated into its architecture to make a proper emulation and implement the hereby-involved concepts.

B. Mininet and Containernet

Mininet [41] is a framework that modifies the Linux kernel to emulate a network environment. It provides a virtual abstraction of communication channels, network nodes, and switches with native OpenFlow[41] support. The configuration of an emulated network can be created from a Python script that defines its virtual elements, their connections with each other, and their interaction during an experiment.

Containernet [42] is a fork of Mininet that replaces native virtual nodes with Docker containers. It isolates resources from the virtual nodes given that native virtual nodes of Mininet share the file system of the host machine. Also, Containernet allows adding or removing containers from the emulated network at runtime and managing the resources (e.g., CPU and memory) of each instance. It allows developers to move their tested applications and services directly into production environments with minimal additional changes.

C. Fogbed and Model APIs

Fogbed [38] is a fork of Containernet that introduces cloud, fog, and edge abstractions. These elements are represented through a virtual instance aggregating one or more virtual nodes and virtual switches as a unique manageable element.

Like Mininet, the configuration of the emulated environment is created from a script written in Python. However, with Fogbed, it is possible to define virtual instances (cloud, fog, and edge), their container images and virtual connections.

In addition, the RESTful API of Fogbed enables the remote execution of different virtual instances installed on distributed host machines. It is possible to define and choose different physical host machines (*workers*) where these virtual elements will be executed, allowing the elaboration of real-world topologies. Each distributed host machine runs a FogWorker agent that manages the execution of part of the experiment, such as edge/fog applications or an emulated DLT/blockchain network.

The Model APIs allow the configuring and managing the behavior of virtual instances emulated on Fogbed. Model APIs were inherited from Mininet, Containernet, or Fogbed and enable different operations such as configuring the virtual environment (topology API), managing virtual nodes (instance API), defining resources limits (resource API), and programming injection of faults in emulated environments (fault API).

D. The Edge/Fog IoT Architecture

The Edge/Fog IoT is the architecture that implements the edge/fog model and IoT system functionalities. This work extended Fogbed to include Docker container images of real-world IoT components with support for DLT/Blockchain technologies. It includes IoT devices, gateways, and applications used by Fogbed to emulate an IoT infrastructure.

To implement the Edge/Fog IoT architecture, the fog of things (FoT) model [43] was applied. It extends the self-organizing FoT for the IoT (SOFT-IoT) [44], a modular middleware approach based on the edge/fog paradigm.

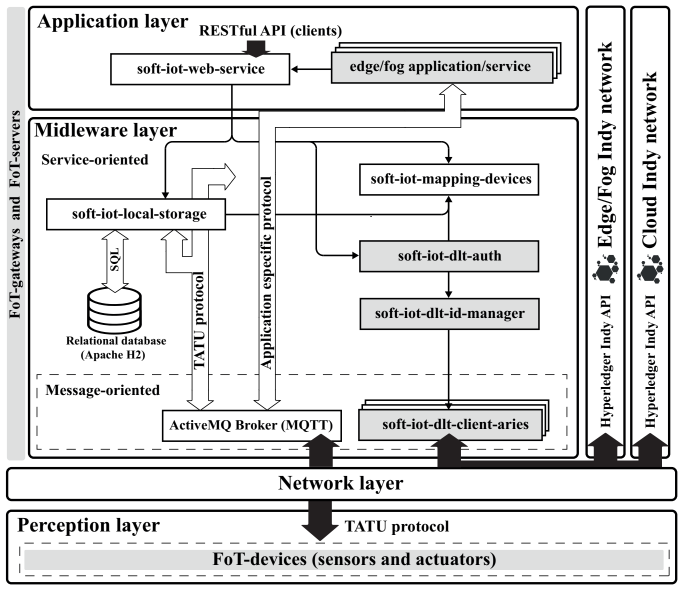

Fig. 2: The SOFT-IoT DLT Extended Architecture.

The SOFT-IoT platform[1] comprises library modules written in Java, JavaScript, C++, and Python that allow the development of real-world FoT elements. It uses low-cost hardware such as Arduino circuit boards connected with sensors/actuators to create FoT-devices, and the Raspberry Pi platform to deploy FoT-gateways. Also, FoT-servers [43] enhanced with specific capabilities such as long-term historical data can be provided as virtual machines (VM) in the cloud.

The SOFT-IoT microservices are deployed on an enterprise service bus (ESB) infrastructure [45]. They follow the open services gateway initiative (OSGi)[2], a middleware specification that combines a modular approach with the functionality of a service-oriented architecture (SOA) [45]. Figure 2 shows the core layers of the SOFT-IoT architecture [44] with modules (bundles) deployed in the FoT-gateways/FoT-servers.

In the perception layer, the FoT-devices implement the accessible thing universe (TATU) protocol [43], which standardizes the communication between FoT-devices and FoT-gateways. The TATU protocol extends the message queue telemetry transport (MQTT) protocol [46] through notations following the JavaScript Object Notation (JSON)[3] format.

The network layer transfers data from the perception layer to the middleware layer, where the used network technology depends on the connection requirements. The middleware layer employs message-oriented and service-oriented models [45]. Message-oriented communication is implemented between FoT-devices and FoT-gateways, and the service-oriented middleware allows the communication between modules deployed in FoT-gateways/FoT-servers.

The application layer provides FoT services that clients can request to obtain information and data about the FoT elements

[1]https://github.com/WiserUFBA
[2]https://www.osgi.org
[3]https://www.json.org/

[47]. It also provides a container for new FoT application services by extending the current SOFT-IoT capabilities towards innovative and sophisticated IoT solutions [44].

In Figure 2, there are SOFT-IoT modules suitable for all solutions that require IoT functionalities. They can be used to set up a minimal and lightweight, yet functional, FoT-gateway.

The first basic module is the soft-iot-mapping-device, that when started, loads a static list of FoT-devices from its configuration file. Also, it provides a service interface that allows other modules to configure (create, read, update and delete) and gather information about the FoT-devices, such as its identifier (ID), coordinate, sensor types, and collection/publishing time.

The Apache ActiveMQ[4] broker provides message-oriented communication through a lightweight implementation of the MQTT protocol. It is used by the soft-Iot-local-storage module that implements the TATU protocol and requests FoT-devices to send sensor data as configured in the soft-iot-mapping-devices module. The soft-iot-local-storage module saves the sensor data in a local H2[5] database and provides a service interface that allows other modules to access the historical sensor data. Finally, the soft-iot-web-service module uses the Apache CFX[6] to offer a RESTful web service API [45] that allows a remote or local client to request sensor data and other information by FoT-gateways and FoT-servers.

E. SOFT-IoT DLT Modules

The proposed IoT architecture extends the SOFT-IoT protocol and middleware to emulate edge/fog scenarios and experiments with DLT/blockchain technologies. In Figure 2, there are SOFT-IoT modules suitable for specific edge/fog application/services and DLT/Blockchain functionalities.

The experiment with FogLedger in Section IV involved the deployment of a application running on a virtual FoT-gateway capable of connecting to an Indy network. The bundles presented in Figure 2 were implemented as extensions in the SOFT-IoT architecture using DLT/Blockchain. These bundles run parallel with other services and applications, offering services for managing digital identities.

The soft-iot-dlt-auth and soft-iot-dlt-id-manager were implemented with DLT/blockchain for the identity management of clients and devices. The soft-iot-dlt-auth is the bundle responsible for controlling access to gateways by monitoring MQTT connections made by new devices. Through the soft-iot-dlt-auth, it is possible to send commands to devices connected to the gateway using the TATU protocol. For security reasons, this bundle is necessary to execute TATU protocol primitives, such as verify the identity of a FoT-device. The soft-iot-dlt-id-manager is the bundle that provides identification services such as unique gateway identifiers, network IP addresses, and group identifiers.

The soft-iot-dlt-client bundles implement the specific protocols necessary for communication with DLT/blockchain networks. The soft-iot-dlt-client-aires is the bundle that implements the Aries Cloud Agent (ACA) API and protocols for issuing, verifying, and holding verifiable credentials over the Hyperledger Indy network.

[4]https://activemq.apache.org/

[5]https://www.h2database.com/

[6]https://cxf.apache.org/

F. FogLedger and DLT/Blockchain

DLT/Blockchain is a set of software components that implements P2P networks, consensus protocol, smart contracts, identities, cryptography, and other concepts required to create secure, interoperable, and decentralized IoT solutions.

In this work, due to the complexity of emulating and running DLT/blockchain networks, we implemented FogLedger. It differs from Fogbed by facilitating the emulation of different DLT/blockchain technologies, where DLT/blockchain components were aggregated as a specific Fogbed configuration and installed as a FogLedger plugin. Each FogLedger plugin encapsulates particular details of a DLT technology and includes an API for proper configuration.

This approach allows Edge/Fog-DLT applications to be simulated in an environment with one or more DLT/Blockchain networks (multichain). Also, it offers flexibility to define edge, fog, cloud, or mixed topologies.

The following section describes a case study in which a plugin was implemented to extend Fogbed with container images of real-world DLT/Blockchain components based on Hyperledger Indy/Aries [19]. These components are used to create permissioned DLT for managing digital, decentralized, and self-sovereign identities (SSI) [48] based on the Hyperledger Indy and its frameworks Aries/Ursa [19].

IV. Edge/Fog-DLT Prototyping Example

This section describes using FogLedger to prototype an Edge/Fog-DLT application, taking the Hyperledger Indy framework as an example. In this regard, it addresses the requirements for running FogLedger, setting up a distributed emulated environment, and executing fog/edge or local experiments with DLT/blockchain.

A. The FogLedger Requirements and Workflow

FogLedger extends Fogbed, and Fogbed extends Containernet, which makes FogLedger inherit a set of requirements. First, to use FogLedger, it is necessary to install Fogbed and then FogLedger[7]. Due to Containernet requirements[8], Fogbed must be installed on host machines running Linux Ubuntu 20.04 and Python 3.6. Also, the Docker container images used to define the emulated elements must follow the running requirements of Containernet and contain the following packages/utilities: *bash*, *iproute*, and *iputils-ping*.

Before a local or distributed experiment can be executed, each host machine has to start a FogWorker service that waits for everything needed to run an emulation, such as topology scripts and configuration settings. The front-end machine runs the configuration script, which can be any host where Fogbed has been installed. However, when executing a distributed experiment, if the front-end machine is not a worker, it does not need to run the FogWorker service.

[7]The project codes are available on...

[8]https://github.com/containernet/containernet/wiki#container-requirements

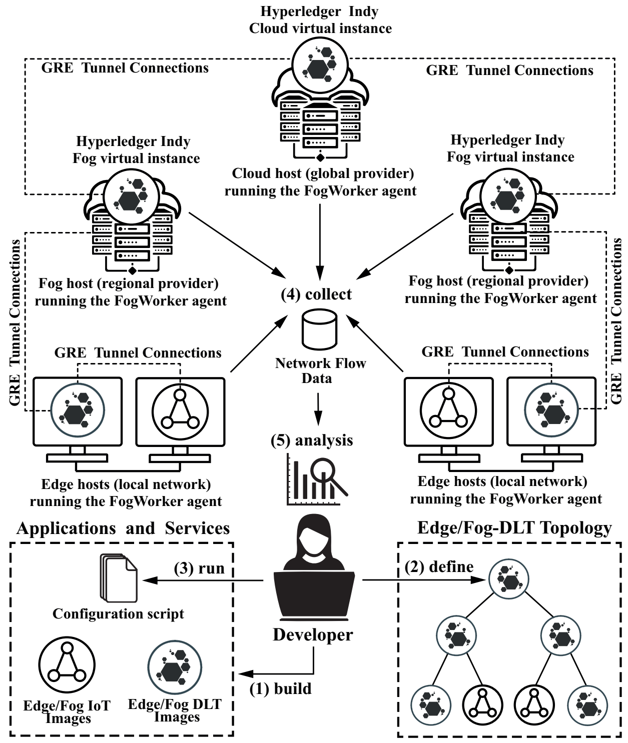

Fig. 3: The FogLedger distributed emulation workflow.

The experiment process running in the front-end machine uses the RESTful API of Fogbed to remotely deploy and start the execution of different virtual instances distributed on edge host machines (local network), fog host machines (regional provider), or cloud host machines (global providers). The configuration script defines which virtual instances and their network infrastructure are emulated inside each worker.

Figure 3 depicts the high-level workflow of a developer using the emulation environment. First, the developer builds and pushes the container images that will be used to set up a FogLedger emulation to the Docker Hub[9] container library (1). Using the Model APIs, the developer defines a distributed environment on which he wants to test the application (2) and runs the configuration script with the topology definition (3). Then, the experiment process uses the RESTful API of Fogbed to allocate workers, pull the necessary container images, create the virtual instances, and link them with other virtual instances in different hosts by setting up GRE tunnel connections as defined in the configuration script. After the FogLedger distributed environment starts, the experiment process executes other instructions defined in the script, such as DLT transactions and read/write operations. The experiment data, such as network flow statistics, can be collected (4) on each host machine and stored for future analysis (5).

B. Environment Configuration Example

The algorithm described in Listing 1 illustrates an experiment containing a Hyperledger Indy network. This network consists of five nodes distributed across fog and cloud host machines. As FogLedger is a Fogbed extension, every script must instantiate a *FogbedDistributedExperiment* (refer to line 7), providing the necessary emulation resources.

[9]https://hub.docker.com

Listing 1: Example of a script utilizing IndyBasic to create an Indy network with three workers.

```
 1  from fogbed import (
 2      FogbedDistributedExperiment,
 3      Worker)
 4  from fogledger.indy import (IndyBasic)
 5
 6  if (_name_ == '_main_'):
 7      exp = FogbedDistributedExperiment()
 8      indyNet = IndyBasic(exp=exp,
 9          trustees_path = 'tmp/trustees.csv',
10          prefix='Node',
11          number_nodes=5)
12
13      worker1 = exp.add_worker('cloud')
14      worker2 = exp.add_worker('fog1')
15      worker3 = exp.add_worker('fog2')
16
17      cloud = exp.add_virtual_instance('cloud')
18      worker1.add(cloud, reachable=True)
19
20      for i in range((len(indyNet.ledgers)−1)/2):
21          worker2.add(indyNet.ledgers[2∗i],
22              reachable=True)
23          worker3.add(indyNet.ledgers[2∗i +1],
24              reachable=True)
25      worker1.add(indyNet.ledgers[4], reachable=True)
26
27      exp.add_tunnel(worker1, worker2)
28      exp.add_tunnel(worker1, worker3)
29
30      try:
31          exp.start()
32          indyNet.start_network()
33      except Exception as ex:
34          print(ex)
35      finally:
36          exp.stop()
```

In line 8, a Hyperledger Indy network is defined. Lines 13-15 define the machines that will execute the experiment. Worker 1 runs the virtual cloud instance, and workers 2 and 3 run the virtual fog instances of Hyperledger Indy. The loop (lines 20-24) adds Hyperledger Indy nodes between hosts 2 and 3, then line 25 adds the last node in host 1. Lines 27 and 28 create tunnel connections between workers to enable communication. Finally, the experiment starts (line 31), and the Hyperledger Indy network is established (line 32).

If another Indy network needs to be created, a new IndyBasic object must be instantiated. In Fogbed, assigning a different value to the prefix variable is mandatory once the names of the virtual instances must be different.

Figure 4 shows the topology created from the configuration script in Listing 1. The GRE tunnels between workers make connecting virtual instances in different host machines possible. Therefore, any emulated Hyperledger Indy client can interact with the Hyperledger Indy network.

To interact with the Hyperledger Indy network, the client needs to possess the genesis file, which contains the information of the network nodes. In the example of the script

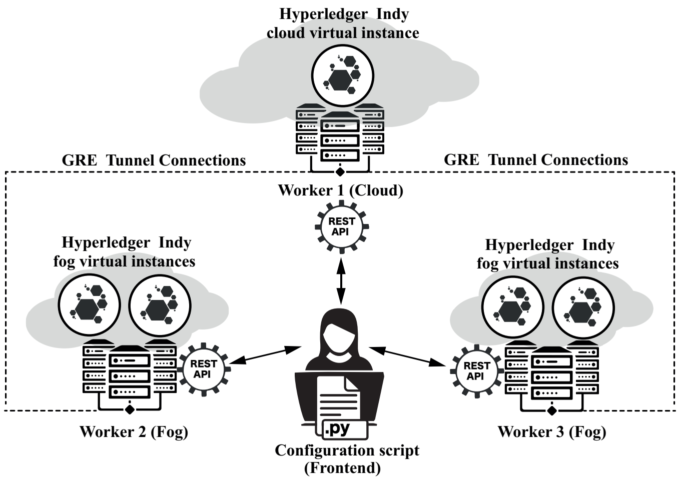
Hyperledger Indy cloud virtual instance
GRE Tunnel Connections
GRE Tunnel Connections
Worker 1 (Cloud)
REST API
Hyperledger Indy fog virtual instances
Hyperledger Indy fog virtual instances
REST API
REST API
Worker 2 (Fog)
.py
Worker 3 (Fog)
Configuration script (Frontend)

Fig. 4: The topology generated from the configuration script described in Listing 1 using three host machines (workers).

described in Listing 1, the developer could create a container with the Hyperledger Indy client and add the genesis file inside the fog virtual instance. The IndyBasic class also provides a method for the developer to transfer the data from the genesis file to the client.

C. Execution of Experiments and Results

After creating the network structure, execute three experiments to evaluate the functioning of FogLedger as an extension of the Fogbed, observing the behavior of the Indy network as a function of the rate of transaction requests per second. The experiments were emulated on different environments: i) on a local network, ii) with instances only on GCP in different zones, iii) with instances only on GCP in the same zone, and iv) client on a local network and validators nodes on GCP.

In the experiments, an agent developed by Pflanzner et al. [49] was used. This agent is capable of connecting to an Indy network and generating sample data for transmission to the network. It can handle up to 250 transaction (TX) requests per second, employing a multi-threaded approach to assess the read/write latency within the system [49]. Various transaction request rates were tested, ranging from 1 to 250 TX/s.

The execution of the agent developed by Pflanzner et al.[10] allows you to compare their work with the Indy network running inside Fogbed. It is expected that the behavior of the experiment carried out in GCP is similar to what was obtained in [49] since the tests were performed with the same types of instances and the number of validator nodes.

The experiments were performed by submitting transaction requests at scenarios with different rates of 1 TX/s, 50 TX/s, 100 TX/s, 150 TX/s, 200 TX/s, and 250 TX/s. These experiments were conducted on machines configured with the following settings:

[10]https://github.com/sed-szeged/IndyPerf

1) **Experiment on the local network:** Used five machines with 12 GB of RAM, Intel Core i3-8100 3.60 Ghz, and operating system (OS) Ubuntu 20.04.
2) **Experiment on Google Cloud Platform (GCP) in different zones:** Emulations on GCP used five instances of the E2-medium VMs (3.8 GHz, 2 vCPUs, 4 GB RAM) and OS Ubuntu 20.04. The virtual GCP instances in the experiment were in different zones (*Asia-northeast1-b, Southamerica-east1-b, Europe-west9-a, Australia-southeast1-b, and US-central1-a*).
3) **Experiment on GCP in the same zone:** Emulations on GCP used five instances of the E2-medium VMs (3.8 GHz, 2 vCPUs, 4 GB RAM) and OS Ubuntu 20.04. The virtual GCP instances were located in the same zone.
4) **Experiment on the local network-GCP:** Used one machine with 12 GB of RAM, Intel Core i3-8100 3.60 Ghz, and four instances of the E2-medium VMs (3.8 GHz, 2 vCPUs, 4 GB RAM), both with the operating system (OS) Ubuntu 20.04. The virtual GCP instances were located in different zones.

The experiments were executed with a network containing four validator nodes. Each scenario was repeated 30 times. From these inputs, the output data was the average latency of reading and writing nodes for each scenario of request rate.

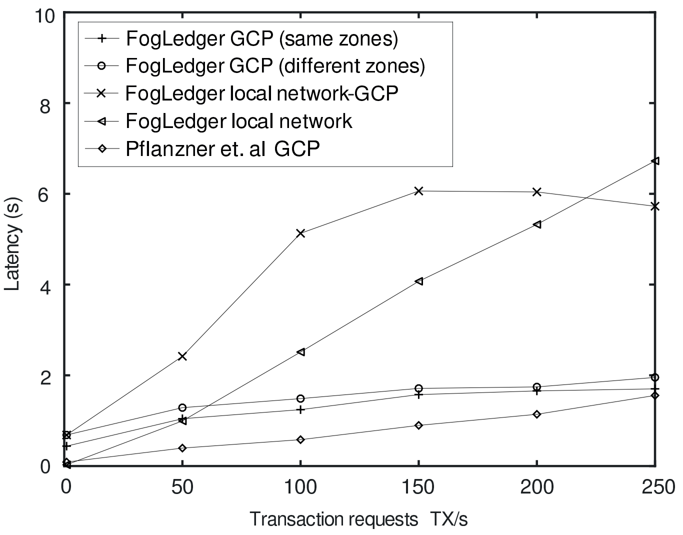

Fig. 5: Result of the experiments for reading latency with variability in the number of TX/s, comparing with the result obtained in Pflanzner et al. [49].

The experiment performed on the local network has lower latency than those performed using GCP resources when the transaction rate is up to 50TX/s. However, experiments ran only with GCP presented lower latency values and less variability when the transaction rate increased. We observed that the experiments performed with GCP resources located in the same zone had slightly lower latency values than those performed in different zones.

When comparing the reading experiments performed in the local network and local network-GPC, we noticed that the

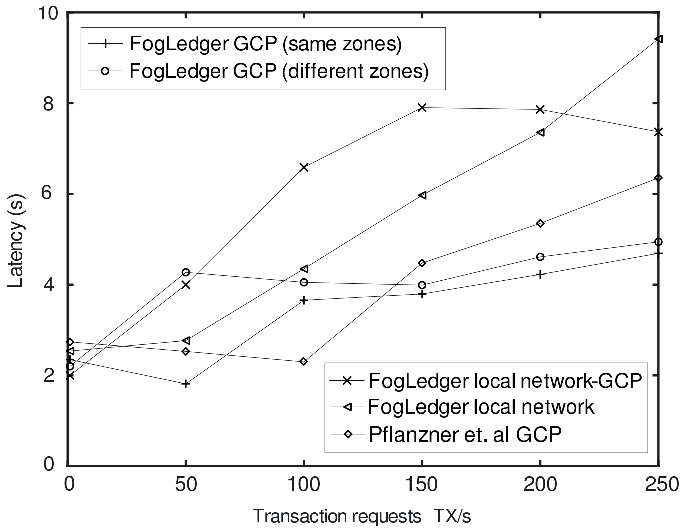

Fig. 6: Result of the experiments for writing latency with variability in the number of TX/s, comparing with the result obtained in Pflanzner et al. [49].

results obtained with the local network have lower latency up to the rate of 150 TX/s. The experiment executed in the local network-GCP presents lower latency from this rate. As for the results regarding writing, the experiment executed in the local network-GCP starts to show lower latency than the local network when the transaction rate is 250 TX/s.

The experiments show an approximately linear relationship between the number of transactions and latency. Latency increases with the number of TXs per second. The experiment on a local network and local network-GCP have a more significant growth tendency of the latency than the experiment carried out alone on the GCP. It should be noted that the GCP infrastructure performs better when the number of transactions increases. In the local network, because the machines are connected to a single switch with a lower backplane than the GPC, the bandwidth between hosts decreases as the transaction rate increases, increasing the latency.

In addition to analyzing the experiments executed in the local network, GCP in different zones, GCP in the same zone, and local network-GCP, we compared our results with the experiment presented by Pflanzner et al. [49] executed on GCP. The results can be observed in Figures 5 and 6, whose latency values are close to the results obtained in our experiments executed with the FogLedger only in GCP.

V. Conclusion and Future Work

This work extended the Fogbed/Containernet emulator to prototype and test Edge/Fog-DLT systems. In particular, it aimed to facilitate emulating virtual instances of Hyperledger Indy/Aries, a permissioned DLT specific for decentralized and self-sovereign identity. Hyperledger Indy was developed as the first FogLedger plugin because it provides fundamental reusable tools, libraries, and components for identifying individuals and ”things” on the network, delivering core functionality for building secure IoT systems.

The Edge/Fog-DLT prototyping example showed that using container images to offer edge/fog virtual instances with Hyperledger Indy in an emulated network is possible. Using FogLedger plugins to start more than one DLT/Blockchain network instance makes the environment setup easy and flexible.

New types of FogLedger plugins can be deployed to test different DLT solutions inside an emulation environment. In the future, continuing the work, we expect that other DLT/Blockchain instances can be virtualized, such as Tangle, Hyperledger Fabric, and Ethereum.

Currently, the configuration of a specific FogLedger is performed manually using Fogbed configuration scripts, where these configurations are extracted from docker-compose files existing in DLT/Blockchain distributions. A possible improvement in Fogbed would be the support for Docker compose files to specify the emulated components of a DLT/Blockchain network. These files are used to define and share applications consisting of multiple containers, such as DLTs.

The main future goals are to enable the testing of reliable distributed algorithms in Edge/Fog-DLT scenarios. Security, fault tolerance, and reliable management can be investigated by inserting simulated attacks and problems in the DLT/blockchain environment while running different applications. Finally, further research concerning a scalable architecture, efficient management of data, and virtualization of its components can allow for using FogLedger for realistic and reproducible Edge/Fog-DLT experiments.

References

[1] NIST, “The nist definition of cloud computing,” Available: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf, 2011, accessed: 20 jan. 2023.

[2] M. Aazam, I. Khan, A. A. Alsaffar, and E.-N. Huh, “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*. IEEE, 2014, pp. 414–419.

[3] 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.

[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] G. Senthil, R. Prabha, A. Pomalar, P. L. Jancy, and M. Rinthya, “Convergence of cloud and fog computing for security enhancement,” in *2021 fifth international conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE, 2021, pp. 1–6.

[7] A. Deshpande, K. Stewart, L. Lepetit, and S. Gunashekar, “Distributed ledger technologies/blockchain: Challenges, opportunities and the prospects for standards,” *Overview report The British Standards Institution (BSI)*, vol. 40, p. 40, 2017.

[8] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, “A survey on iot security: Application areas, security threats, and solution architectures,” *IEEE Access*, vol. 7, pp. 82 721–82 743, 2019.

[9] H. Baniata and A. Kertesz, “A survey on blockchain-fog integration approaches,” *IEEE Access*, vol. 8, pp. 102 657–102 668, 2020.

[10] G. Zyskind, O. Nathan *et al.*, “Decentralizing privacy: Using blockchain to protect personal data,” in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.

[11] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The blockchain as a software connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2016, pp. 182–191.

[12] N. Tariq, M. Asim, F. Al-Obeidat, M. Zubair Farooqi, T. Baker, M. Hammoudeh, and I. Ghafir, "The security of big data in fog-enabled iot applications including blockchain: A survey," *Sensors*, vol. 19, no. 8, p. 1788, 2019.

[13] T. Alam and M. Benaida, "Blockchain, fog and iot integrated framework: Review, architecture and evaluation," *Tanweer Alam. Mohamed Benaida." Blockchain, Fog and IoT Integrated Framework: Review, Architecture and Evaluation.", Technology Reports of Kansai University*, vol. 62, no. 2, 2020.

[14] R. B. Uriarte and R. De Nicola, "Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 22–28, 2018.

[15] L. Ghiro, F. Restuccia, S. D'Oro, S. Basagni, T. Melodia, L. Maccari, and R. L. Cigno, "What is a blockchain? a definition to clarify the role of the blockchain in the internet of things," *arXiv preprint arXiv:2102.03750*, 2021.

[16] S. Svorobej, P. Takako Endo, M. Bendechache, C. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, J. Byrne, and T. Lynn, "Simulating fog and edge computing scenarios: An overview and research challenges," *Future Internet*, vol. 11, no. 3, p. 55, 2019.

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

[18] 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.

[19] M. P. Bhattacharya, P. Zavarsky, and S. Butakov, "Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, 2020, pp. 1–7.

[20] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in *2016 Cloudification of the Internet of Things (CIoT)*. IEEE, 2016, pp. 1–6.

[21] 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.

[22] 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.

[23] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[24] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2017, pp. 105–114.

[25] J. Byrne, S. Svorobej, A. Gourinovitch, D. M. Elango, P. Liston, P. J. Byrne, and T. Lynn, "Recap simulator: Simulation of cloud/edge/fog computing scenarios," in *2017 Winter Simulation Conference (WSC)*. IEEE, 2017, pp. 4568–4569.

[26] J. Hasenburg, S. Werner, and D. Bermbach, "Fogexplorer," in *Proceedings of the 19th International Middleware Conference (Posters)*, 2018, pp. 1–2.

[27] T. Qayyum, A. W. Malik, M. A. K. Khattak, O. Khalid, and S. U. Khan, "Fognetsim++: A toolkit for modeling and simulation of distributed fog environment," *IEEE Access*, vol. 6, pp. 63 570–63 583, 2018.

[28] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, "Fogbus: A blockchain-based lightweight framework for edge and fog computing," *Journal of Systems and Software*, vol. 154, pp. 22–36, 2019.

[29] S. Forti, A. Ibrahim, and A. Brogi, "Mimicking fogdirector application management," *SICS Software-Intensive Cyber-Physical Systems*, vol. 34, no. 2, pp. 151–161, 2019.

[30] X. Liu, L. Fan, J. Xu, X. Li, L. Gong, J. Grundy, and Y. Yang, "Fogworkflowsim: an automated simulation toolkit for workflow performance evaluation in fog computing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1114–1117.

[31] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91 745–91 758, 2019.

[32] S. Forti, A. Pagiaro, and A. Brogi, "Simulating fogdirector application management," *Simulation Modelling Practice and Theory*, vol. 101, p. 102021, 2020.

[33] C. Puliafito, D. M. Gonçalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, and L. F. Bittencourt, "Mobfogsim: Simulation of mobility and migration for fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102062, 2020.

[34] H. Baniata and A. Kertesz, "Fobsim: an extensible open-source simulation tool for integrated fog-blockchain systems," *PeerJ Computer Science*, vol. 7, p. e431, 2021.

[35] I. McGregor, "The relationship between simulation and emulation," in *Proceedings of the Winter Simulation Conference*, vol. 2, 2002, pp. 1683–1688 vol.2.

[36] J. Hasenburg, M. Grambow, E. Grünewald, S. Huk, and D. Bermbach, "Mockfog: Emulating fog computing infrastructure in the cloud," in *2019 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 2019, pp. 144–152.

[37] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 42–54.

[38] 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.

[39] A. Buzachis, D. Boruta, M. Villari, and J. Spillner, "Modeling and emulation of an osmotic computing ecosystem using osmotictoolkit," in *2021 Australasian Computer Science Week Multiconference*, 2021, pp. 1–9.

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

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

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

[43] 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.

[44] L. Andrade, C. Lira, B. de Mello, A. Andrade, A. Coutinho, and C. Prazeres, "Fog of things: Fog computing in internet of things environments," *Special Topics in Multimedia, IoT and Web Technologies. Springer*, pp. 23–50, 2020.

[45] C. Prazeres, J. Barbosa, L. Andrade, and M. Serrano, "Design and implementation of a message-service oriented middleware for fog of things platforms," in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 1814–1819.

[46] MQTT, "Mqtt: The standard for iot messaging," Available: https://mqtt.org/, 2020, accessed: 20 jan. 2023.

[47] L. Andrade, C. Lira, B. Mello, A. Andrade, A. Coutinho, F. Greve, and C. Prazeres, "Soft-iot platform in fog of things," in *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, 2018, pp. 23–27.

[48] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[49] T. Pflanzner, H. Baniata, and A. Kertesz, "Latency analysis of blockchain-based ssi applications," *Future Internet*, vol. 14, no. 10, p. 282, 2022.