The Journal of Systems and Software 154 (2019) 22–36

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

New Trends and Ideas

FogBus: A Blockchain-based Lightweight Framework for Edge and Fog Computing

Shreshth Tuli[a,b], Redowan Mahmud[a,*], Shikhar Tuli[c], Rajkumar Buyya[a]

[a] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia
[b] Department of Computer Science, Indian Institute of Technology (IIT), Delhi, India
[c] Department of Electrical Engineering, Indian Institute of Technology (IIT), Delhi, India

ARTICLE INFO

Article history:
Received 2 October 2018
Revised 13 March 2019
Accepted 12 April 2019
Available online 13 April 2019

Keywords:
Fog computing
Edge computing
Cloud computing
Internet of Things(IoT)
Blockchain

ABSTRACT

Recently much emphasize is given on integrating Edge, Fog and Cloud infrastructures to support the execution of various latency sensitive and computing intensive Internet of Things (IoT) applications. Although different real-world frameworks attempt to assist such integration, they have limitations in respect of platform independence, security, resource management and multi-application execution. To address these limitations, we propose a framework, named *FogBus* that facilitates end-to-end IoT-Fog(Edge)-Cloud integration. FogBus offers platform independent interfaces to IoT applications and computing instances for execution and interaction. It not only assists developers to build applications but also helps users to run multiple applications at a time and service providers to manage their resources. Moreover, FogBus applies Blockchain, authentication and encryption techniques to secure operations on sensitive data. Due to its simplified and cross platform software systems, it is easy to deploy, scalable and cost efficient. We demonstrate the effectiveness of FogBus by creating a computing environment with it that integrates finger pulse oximeters as IoT devices with Smartphone-based gateway and Raspberry Pi-based Fog nodes for Sleep Apnea analysis. We also evaluate the characteristics of FogBus in respect of other existing frameworks and the impact of various FogBus settings on system parameters through deployment of a real-world IoT application. The experimental results show that FogBus is comparatively lightweight and responsive, and different FogBus settings can tune the computing environment as per the situation demands.

1. Introduction

Internet of Things (IoT) paradigm enables different sensors, machines and objects to perceive external ambiance and exchange data via Internet. It supports data integration and analysis using applications to identify events and trigger necessary actions through actuators. Thus, IoT helps in building cyber-physical systems with limited human intervention in different application domains such as healthcare, transportation, utility infrastructure, agriculture and surveillance management services (Gubbi et al., 2013). It leads to the realization of smarter environments such as smart health, smart cities, smart transport, smart grid, smart farming, and smart security enabling real-time decisions. For example, surveillance for disaster detection and prevention needs IoT data analysis to be completed within strict deadlines to enable response teams for rapid and timely action. Based on current trends, it is expected that by 2025 such smart environments will incorporate over 1 trillion IoT devices with 50% increased demand for latency sensitive applications (McKinsey & Company, May, 2018).

By hosting applications for different smart systems, existing Cloud datacenters can offer a possible way to deal with large number of geographically distributed IoT devices (Muhammad et al., 2017). However, Cloud datacenters reside at a multi-hop distance from IoT devices that increases communication delay in data transfer and application service delivery (Afrin et al., 2015). For latency-sensitive applications such as remote patient monitoring, forest fire detection and traffic signal management, this high-latency communication affects the service quality drastically. Moreover, IoT devices can generate huge amount of data within minimal time. When large number of IoT devices simultaneously transfer data to Cloud datacenters through Internet, severe network congestion occurs. To overcome these limitations of Cloud-centric IoT model, Fog and Edge computing paradigms have been emerged (Bonomi et al., 2012). Both utilize edge computing resources for decision making and executing IoT applications in real-time. Smart

[*] Corresponding author.
E-mail address: mahmudm@student.unimelb.edu.au (R. Mahmud).

https://doi.org/10.1016/j.jss.2019.04.050
0164-1212/© 2019 Published by Elsevier Inc.

devices with computing processors such as Raspberry Pi devices, personal computers, mobile phones, routers and micro-datacenters can provide potential edge resources (Mahmud et al., 2018b). Based on the operations running on edge resources, some consider Fog and Edge computing paradigm similar and use them interchangeably while others treat Edge computing as a subset of Fog computing (Chang et al., 2019), which is in accord with the view taken by us in this paper.

Fog environment manages an intermediate layer between IoT devices and Cloud datacenters, and supports a variety of IoT applications which exhibit different characteristics. Some of these applications are compute intensive while others are network or storage intensive. The computing instances of Fog are known as Fog nodes and deployed across the edge network in distributed manner. Physical resources within Fog nodes, different service-oriented frameworks and various application programs assist Fog to extend Cloud-like service models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) closer to the IoT data sources (Yi et al., 2015b). Consequently, it reduces service latency and network congestion, and improves Quality of Service (QoS) and user experience (Mahmud et al., 2018c). However, most of the Fog nodes are resource constrained and heterogeneous in terms of their computing capabilities, operating systems and peer-to-peer communication standards (Gupta et al., 2017). With limited resources, it is not possible to accommodate every IoT application at the Fog layer. Therefore, seamless integration of IoT-enabled systems with Fog and Cloud infrastructures is required so that both edge and remote resources can be harnessed according to the requirements of applications (Mahmud et al., 2018a). In this integration, Cloud-centric top-down approach for managing Fog-based resources becomes infeasible when IoT-data is received at a higher frequency for processing. On such occurrence, rather than relying on centralized resource management policies, it is effective to take decisions locally and provision resources following distributed bottom-up approach. Moreover, while placing and executing applications on integrated environment, both internal and external operations get obstructed by the heterogeneity of computing instances. In this circumstance, platform independent techniques can overcome the impediments of peer to peer communication and application runtime environment. Nevertheless, the implementation of an integrated environment going beyond the infrastructure and platform-level diversity with decentralized resource management policies is a challenging task. Its complexity is further intensified due to coexistence of multiple decision-making entities, unaware network topology security and scalability issues.

In literature, there exists a number of works implementing software frameworks for integrating IoT-enabled systems, Fog and Cloud infrastructures (Rahmani et al., 2018; Dubey et al., 2017; Yangui et al., 2016; Bruneo et al., 2016). However, these frameworks barely support simultaneous execution of multiple applications and platform independence. Moreover, they offer narrow scope to application developers and users to tune the framework according to individual requirements. These frameworks exploit Cloud resources for data storage and often compel energy constrained IoT devices to process the raw data. To reduce the management overhead, existing frameworks apply centralized techniques that eventually degrade the QoS. They also confine the concentration on few security aspects which in consequence increases vulnerability of the integrated environment. In order to overcome such limitations of available frameworks, we develop a lightweight framework named *FogBus*.

FogBus allows an end-to-end implementation of integrated IoT-Fog-Cloud environment by harnessing variety of edge network resources. It supports platform independent application execution and node-to-node interaction. It is designed to assist in implementation of (a) different concurrent/application programming models such as SPMD (Single Program and Multiple Data), workflows and streams and (b) resource management and scheduling policies for executing these kinds of applications in Fog and Cloud integrated computing environments. To ensure data integrity, protection and privacy, FogBus also implements Blockchain (Zyskind et al., 2015) and applies authentication and encryption techniques which consequently increase its reliability.

The **major contributions** of this work are as follows:

- We propose a lightweight framework named FogBus for integrating IoT-enabled systems, Fog and Cloud infrastructure to harnesses edge and remote resources according to application requirements. It applies Blockchain to ensure data integrity while transferring confidential data.
- We design a platform independent application execution and node-to-node interaction architecture to overcome heterogeneity within the integrated environment.
- We develop a prototype application system for Sleep Apnea data analysis, evaluate characteristics of FogBus and illustrate how an application (in healthcare domain) composed using SPMD model can be realised using different settings of FogBus to process IoT-data in integrated computing environment.

The rest of the paper is organized as follows. Section 2 highlights key elements of existing frameworks and compare them with proposed framework. In Section 3, the description of FogBus framework is provided. The design and implementation of FogBus are discussed in Section 4. In Section 5 a case study on Sleep Apnea analysis is presented. Sections 6 and 7 look into evaluation of FogBus in respect of framework characteristics and application deployment respectively. Section 8 concludes the paper proposing future works to improve FogBus.

2. Related work

The existing frameworks that integrate different IoT-enabled systems with Fog and Cloud infrastructure are roughly classified into two types. The first type focuses on application specific prototypes while the other offers generalized PaaS model. Table 1 provides a brief summary of these frameworks.

Rahmani et al. (2018) develop a prototype-based framework for IoT-enabled health-care system with an architecture of smart gateways to facilitate local storage and data processing. In this framework, Cloud acts as a back-end infrastructure for data analysis and decision making. The framework depends on operating system-level security features. Another prototype framework for smart health-care is developed by Dubey et al. (2017). Intel Edison and Raspberry Pi devices are used in the framework as Fog nodes. Through role-based authentication, it ensures data privacy. In this framework, Cloud is partially adopted for storing data. Azimi et al. (2017) also discuss a hierarchical prototype framework for health-care solutions. The health analytics are divided into two parts and are placed separately in Cloud and Fog infrastructures. The framework follows MAPE-K model to conduct the computations that supports execution of diverse applications and provides data encryption.

Moreover, Gia et al. (2017) present a low-cost health monitoring framework to facilitate autonomic analysis of IoT data. In the framework, IoT devices are computationally capable and can pre-process raw data. They can also forward data to different Fog nodes consuming less energy. In Fog layer, distributed database is managed for data categorization and security. Akrivopoulos et al. (2017) develop another prototype framework that allows users to share health data and notify during emergency. Each operation within the framework is managed by a Spark IoT Platform Core residing at the Cloud. The framework uses encryption and authentication techniques for security.

Table 1
Summary of the literature study.

Work	Integrates			Platform independent	Security features			Supports multi-applications	Targets		Decentralized management
	IoT	Fog	Cloud		Integrity	Authentication	Encryption		Developers	Users	
Rahmani et al. (2018)	✓	✓	✓			✓	✓			✓	✓
Dubey et al. (2017)	✓	✓				✓		✓		✓	✓
Azimi et al. (2017)	✓	✓	✓			✓	✓	✓		✓	✓
Gia et al. (2017)	✓	✓				✓	✓			✓	✓
Akrivopoulos et al. (2017)	✓	✓	✓			✓	✓			✓	
Chen et al. (2017)	✓	✓								✓	✓
Craciunescu et al. (2015)	✓	✓								✓	✓
Hu et al. (2017)	✓	✓	✓							✓	
Yangui et al. (2016)	✓	✓	✓	✓		✓	✓	✓	✓		
Bruneo et al. (2016)	✓	✓	✓	✓		✓	✓	✓	✓		
Verba et al. (2017)	✓	✓		✓		✓	✓	✓	✓		✓
Yi et al. (2015a)	✓	✓	✓	✓		✓		✓	✓		
Vatanparvar et al. (2015)	✓	✓		✓			✓	✓	✓	✓	✓
Chang et al. (2017)	✓	✓		✓	✓	✓		✓	✓		✓
Mohamed et al. (2017)	✓	✓	✓	✓	✓	✓		✓	✓		✓
FogBus [this work]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Chen et al. (2017) and Craciunescu et al. (2015) develop separate prototype framework for smart city surveillance and gas-leak monitoring system respectively. In both frameworks the Fog infrastructure conducts necessary data processing and decision-making operations. Remote Cloud is partially adopted in these frameworks for managing a record of IoT-data. However, the authors neither mention any security features nor techniques for managing heterogeneity of the Fog nodes within the framework. Likewise, Hu et al. (2017) omit the security issues of their developed prototype framework for face identification system. In this framework, a centralized Cloud manages all resources of integrated environment and offloads partial computational tasks to Fog infrastructure. After completing tasks on Fog, only the results are sent back to Cloud for further analysis and storage.

To promote IoT, Fog and Cloud integration, a Cloud-centric PaaS framework is developed by Yangui et al. (2016) that automates the provisioning of applications. The PaaS facilitates developing diverse applications, their deployment and management of Fog nodes. The framework can deal with heterogeneity of the nodes. In addition, security features from Cloud Foundry architecture are extended in the framework. Similarly, Bruneo et al. (2016) propose a Fog-centric PaaS framework named *Stack4Things* for deploying and executing multiple applications over computationally sound IoT devices. There, Fog infrastructure acts as a centralized programmable coordinator. The framework applies Cloud-based security features and deal with diverse applications surpassing heterogeneity of the instances.

Moreover, Verba et al. (2017) propose a gateway architecture that offers PaaS for integrating Fog nodes and IoT devices. The gateways assist messaging communication with authentication. The framework supports horizontal integration of gateways and Cloud datacenters for application deployment and task migration. Yi et al. (2015a) also propose a comprehensive *Cloudlet-based PaaS* framework for integrated environment. The framework requires resource-enriched Fog nodes to run and each node including IoT devices are required to be virtualized. Existing authentication techniques help to secure the framework operations. Additionally, Vatanparvar et al. (2015) develop a PaaS framework that manages electricity usage in a home and in micro grid levels over Fog infrastructures. It deals with the heterogeneity of Fog nodes and IoT devices, and encrypts data.

The PaaS framework called *Indie Fog* proposed by Chang et al. (2017) utilizes user's networking devices to execute IoT applications. In this framework, core services and resource management instructions are extended from Cloud datacenters to Fog nodes based on application requirements. It supports Cluster of Fog nodes and incorporates user's hand-held devices. For security, it runs a registry service. Mohamed et al. (2017) also discuss a service-oriented framework for managing smart-city based services through Fog and Cloud infrastructures. In this framework, services are classified in two types. The first one manages the core operations of the framework including resource management and security. and another type incorporates the requirements of specific applications. The security of the framework is ensured by authentication and access control mechanisms. In aforementioned frameworks, security issues are exploited from narrow perspective and computing capabilities of both edge and remote resources have not been fully leveraged. Due to pushing computation towards IoT devices or resource enriched Fog nodes, their overall deployment cost and energy consumption also increase. In addition, most of the frameworks overlook heterogeneity within computing infrastructures and it is difficult for them to support multiple applications simultaneously. However, our proposed FogBus framework offers service tuning facility to both users and providers. It ensures data integrity through Blockchain and assists user authentication and data encryption side by side. FogBus expands the execution platform for different applications from resource constrained Fog nodes to Cloud datacenters going beyond their diversity.

3. Fogbus framework

The FogBus framework integrates diverse hardware instruments through software components that offer structured communication and platform independent execution of applications. High level view of integrated IoT-Fog-Cloud environment using FogBus is shown in Fig. 1. It includes the following elements.

3.1. Hardware instruments

The hardware instruments such as IoT devices, Fog Gateway Nodes (FGN), Fog Computational Nodes (FCN) and Cloud datacenters that form the base for FogBus are discussed below.

IoT devices: IoT devices contain sensors that perceive the external environment and actuators that convert any given command to physical actions. Usually, IoT devices are energy and resource constrained and act as mere data producer or consumer. In some cases, IoT devices are equipped with limited computation capabilities to pre-process raw data. FogBus allows IoT devices to connect with proximate gateways via wireless or wired communication protocols such as Zigbee, Bluetooth and NFC. The sensing

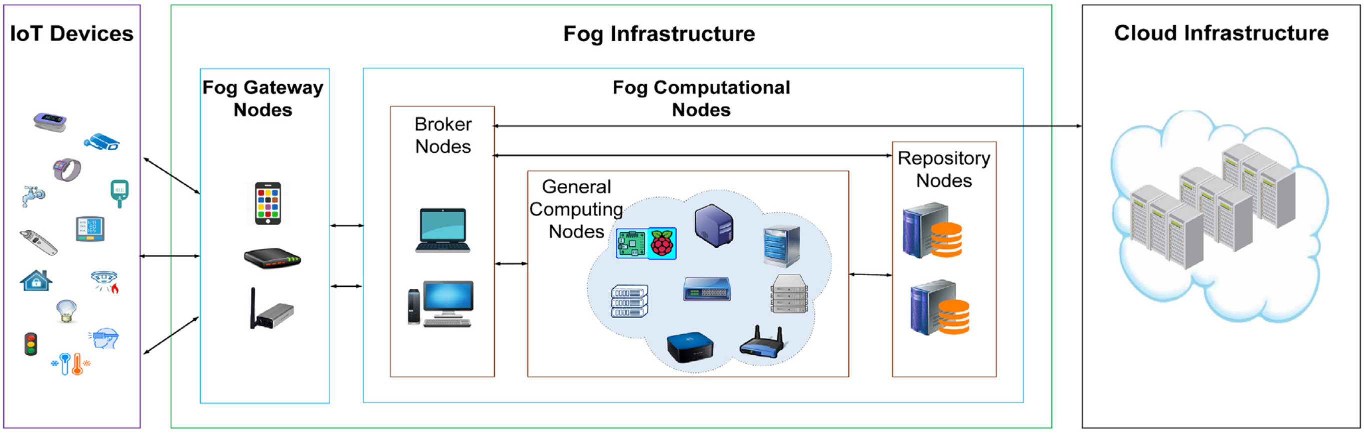


Fig. 1. High level view of integrating IoT-Fog and Cloud through FogBus framework.

frequency of IoT devices can be tuned according to context of the system where the format of IoT-data varies from device to device.

Fog Gateway Nodes (FGN): In FogBus, Fog Gateway Nodes (FGN) are the entry points of distributed computing infrastructures. FGNs assist IoT devices to get configured with integrated environment for placing and executing corresponding applications. Through FGNs, FogBus offers front-end program of applications so that users can set authentication credentials, access the back-end program, convey service expectations, receive service outcome, manage IoT-devices and request resources according to their affordability. In addition, FGNs operate data filtration and organize them in a general format. FGNs also aggregate the data received from different sources of a smart system. For large scale processing, FGNs forward the data to other computing instances of integrated environment. FGNs maintain rapid and dynamic communication with accessible Fog nodes through either Constrained Application Protocol (CoAP) or Simple Network Management Protocol (SNMP) (Slabicki and Grochla, 2016).

Fog Computational Nodes (FCN): FogBus can deal with many heterogeneous Fog Computational Nodes (FCNs) simultaneously. FCNs are equipped with processing cores, memory, storage and bandwidth to conduct various FogBus operations. Based on these operations, FCNs perform three different roles:

1. *Broker nodes*: To run the back-end program of IoT-applications, FogBus facilitates corresponding FGNs to connect with any of the accessible FCNs. This FCN initiates data processing provided that required resources for the application are available within it. If it fails to meet the application's requirements, as a broker node it communicates with other FCNs and Cloud datacenters to provision necessary resources for executing the back-end program. In this case, it distributes the computational tasks over computing instances, seamlessly monitors and coordinates their activities, and conducts load balancing. FogBus supports such broker nodes with adequate security features and fault tolerant techniques such as Blockchain and replication so that they can ensure reliability while interacting with FGNs, FCNs and Cloud datacenters.

2. *General Computing Nodes (GCNs)*: For security issues, FogBus does not expose all FCNs directly to FGNs. They are used for general computing purposes and accessible via broker nodes. In this case, broker nodes work as firewall for GCNs. Besides, broker nodes explicitly manage their resources and forwards the data along with executable back-end applications for processing. GCNs form clusters under the supervision of specific broker node while executing distributed applications. A GCN can relate to multiple broker nodes. In this case, basic Vector Clocks are used for synchronizing the system (Bravo et al., 2015). Vector Clocks help GCNs to identify the concurrent commands issued to them by different broker nodes. Later, the concurrent commands are arbitrarily ordered by GCNs and corresponding broker nodes are notified. While carrying out a command, GCNs interact with associated broker node in one-to-one manner. Furthermore, to ensure application-level consistency, a GCN executes at most one application at a time.

3. *Repository nodes*: Apart from conducting brokering and computing operations, some FCNs manage distributed database to facilitate data sharing, replication, recovery and secured storage. The repository nodes offer interfaces for instant access and analysis of historical data. They maintain meta-data of various applications including application model, runtime requirements and dependencies. Moreover, these nodes can preserve some intermediate data during application execution so that data processing can be started from any anomaly-driven stopping point. In addition, to ensure data-level consistency, repository nodes manage all data in a log structured manner which is driven by their updating timestamp and source.

Cloud datacenters: When Fog infrastructure becomes overloaded or service requirements are latency-tolerant, FogBus extends resources from Cloud datacenters to execute back-end IoT applications. Through Cloud datacenters, FogBus expands the computing platform for IoT applications across the globe. In association with Fog repository nodes, it facilitates extensive data storage and distribution so that access and processing of data become location independent.

3.2. Software components

To simplify IoT-Fog-Cloud integration, FogBus provides various interrelated software components that can deal with Operating System(OS) and peer to peer (P2P) communication-level heterogeneity of different hardware instruments. These components are broadly classified into three types of **System Services**. The Broker service manages the functionalities of a broker node and initiates other software components according to the necessity, whereas the Computing service is responsible for controlling the operations of a GCN. When a broker node itself starts the execution of back-end applications, the computing service is triggered within it. Conversely, Repository service can run across all the Fog nodes to manage repository-related operations. The interaction among different FogBus software components are presented in Fig. 2. The details of FogBus software components are discussed as follows.

3.2.1. Broker service

Broker Security Manager: After receiving user's authentication credentials from a FGN, the Broker Security Manager verifies them in association with Credential Archive of Repository Service.

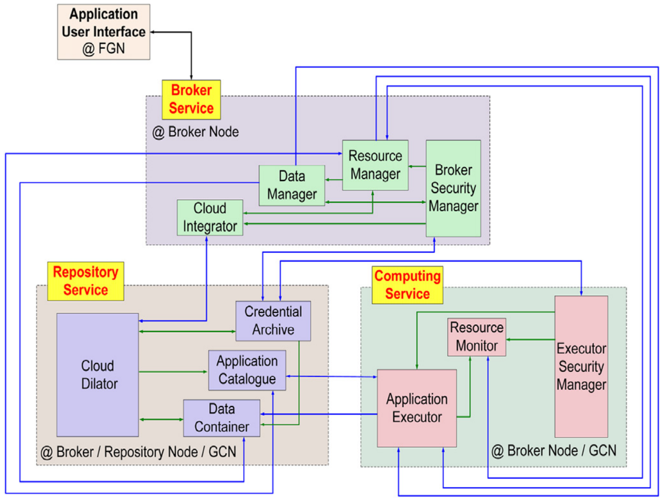


Fig. 2. Interaction of different software components within FogBus.

The Credential Archive also assist this component with required security certificates for remote Cloud integration. The Broker Security Manager generates the public and private key value pairs to facilitate port knocking, privileged port authentication and attribute-based encryption for securing the communication of corresponding broker node with other Fog nodes. Additionally, this component acts as the Blockchain interface for ensuring integrity while exchanging data with multiple entities. In this case, with the help of Data Manager it creates new blocks from the received data. The hash values and proof-of-work for each block are sent to Credential Archive for distributing among other nodes so that consistent verification of the chain can be ensured at different destinations. The Broker Security Manager along with Credential Archive and Executor Security Manager of Computing service manage further security issues within FogBus and offers other components flexible accesses to the required information.

Resource Manager: This component is responsible for selecting suitable resources to execute applications. It identifies the requirements of different applications from Application Catalogue of Repository service and perceives the resource status within each broker and GCN through Resource Monitor of Computing service. Cloud Integrator assists Resource Manager with contextual data of Cloud-based instances such as virtual machines and containers. After attaining all information, Resource Manager provisions required resources on FCNs and Cloud for applications. In this case, Resource Manager explicitly considers the heterogeneity of computing instances in terms of their processing capabilities. Application Executor from Computing service and internal software system of FCNs and Cloud respectively help the Resource Manager to provision resources. FogBus facilitates service providers to apply various policies with Resource Manager during resource provisioning. In addition, this component maintains a resource configuration file that tracks the addresses of FCNs and Cloud instances along with deployed applications so that subsequent data streams can directly be sent to the allocated resources for processing. This file is also shared with Cloud for recovering the placement information during failure of the corresponding nodes.

Data Manager: This component receives the sensed and preprocessed data from the IoT devices. It can also aggregate data from multiple sources and calibrate data receiving frequency according to the context. However, with this data, blocks and their chains are created for maintaining integrity in association with the Broker Security Manager. Later it forwards the data to Application Executor of Computing service for processing and stores them in encrypted manner on Data Container of Repository service for further use. After deployment of applications on allocated resources, Resource Manager shares the resource configuration file to Data Manager so that it can directly send subsequent data stream to the processing destination.

Cloud Integrator: All interactions of FogBus with Cloud are handled by Cloud Integrator. It notifies the context of Cloud instances to the framework and forwards the storage and resource provisioning commands to the Cloud. Through this component, FogBus not only offers interface to providers for developing customized Cloud-integration scripts but also allows third-party software systems such as Aneka Calheiros et al. (2012b) to deal with multiple Cloud datacenters simultaneously using their Application Programming Interfaces (API).

3.2.2. Repository service

Credential Archive: User's authentication credentials, that are set during IoT device configuration, are preserved in Credential archive. It distributes the security keys and details of each data block generated by the Broker Service to others. This component also provides the Secure Socket Layer (SSL) and Transport Layer Security (TLS) certificates for Cloud integration. In addition, it supports Data Container for encrypting and decrypting the stored data. Through Cloud Dilator of Repository service, it periodically updates its image on Cloud so that security attributes can be recovered and distributed among others easily after uncertain failure occurs.

Application Catalogue: This component is responsible for maintaining the details about various types of applications including their operations, recommended system properties from their developers, execution and programming model. Moreover, it specifies resource requirements and dependencies of the applications and their member tasks. The Application Catalogue can extend this information from Cloud through Cloud Dilator. Based on its provided specifications, Resource Manager of Broker service provisions resources for an application. According to the commands of Resource Manager, it also assembles applications on allocated resources in association with the Application Executor of Computing service.

Data Container: Data received from IoT devices are stored in Data container so that it can be used for long term analysis. Here, data privacy is ensured by applying encryption techniques. During application execution, it also receives some intermediate data from Application Executor that helps FogBus to restart the processing of data from any halting point. Moreover, in FogBus, the schema of Data Container based databases can be customized and shared according to the requirements of different IoT-enabled systems. In addition, Data Container maintains simultaneous association with Cloud Dilator to grasp the remote data and disperse the local data through Cloud.

Cloud Dilator: This component facilitates other software components of Repository service to interact with Cloud. In this case, the Cloud Integrator of Broker service assists Cloud dilator with required commands for extending application specifications, transferring security attributes and exchanging data.

3.2.3. Computing services

Executor Security Manager: While conducting computing operations, the seamless secured interactions of an FCN with others are managed by the Executor Security Manager. The Credential Archive of Repository service assist this component with required security attributes. Along with Credential Archive and Broker Security Manager, this component plays a significant role in verifying the Blockchains.

Resource Monitor: Both busy and idle status (for example: CPU usage, memory occupation, network utilization, power consumption, etc) of computing resources are monitored by this component in association with the Application Executor. Based on these perceived information, Resource Manager provisions resources for different applications. It also tracks performance of allocated resources in meeting QoS requirements of applications during runtime. Whenever the load on resources exceeds a threshold value defined by service providers or uncertain failure occurs, this component immediately notifies Resource Manager. To deal with such scenarios, Resource Manager can initiate some actions such as dynamic resource provisioning, application execution migration and intermediate data storage. However, in current version of FogBus, these actions are kept in abstract form so that FogBus users can customize their resource management policy according to the system's context. Nevertheless, this component conducts necessary operations to achieve synchronization on the system.

Application Executor: Based on the provisioning instructions issued by the Resource Manager, this component allocates resources for different applications on corresponding FCN. It also extends the application executables from Application Catalogue for deployment on allocated resources. Once the application deployment is conducted, it begins to receive data forwarded by Data Manager for processing. In addition, this component periodically informs the status of resources to the Resource Monitor. When any anomaly is detected or predicted, this component is asked by the Resource Manager to extract intermediate data from application execution and store them on Data Container to make the framework fault tolerant.

3.3. Network structure

The software components of FogBus share numerous data and information among themselves. To facilitate their interplay, persistent and stable network communication among hardware instruments of the framework is necessary. It is also required to ensure that hardware instruments do not become overwhelmed with the communication burden. In addition, the FogBus networking should be secured, scalable and fault tolerant. Taking cognizance of these facts, we design the FogBus network structure as shown in Fig. 3. Different aspects of the network structure are described as follows.

Topology: The master-worker topology is applied in designing the network structure for FogBus framework. **Here, broker nodes are the masters while other FCNs function as the workers**. Being master, a broker node receives the data stream and user information from the FGNs and discovers workers for processing and storing them. During application runtime it manages functionalities of the workers and delivers the service result to FGNs derived from application execution. In addition, it connects the Fog infrastructure of a FogBus enabled system with the Cloud infrastructure. To foster data sharing and reduce overhead from the masters, worker nodes also communicate among themselves under the explicit supervision of the masters. The masters, workers and FGNs of a FogBus-enabled system relate to a common wireless local area network (WLAN) that is managed by one or multiple routers.

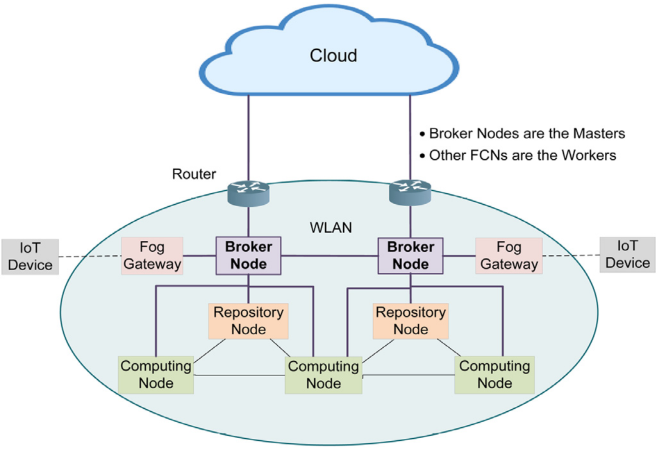


Fig. 3. Network Structure of FogBus framework.

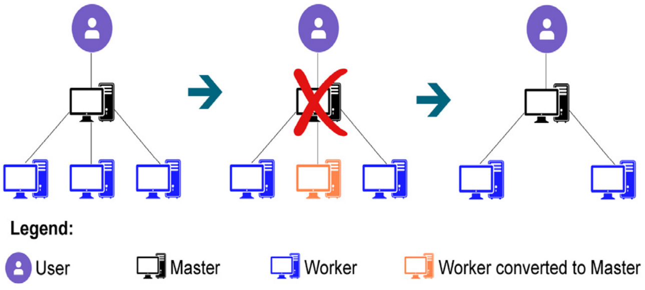


Fig. 4. Ensuring reliability in FogBus framework.

Scalability: FogBus framework allows service providers to scale-up the number of active Fog nodes according to context of the system. An FCN connected with the same WLAN can simply become a worker by making itself accessible to the corresponding master. Later, the master configures required software components on that FCN to conduct desired operations. The FogBus supports coexistence of multiple masters in a WLAN so that FGNs can get diverse options to dispatch the data streams for processing. The masters also share workers among themselves. In this case, the data integrity and privacy are not affected since each master maintains its own chain of blocks and separate database on the workers. In addition, the software components running at the masters facilitate Fog infrastructure to integrate with multiple Cloud datacenters simultaneously.

Reliability: The facility of running multiple masters implicitly eradicates the inherent single point failure limitation of master-worker model within FogBus. Vector Clock-based synchronization and one-to-one interaction during application execution and data management support explicit isolation and consistency of their individual operations on different worker nodes. The framework allows each master to replicate its image over one of the worker nodes. During uncertain failure of that master, this replication helps corresponding worker to get the master privileges and defend the collapse of communication network as shown in Fig. 4. Here, the platform-independent characteristic of FogBus software components plays the key role. The masters also periodically check status of its workers and store their intermediate data and configurations including deployed applications in different places. When a worker fails, the masters share the worker's information with other workers so that residual data processing can start immediately. If all the workers of a master are overloaded, workers of other masters are considered. In this case, all the masters maintain an internal communication among themselves. Thus, the computation facility remains always available within the framework.

Security: The inclusion of FGNs and FCNs in FogBus provided network require proper authentication. It is explicitly handled by the routers managing the WLAN. The masters also apply network level access control and packet filtration techniques to resist the network infrastructure from being compromised and eliminate the malicious contents. In FogBus, multiple communication links also exist to reach different Fog nodes. It eventually helps to readjust the routing path when any network anomaly is perceived. Cloud provided network security policies are further used in FogBus framework while interacting with different Cloud infrastructures.

Performance: FogBus framework utilizes the network bandwidth dedicatedly for a specific system. Since the network resources are not shared with external entities, the overall performance of the system from network perspective does not degrade.

If the service providers intent to increase the number of FCNs in the framework, requirements for additional network resources will not be very high as well. In addition, it facilitates easy deployment of the Fog nodes and faster service delivery to the users. As a localized network, its throughput also remains at an acceptable level with the course of time. Additionally, FogBus supports periodic adjustment of network resources so that it can deal with any frequency and volume of incoming traffic. Moreover, the network structure of FogBus does not depend much on external hardware instruments for managing and configuring the network operations that implicitly reduce the capital and the operational expenditure for the service provider.

4. Design and implementation

The APIs, execution environment, scripting and programming languages that are used in FogBus, can be supported by each hardware of the integrated environment. It eventually helps FogBus to function beyond their OS and P2P communication-level heterogeneity. The implementation of different FogBus elements are described as follows.

4.1. System services

Each System Service (Broker service, Repository service and Computing service) of FogBus is divided into **Service Interface** and **Management Activity**. At masters, the Service Interface assists in receiving data and user's specifications from the gateway devices and presents the service results. Service providers also notify the workers IP addresses to masters through this interface. The Management Activity within the master contains the resource provisioning and load balancing policies, and updates the configuration files. Additionally, it generates and forwards commands to the workers. The Service Interface at workers functions as a receptor of the corresponding node and is responsible to decode the output file of applications to masters. Based on the master's commands, the Management Activity at worker functions resource allocation, monitoring and status sharing. It also stores data in relational databases and creates input file for back-end program of applications. Apart from Blockchain, Service Interface and Management Activity of different System Services jointly handle other security aspects such as encryption and authentication for both masters and workers in FogBus.

In FogBus, Service Interface of each System Service is implemented as web programs. They are developed on PHP, an HTML-embedded server-side scripting language and use HTTP protocol based RESTful APIs to exchange data and share information among different FCNs within the WLAN. PHP based web programs can function in every operating system such as Unix, Windows, Linux and NetWare. On the other hand, HTTP is an application layer protocol that can be adapted to run with different transport layer protocols such as TCP and UDP. Besides, most of the embedded, networking and IoT devices are either designed with built-in protocol stacks for HTTP communication or support their easy installation. Thus, the Service Interface of a FogBus System Service can run across different types of OSs and P2P communication standards. In FogBus, an Apache server is also deployed in each FCN to run the web programs of corresponding System Service. Furthermore, in FogBus, the Management Activity of each System Service is developed in Java programming language. Compiled Java programs run on Java Virtual Machine (JVM), which can be installed easily across various OSs. Hence, the Management Activity of a FogBus System Service functions in wide range of platforms. In addition, MySQL servers are installed in different FCNs of FogBus to manage databases and their operations.

4.2. Blockchain

Maintaining integrity of data and ensuring that data is not sent by an unregistered source are very important for credibility of the system. For data integrity and data prevention from tampering, Blockchain technology is recently adopted in many real-time systems Zyskind et al. (2015). Theoretically, Blockchain is a set of distributed ledgers that can be programmed to record and track the value of anything. In Blockchain, whenever new data is received by an entity of the distributed system, it forms the data into a block. This block possesses a hash value that is usually created by using the corresponding data, index of the block in the chain, the timestamp of the data reception and the hash of its previous block within the chain. Additionally, the node mines the block with other blocks of the chain to create a proof-of-work for that block so that its hash follows a similar pattern with others. Later, the data, copy of the block is sent to other nodes for linking with their local chains. In this operation, nodes mine the block to certify the proof-of-work. Digital signature is also used to verify source of the block at the destination. However, if the data of any block is altered on a node, the hash of that block will change and mismatch with its hash saved in the next block. As a consequence, the later part of the chain will become invalid. To make the chain valid again, hash of the invalid blocks is required to be recalculated. Besides, the proof-of-work of each block requires to be generated again. Both operations are time consuming and compute intensive. Moreover, this fraudulent manipulation of data in a Blockchain will not be successful unless 50% of its distributed copies are individually reformed by following the same set of operations. Thus, it becomes very hard to alter any data in Blockchain within rigid time limit (Swan, 2015).

In FogBus, the masters create the blocks from received data and calculate the hash of each block based on the data, hash of the previous block, timestamp and a nonce value using SHA256 algorithm (Brownworth, 2017). Masters also create random public/private key pairs that help to generate unique signatures with the original data. Later, they share Blockchain details, digital signature attributes and the data in Base64 encoding format with workers. With the received public key of the masters, the workers can verify that the data is coming from a legitimate source. If any other key is used, that data is rejected. The public-private key pair in this case is kept dynamic per block to prevent the generation of private key using brute force techniques. Additionally, each block and its hash are verified at the workers by mining the nonce value that supports the proof-of-work. If any worker reports error in terms of Blockchain tampering or signature forgery, then the Blockchain in majority of the network is copied to that node. FogBus also offers users and providers to track the data/block flow through the Service Interface running at masters by displaying the latest hashes of the Blockchain copy at each worker. Thus, it helps users and providers to take necessary action on suspicious activity within the FogBus network. In FogBus, the Blockchain is developed in Java programming language and in different FCNs of FogBus, this utility directly interacts with the corresponding System Service.

4.3. Cloud plugin

In FogBus, the Service Interface running at master prompts the user to specify their intention regarding Cloud integration for data processing. If users wish to extend Cloud resources for computation, only then the Cloud Plugin of FogBus which is deployed on the master, becomes activated. For other operations such as storage and distribution, the Management Activity at masters directly communicates with the Cloud. However, FogBus offers flexibility to providers for using different customized or third-party Cloud Plugin services to integrate Cloud and Fog infrastructure for computing purposes. In the case of running a third-party Cloud Plugin service, the master is configured according to the requirements of that plugin. However, to develop customized plugin, it is preferable to use cross-platform programming languages. In the current version of FogBus, Aneka, a third-party software is used for Cloud integration to perform computational operations.

Aneka is a PaaS framework for facilitating the management of Cloud-based applications (Calheiros et al., 2012b). The Aneka framework functions in a service-oriented manner. It is equipped with a set of software components to configure, operate, and monitor an Aneka-Cloud environment. The Aneka-Cloud can be formed with heterogeneous instances from either public or private, or hybrid Cloud. Aneka offers the developers diverse APIs for provisioning and scheduling both physical and virtual resources in the Aneka-Cloud. Developers formulate the logic of applications using different programming models and set the runtime environments for their deployment and execution. Currently Aneka platform supports the Bag of tasks, Distributed threads, MapReduce and Parameter sweep model. In the Aneka-based Cloud plugin of FogBus, IP addresses of Cloud instances are specified by the providers. This plugin can initiate both task and thread model in Aneka-Cloud to conduct data processing on single and multiple Cloud instances respectively (Calheiros et al., 2012a).

According to the built-in resource provisioning policy of FogBus, at first Fog infrastructure is exploited to process data. If load on Fog infrastructure increases up to a threshold value, the application and its input data stream is referred to Cloud infrastructure. For the second case in FogBus, the Management Activity at a master stores the data in a Cloud input file. The Aneka-based Cloud plugin at the master parses this file in every 500 milliseconds of polling period and checks for the pending data for processing. If any pending data exists, it forms either a task or threads; encapsulating the data at Aneka-Cloud and launches to one or multiple Cloud instances. In this case, Blockchain is also applied to ensure data integrity.

4.4. Application

FogBus supports execution and deployment of applications of different IoT-enabled systems. In FogBus these applications are divided into front-end and back-end program. Although applications are not the part of FogBus software components, FogBus offers developers some guidelines to build their front-end and back-end programs aligned with the features of FogBus framework. The required specifications of front-end and back-end program of applications are described as follows.

4.4.1. Front-end program

The front-end program of applications runs in FGNs. The underlying platform of most of the FGNs are Android, iOS, Windows, Tizen, WebOS and RTOS. In this case, the programming language for developing the front-end program should be supported by these platforms. Moreover, for some applications, front-end program requires to store data temporarily. On that note, the developers should use compatible database system and schema to these platforms. Besides, the front-end program deals with the incoming data from IoT devices. The majority of IoT devices run Bluetooth Low Energy network technology for communication as they are energy constraint. To handle this issue, the front-end program should support both general and low energy Bluetooth interactions. In FogBus framework, front-end program is directly correlated with the Service Interface running at the masters for forwarding IoT data and user information, and receiving the service outcome. For simplicity of these interactions, the user interface of front-end program can be designed in such a way that easily parses the web programs of master's Service Interface.

4.4.2. Back-end program

In FogBus, the back-end program of applications is executed in the FCNs. Since the FCNs are distributed, to fully leverage their capabilities it is preferable to build the back-end program in distributed manner. In this case, modular development of back-end program can be applied by the developers. In addition, the execution of back-end program should not be obstructed by the OS-level heterogeneity of FCNs. To address this issue, developers can use cross platform programming languages such as Java to develop the back-end program. While developing the back-end program some specific points within the script should be specified so that application's intermediate data on those points can be stored during execution. Furthermore, the back-end program should be able to extract the input file and update the output file at the workers.

5. A case study: sleep apnea analysis

In this work, FogBus framework has been adopted for deploying and executing a real-world application named *Sleep Apnea Analysis*. Sleep Apnea is a disease in which air stops flowing into the lungs for 10 seconds or even longer period during sleep. Hence, it reduces oxygen level in blood of the patient, downs the heartbeat rate and resembles that the patient has stopped breathing. It can happen very frequently and create severe obstruction in sound-sleep of the patient. Furthermore, if oxygen saturation becomes significantly low for aged and asthma patients, Sleep Apnea could provoke cardiac failure or brain stroke. However, Sleep Apnea is a very common disease although most of the people either ignore or unaware of it. To determine the intensity of Sleep Apnea, it is required to monitor oxygen saturation rate in blood time to time. If the intensity becomes higher than normal, it is recommended to consult with the Doctor before it occurs other complications (He et al., 1988).

Usually, Sleep Apnea analysis is difficult and cumbersome since it requires an overnight sleep study to grasp the necessary data. In this procedure, pulse oximeter and Electrocardiogram (ECG) machines are hooked up with various parts of the patient's body during sleep time. Based on the received peripheral capillary oxygen saturation, SpO2 and ECG data, the doctors determine Apnea Hypopnea Index (AHI) of the patients that presents the Sleep Apnea intensity in proportional manner. Currently, to conduct the Sleep Apnea analysis, hospital or laboratory-based machineries are required which are expensive to own by individuals. Besides, this analysis becomes very latency sensitive while critical patients are being monitored. Therefore, we develop a prototype for low cost Sleep apnea analysis using FogBus framework that gathers both SpO2 and heart beat rate from a finger pulse oximeter (Nigro et al., 2011) and harness local resources for their processing. It is affordable for patients, easily configurable and provides faster results compared to Cloud-based processing. The detail of FogBus-enabled Sleep Apnea analysis prototype is described as follows.

5.1. System configuration

The system setup for FogBus-enabled Sleep Apnea analysis prototype is presented in Fig. 5. The configuration of different hardware instruments are given below.

IoT Device: Jumper JPD-500F Finger Pulse Oximeter, 1.5V, Bluetooth Low Energy v4.2 (BLE), UTF-8 data encoding.

Gateway: Smartphone, Oppo A73 CPH1725, Android 7.1.1.

Broker/Master Node: Dell Latitude D630 Laptop, *Intel(R) Core(TM)2 Duo CPU E6550 @ 2.33GHz 2GB DDR2 RAM*, 32-bit, Windows 7, Apache HTTP Server 2.4.34, Java SE Runtime Environment (JRE) 1.6, MySQL 5.6,.net 3.5, Aneka 3.1.

Other FCN/Worker Node: Raspberry Pi 3 B+, *ARM Cortex-A53 quad-core SoC CPU @ 1.4GHz 1GB LPDDR2 SDRAM*, IEEE 802.11,

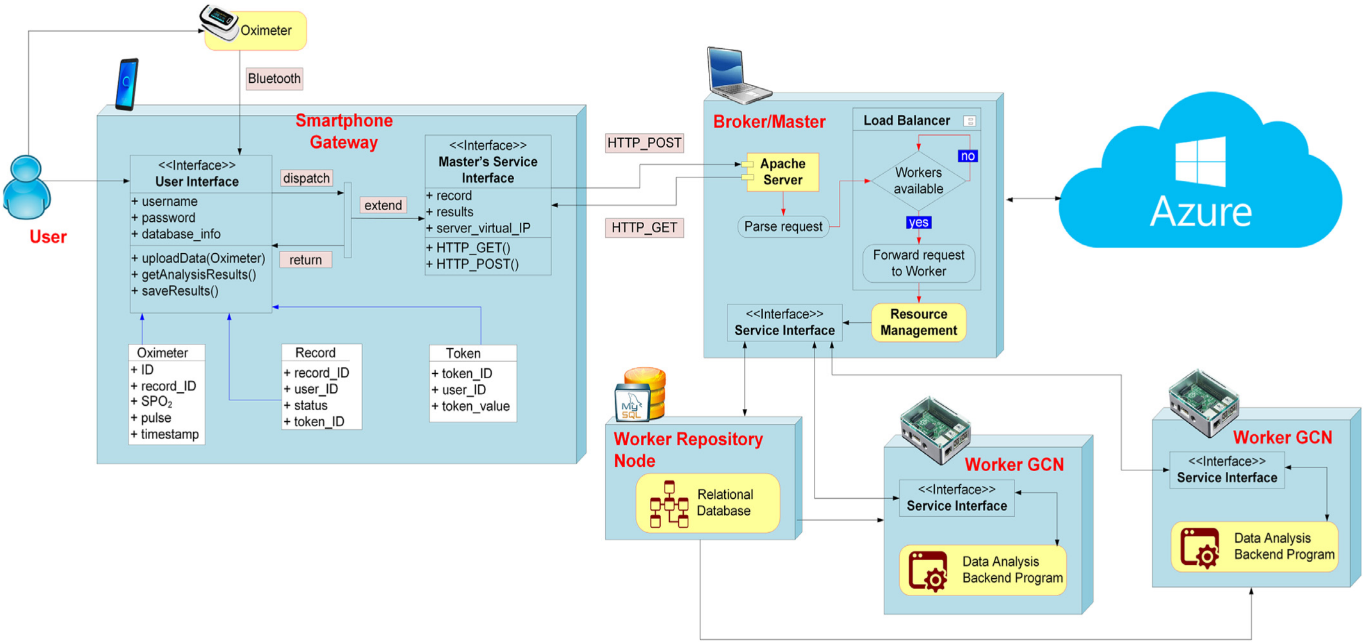


Fig. 5. FogBus framework enabled system model for Sleep Apnea analysis.

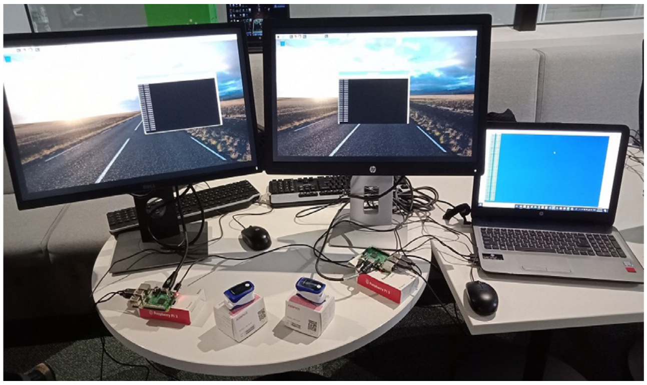
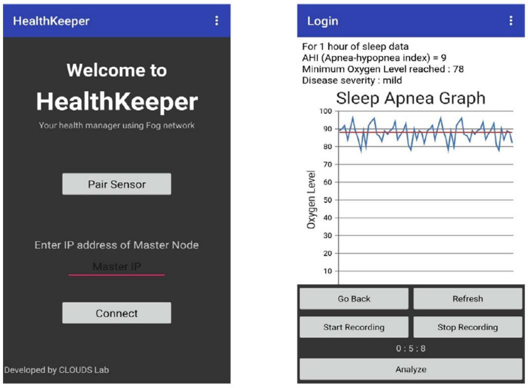


Fig. 6. Real-world implementation of FogBus-based Sleep Apnea analysis.



HealthKeeper

Welcome to

HealthKeeper

Your health manager using Fog network

Pair Sensor

Enter IP address of Master Node

Master IP

Connect

Developed by CLOUDS Lab

(a)

Login

For 1 hour of sleep data
AHI (Apnea-hypopnea index) = 9
Minimum Oxygen Level reached : 78
Disease severity : mild

Sleep Apnea Graph

Go Back | Refresh

Start Recording | Stop Recording

0 : 5 : 8

Analyze

(b)

Fig. 7. (a) Home and (b) Session Screen of the android interface.

64-bit, Raspbian Stretch, Apache HTTP Server 2.4.34, JRE 1.6, MySQL 5.6.

Public Cloud: Microsoft Azure B1s Machine, 1vCPU, 1GB RAM, 2GB SSD, Windows Server 2016,.NET 3.5, Aneka 3.1.

Fig. 6 depicts the real implementation of this system model.

5.2. Installed package

The developed prototype for Sleep Apnea analysis is mostly Fog infrastructure centric. However, if Fog infrastructure is unable to process the data, using built-in Aneka-based Cloud Plugin of FogBus, the data is sent to Azure VM. The application package for Sleep Apnea analysis installed in the prototype consists of an android front-end and a data analytic back-end program. Description of the installed package is given below.

5.2.1. Android interface at smart phone gateway

An android executable named *HealthKeeper* launches the android interface to the prototype operator. The executable installed on the Smartphone allows the device to act as an mediator between the Pulse Oximeter and the Master. It is developed on *MIT App Inventor*, an open source platform (Lab, 2015). The interface is divided into *Home* and *Session* screen (Fig. 7). The Home screen helps operator to pair the Oximeter with the Smartphone for receiving patient data using Bluetooth and enter the master's IP address. The Session screen handles all interaction with the master including data transmission. In this case, rather than sending data manually through the HTML form, the interface records and transmits data automatically. An empty data list is initialized and timer is reset when recording starts. Each data value received from the Oximeter is appended to the list. When the recording is stopped, the list is sent to the master for storage and distribution to the workers. This screen also extends the Service Interface running at the master and displays the result to operators once they become available to the master.

5.2.2. Data analytic at worker computing nodes

The data analytic for Sleep Apnea analysis encapsulates two open source programs found in Manigadde (2018) and Initiative (2017). These Java programs are stored in the repository worker and based on the command of master, they are forwarded to the computing workers for installation. The data analytic takes the input data as a file. From the input file the first, second and third columns are parsed as the timestamp, heart beat rate and blood oxygen level respectively. In the analytic, a Boolean variable tracks whether there is a dip in oxygen level or not. Whenever the oxygen level goes below 88, the dip Boolean variable turns to true and stays true till oxygen level is above 88. It is verified by the rise of heart beat rate in nearby timestamps of the dip occurrence in oxygen level. A counter variable in the analytic narrates how many times the dip Boolean variable has been changed to true. This count is known as the *Apnea - Hypopnea Index, AHI* that is used to determine the intensity of Sleep Apnea. AHI based cases for Sleep Apnea analysis are given below.

$$\text{Sleep Apnea} = \left\{\begin{array}{lr} \text{No/Minimal,} & \text{for } AHI < 5 \text{ per hour} \\ \text{Mild,} & \text{for } 5 \geq AHI < 15 \text{ per hour} \\ \text{Moderate,} & \text{for } 15 \geq AHI < 30 \text{ per hour} \\ \text{Severe,} & \text{for } AHI \geq 30 \text{ per hour} \end{array}\right\}.$$

However, as additional information, the data analytic stores the minimum oxygen level for the given period. For the heart rate data, minimum and maximum value are identified. The average heart rate and average rise or fall of the heart rates are also determined. In addition, heart beat pattern during the dips in oxygen level are filtered and ECG is generated. After identifying these information and Sleep Apnea intensity, the analytic delivers the result in a file. This file is later parsed by the master's Service Interface to notify the prototype operator.

5.3. Sequence of communication

In the prototype of FogBus-enabled Sleep Apnea analysis, all hardware instruments belongs to same WLAN. Their sequence of communication is presented in Fig. 8. This sequence initiates by configuring the Pulse Oximeter with the Smartphone using required credentials of the operator. The Oximeter senses patient's SpO2 and heart beat rate and forwards to the Smartphone through bluetooth communication. From Smartphone, these data are sent to the master. The master later stores the data on repository worker. After the storage operation acknowledgement is confirmed from the repository worker to the master. Since the Smartphone extends master's Service Interface, this acknowledgement becomes visible to the operator. After recording the data for a certain period, the operator prompts a request to the master via Smartphone for analyzing the stored data. Then, the master communicates with a suitable computing worker and issues required privileges for data analysis to it. The computing worker requests the stored data and analytic executable from the repository worker. On reception of these elements, the computing worker starts the analysis operation. Once the analysis operation is finished, the result is sent back to the master. The Smartphone pulls the result from the master and displays to the operator.

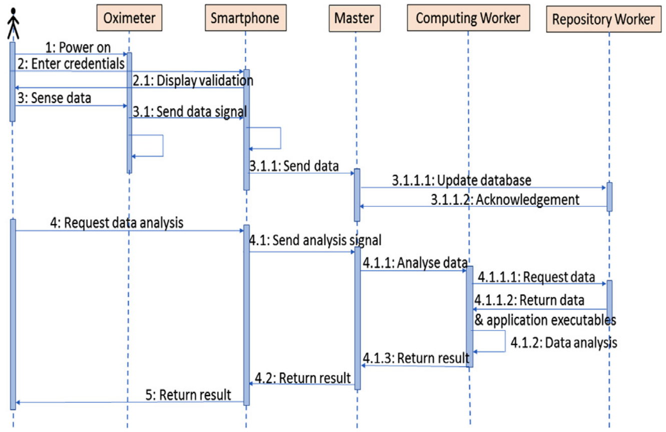


Fig. 8. Sequence of communication during Sleep Apnea analysis.

6. Framework characteristics evaluation

6.1. Experimental setup

To evaluate the framework characteristics of FogBus, we implement different features of Stack4Things (Bruneo et al., 2016), Cloudlet-based PaaS (Yi et al., 2015a) and Indie Fog (Chang et al., 2017) framework by following the given guidelines and compare their performance with FogBus. During experiments, the Blockchain feature of FogBus is kept enabled and the framework manages an integrated Fog-Cloud infrastructure. In experiments, data processing requests are launched sequentially to the frameworks with no interval. Here, the following framework characteristics are evaluated.

1. *Load on resources*: Computing processor and memory usage of a framework can indicate towards its load on resources. Since most of the Fog nodes are not abundant in resources, execution of heavyweight software systems can cause significant computing overhead on them. Therefore, it is required to deploy lightweight frameworks in Fog computing environments. The framework that consumes less computing processor and memory is considered lighter than the other frameworks.

2. *QoS expectation miss rate*: Due to management overhead and limitations of resources, frameworks often failed to meet QoS expectations of users. The lower rate of QoS expectation miss rate reflects higher efficiency of the framework in managing and harnessing its computing resources.

3. *Time-based attributes*: System initiation time and data retrieving time in vital for indicating performance of a framework. Required time to interconnect and start all components within a framework is defined by the system initiation time. Data retrieving time refers how much time is required to access data through a framework. The lower value of these time-based attributes signifies higher responsiveness of the framework.

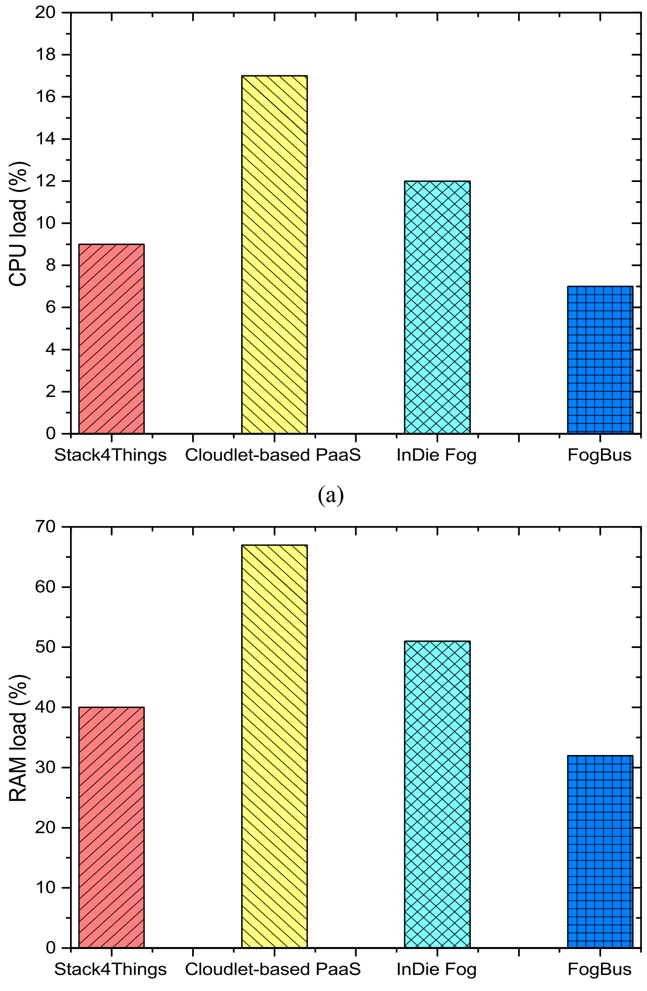
6.2. Result analysis

6.2.1. Lightness of frameworks

Apart from FogBus, the other frameworks incorporate third-party software systems extensively. As a result, their load on computing (CPU) and memory (RAM) resources of core/master/broker Fog nodes are higher than FogBus (Fig. 9). Moreover, in Cloudlet-based PaaS framework, system operations are not distributed across multiple Fog nodes that increases the CPU and RAM utilization of core Fog node. Conversely, in Indie Fog and Stack4Things, despite of operating Fog nodes collaboratively, the *Indie Fog registry* and *virtual board* running in core Fog node are given additional responsibilities like federated resource, repository and security management, service discovery, application service management etc. In consequence, the CPU and RAM load on the core Fog nodes of Indie Fog and Stack4Things framework increase.

6.2.2. Performance in managing resources

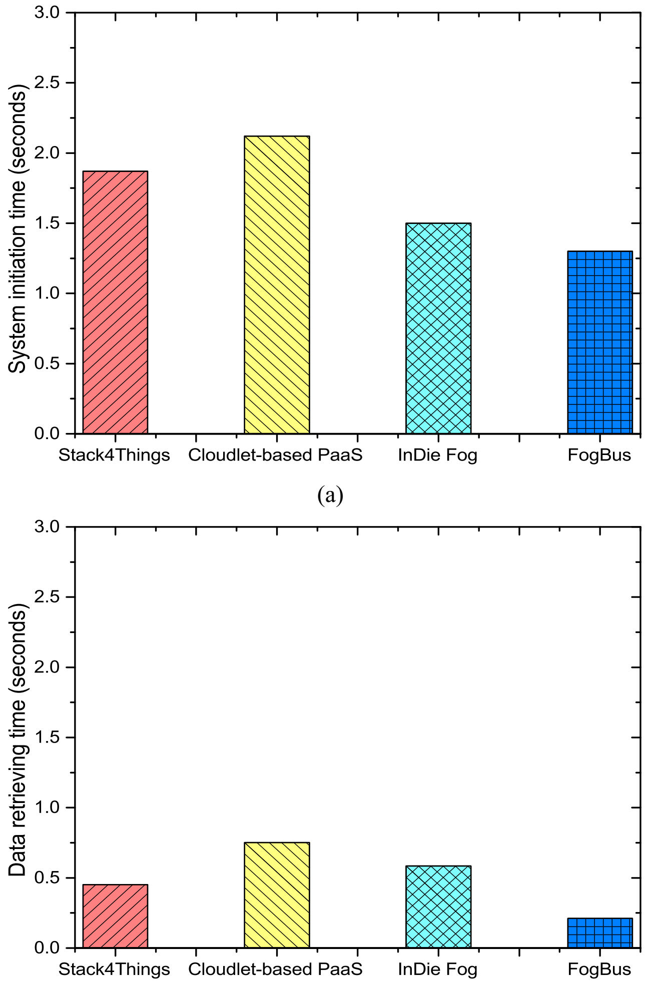
FogBus framework is capable of harnessing both edge and remote resources simultaneously. When Fog nodes become overloaded, FogBus continues input processing through Cloud datacenter-based resources. Hence, it provides a larger scope to meet the QoS expectation of users in respect of processing their data. As a consequence, the QoS expectation miss rate in FogBus becomes comparatively lower than InDie Fog, where application execution are solely managed by Fog node-based resources (Fig. 10). In Cloudlet-based PaaS framework, excessive load on CPU and RAM force the core Fog node to slow down management operations for input data streams that increases its QoS expectation miss rate compared to the lightweight FogBus. Furthermore, due



(a)

(b)

Fig. 9. (a) CPU and (b) RAM usage of core node in different frameworks.



(a)

(b)

Fig. 11. (a) Initiation and (b) Data retrieval time of different frameworks.

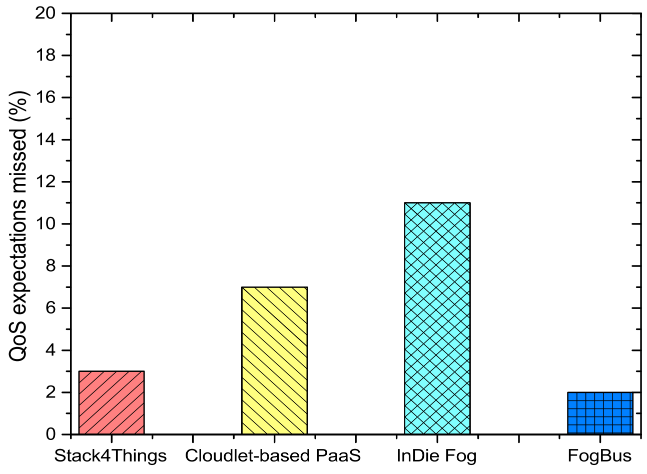


Fig. 10. QoS expectaion miss rate in different frameworks.

to managing the application service through a core Fog node deployed in Cloudlet, the QoS expectation miss rate on Stack4Things is slightly higher than FogBus.

6.2.3. *Responsiveness of frameworks*

Compared to other frameworks, the system initiation time of FogBus is shorter (Fig. 11). Less number of third-party software dependencies, on-demand Cloud interaction, and more horizontal-level connections than vertical-level help FogBus in this case. Data retrieval in FogBus also takes less time than others. FogBus stores data in local repository nodes in distributed manner rather than sending them towards Cloud or any centralized data storages. Consequently, data can be easily accessed through FogBus during application execution.

7. Application deployment evaluation

FogBus supports implementation of various resource management and scheduling policies for executing IoT applications composed using concurrent programming models such as SPMD (Single Program and Multiple Data), workflow and stream. As an example illustrator, we selected a health care application developed using SPMD model for evaluating application deployment scenarios in FogBus.

7.1. *Experimental setup*

The prototype for Sleep Apnea analysis, discussed in Section 5, is used to evaluate the impact of Blockchain and management overhead of FogBus on various system parameters such as service latency, energy consumption and network usage, while executing the application. For experiments, data from multiple oximeters are recorded for a specific period, later the master sequentially generates processing request for each recorded data chunk to computing workers. Each experiment scenario in FogBus is modelled under the following settings.

1. *With / Without Interval*: In With Interval setting, master sends the next data processing request to its computing worker after a certain interval of receiving the outcome for previous request. This time interval helps both master and computing worker to reduce their overhead. On the contrary, in Without Interval case, master sends the next request to its computing worker as soon as the outcome of the previous request becomes available. It ensures that the FogBus framework remains consistently active and there exists no idle time on the nodes.

2. *With / Without Blockchain*: FogBus offers flexibility to either enable or disable its Blockchain security feature according to the requirements of the users and service providers. The segments of this experiment setting differ from each other based on the status of Blockchain feature in FogBus.

3. *Fog / Cloud Only / Integrated*: FogBus supports application execution across diverse computing infrastructures. This experiment setting refers whether the application execution is solely conducted on Fog or Cloud, or integrated infrastructure. During the experiments, data parameters are recorded using Microsoft Performance Monitor at the Master and the Azure VM whereas at the Raspberry Pi circuits NMON Performance Monitor is used (Microsoft, 2017; Splunkbase, 2018). Apart from the system model parameters specified in Section 5.1, additional parameters used for the experiments are given in Table 2.

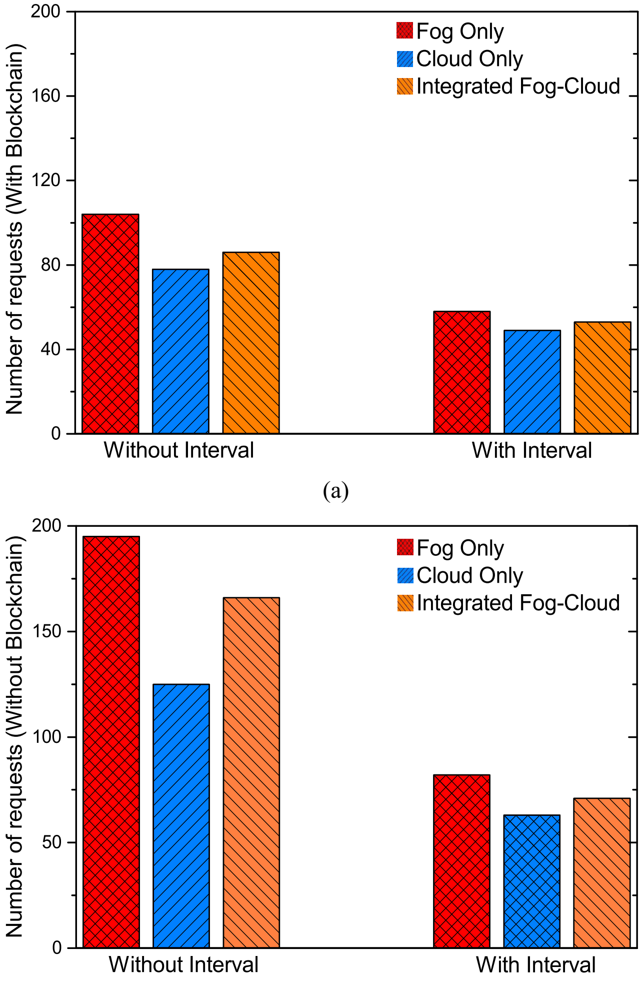
Table 2
Experiment parameters.

Parameter	Value
Analysis Task:	
Interval between creating sequential data processing requests	5 seconds
Data recording time per processing requests	3 minute
Pulse Oximeter:	
Signal length	18 KB
Sensing frequency	2 signal per second
WLAN:	
Download Speed	7 Mbps
Upload Speed	2 Mbps

7.2. Result analysis

7.2.1. Number of requests

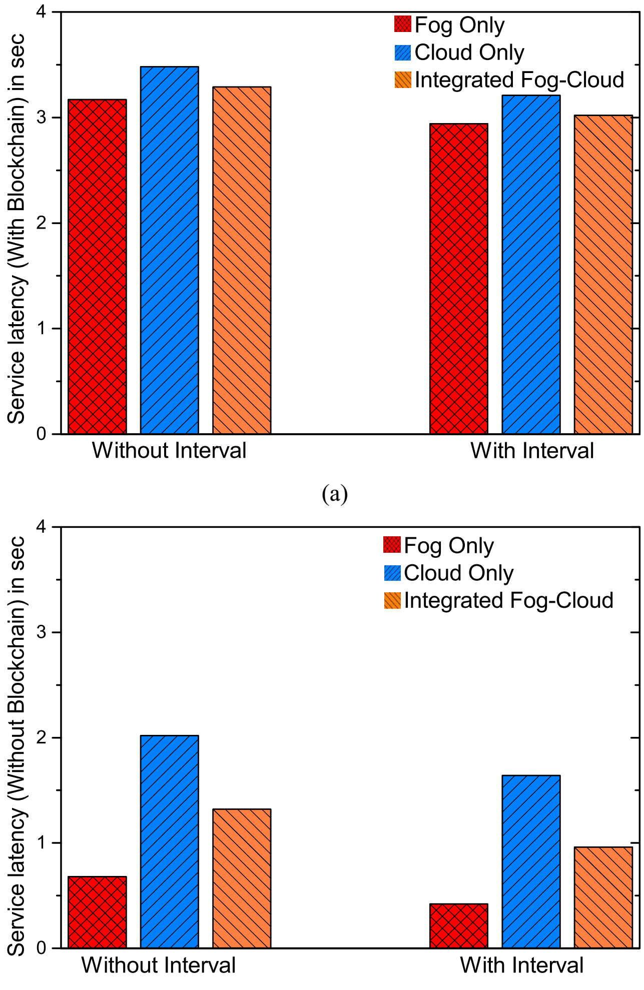
Fig. 12 depicts the number of requests generated in FogBus on different experiment settings. It is observed that the number of requests is higher in the Fog Only setting compared to the Cloud Only and Integrated Fog-Cloud case. It happens since Fog infrastructure quickly delivers outcome of the previous request. During Without Interval setting, this value rises significantly than the With Interval setting since requests are generated continuously by



(a)

(b)

Fig. 12. Number of requests (a) With and (b) Without Blockchain.



(a)

(b)

Fig. 13. Service latency (a) With and (b) Without Blockchain.

the master. It is also noticed that, if Blockchain feature of FogBus is turned off, comparatively higher number of requests are generated. In this case, as lower amount of additional data is shared and processed over the infrastructure, it consequently improves the speed of receiving outcome for the previous request. Based on these observations, it is understood that if there exists higher number of requests to be handled with less security requirements, FogBus can be set to Fog Only setting with disabled Blockchain feature. However, in such state the management and processing overhead of the infrastructures will increase in proportion to the number of requests and the size of data chunk for individual request. It can be managed by tuning the interval between request creation.

7.2.2. Service latency

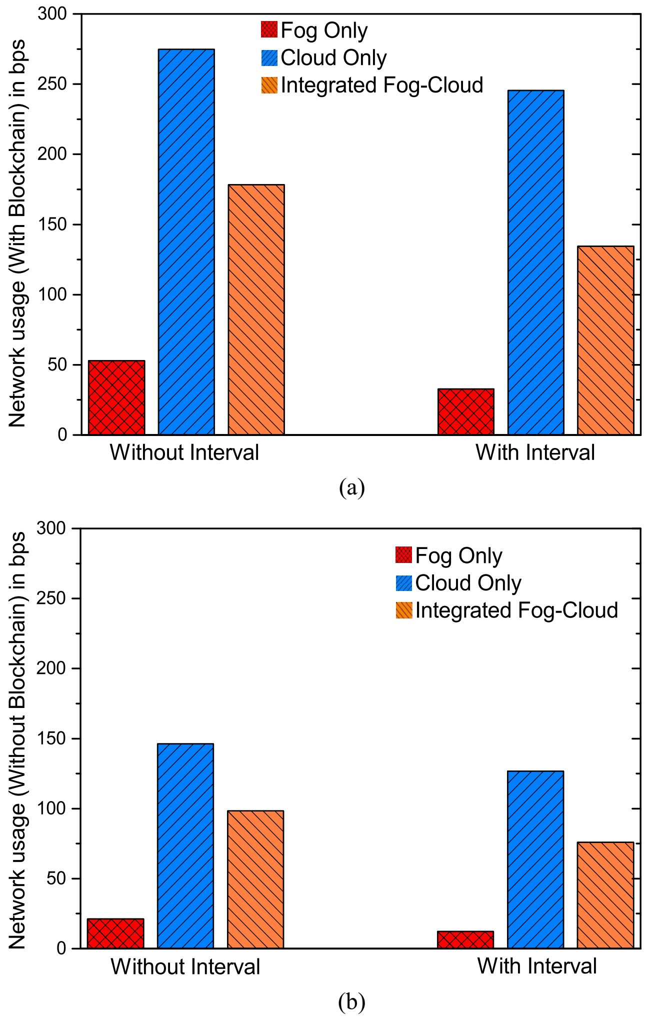
Fig. 13 presents the impact of different settings of FogBus on service latency. Here service latency is modelled as the summation of network propagation delay and task completion or application execution time. It is known that computational capability of Fog infrastructure is not enriched but it resides closer to the data source. As a consequence, network propagation delay is quite less for Fog infrastructure. Furthermore, if the size of data chunk for a request is not huge, its completion time will not differ significantly whether the application is executed in Fog or Cloud. Since, in this experiment, size of data chunk for a request is not huge, the service delivery latency much depends on the network propagation delay. As a result, in Fog Only setting of the FogBus, service delivery latency is minimal compared to Cloud Only and Integrated Fog-Cloud case. This latency becomes much lower on disabled state of Blockchain feature since its management add some more time to complete the processing requests. Moreover, the With Interval setting reduces overhead from the infrastructure and network in this case; that also contributes to improve the service delivery latency. Therefore, it can be realized that these settings assist FogBus to deal with the requests having stringent deadline.

7.2.3. Network usage

Network usage in different settings of FogBus are presented in Fig. 14. In this experiment, Fog Only setting provides improved performance than Cloud Only and Integrated Fog-Cloud case, since it solely utilizes the local networking resources. The disabled Blockchain features also reduces the network usage as less amount of security attributes are required to be transferred across the infrastructures. However, network usage gets elevated when continuously requests are generated and their associated data and information are exchanged. In this case, tuning of subsequent request generation interval can reduce the network usage to a certain scale. Thus, these adjustments make FogBus operational even when less amount of network resources are allocated for an IoT-enabled system.

7.2.4. Energy

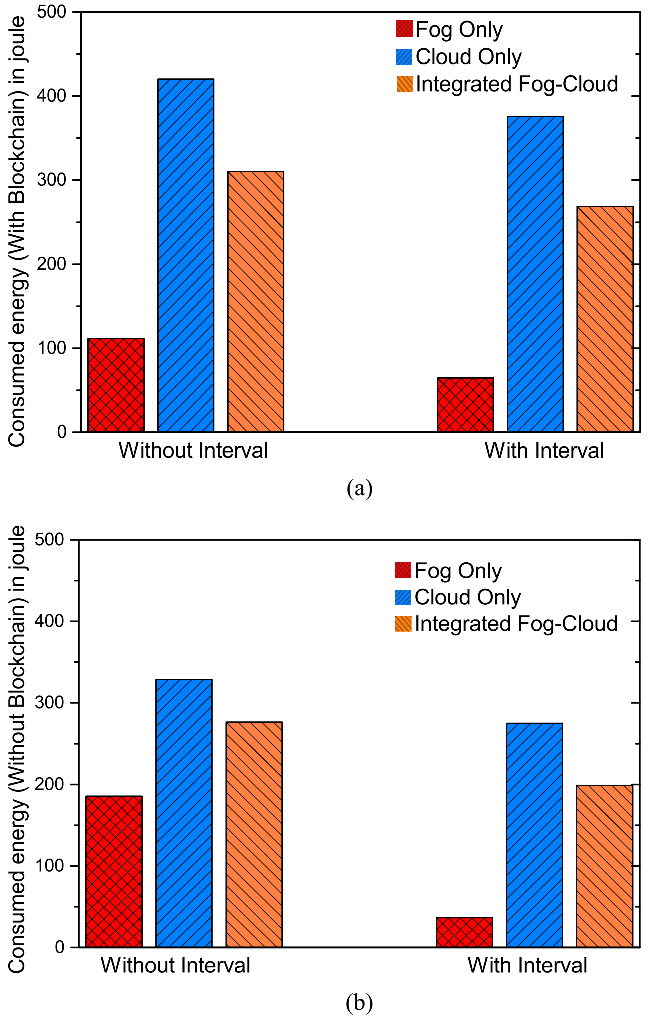
Fig. 15 presents how different settings of FogBus influence energy consumption of the infrastructure. In Cloud Only setting the Fog nodes are used for networking and Cloud VMs conduct the



(a)

(b)

Fig. 14. Network usage (a) With and (b) Without Blockchain.



(a)

(b)

Fig. 15. Energy consumption (a) With and (b) Without Blockchain.

computation whereas in Fog Only setting both the networking and computation are handled by Fog nodes. In Integrated Fog-Cloud case computational tasks are distributed to both the infrastructures according to the context of the system. Since, Cloud VMs consume much more energy compared to the Fog nodes, in Fog Only setting less energy is required to conduct the operations. Besides, to manage the Blockchain feature of the FogBus, additional energy is devoured. In this case, disabled Blockchain feature saves some energy for FogBus. In addition, energy consumption of an infrastructure during busy time is higher compared to its idle time. Therefore, interval between subsequent request creation assists to improve the energy usage of the infrastructure. However, it leads FogBus to process a smaller number of requests which can be overcome by efficient tuning of the interval time. These configurations help FogBus to execute applications under energy constraints.

8. Conclusion and future work

In this work, we propose FogBus framework that can integrate different IoT-enabled systems to both Fog and Cloud infrastructures. The framework facilitates IoT application deployment, resource monitoring and management. System Services of FogBus are developed in cross-platform programming languages (PHP and Java) and are used with extensible application layer protocol (HTTP) that help FogBus to overcome the OS and P2P communication-level heterogeneity of different Fog nodes. Additionally, the FogBus framework functions as a Platform-as-a-Service (PaaS) model for integrated Fog Cloud environment that not only assists application developers to build different types of IoT applications but also supports users to customize the services, and service providers to manage the resources according to the context of the system. Since some IoT-enabled systems such as health monitoring and utility service metering deal with sensitive data, FogBus applies authentication for data privacy and Blockchain for data integrity. To procure data transfer across less secure network, encryption techniques are applied in FogBus. Based on the principles of FogBus, a cost-efficient prototype for Sleep Apnea analysis is also developed. Moreover, comparing with existing frameworks, it is realized that the software components of FogBus are lightweight, adequately responsive and capable of harnessing both edge and remote resources. Besides, different FogBUs settings can efficiently deal with diversified situations while executing an application. Although FogBus is able to enhance service quality across diverse infrastructures, it can be still improved in a larger scope under the following aspects.

Resource management policies: FogBus provides flexibility to apply customized provisioning polices while allocating resources for different applications. Dynamic resource management policies on top of existing static management policy can be developed targeting load balancing among the computing infrastructures and the QoS enhancement.

Fog infrastructure virtualization: FogBus assists integration of Fog and Cloud computing with IoT-enabled systems. Although Cloud computing can be virtualized, in depth exploration is required to virtualize the Fog infrastructure in FogBus.

Artificial Intelligence: Currently FogBus does not support any artificial intelligence techniques for controlling the operations in different infrastructure and improving the resilience of the system. Inclusion of Artificial Intelligence techniques can be a significant contribution towards FogBus.

Application placement techniques: FogBus inherently supports distributed application execution. While placing applications in distributed manner, service latency, user expectations and deployment cost become predominant. In this case, different efficient application placement techniques can be added to the software stack of FogBus. Besides, some comparative performance studies on FogBus in dealing with different compute-intensive, network-intensive and storage-intensive applications can be conducted in future.

Runtime application migration: Migration of applications during runtime is very crucial if any anomaly is predicted. Different runtime application migration strategies for FogBus can be developed to handle such uncertain events.

Lightweight security features: Existing security features of FogBus require comparatively higher computational assistance. This consequently affects the service delivery latency, energy and network usage. Therefore, lightweight but effective security features can be helpful for further uplift of FogBus.

Improvement of embedded Blcokchain feature: The embedded Blockchain feature of FogBus is generic. However, it offers abstraction so that it can play a vital role on future research of Blockchain in reducing data retrieval time, managing smart contracts and implementing chain of chains over integrated Fog-Cloud environment.

Software Availability

We released FogBus as open source software. Its source codes along with users and developers manuals can be accessed from https://github.com/Cloudslab/FogBus.

Acknowledgements

This work is partially supported by Melbourne-Chindia Cloud Computing (MC3) Research Network and Australia Research Council (ARC) Discovery Project. We thank Editor-in-Chief (Prof. Paris Avgeriou and Prof.David C. Shepherd), Area Editor (Prof. Juan Carlos Duenas), and anonymous reviewers for their excellent suggestions on improving the paper. We also thank Jungmin Jay Son and Shashikant Ilager for their comments on enhancing the presentation.

References

Afrin, M., Mahmud, M.R., Razzaque, M.A., 2015. Real time detection of speed breakers and warning system for on-road drivers. In: Proc. of the IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE), pp. 495–498.

Akrivopoulos, O., Chatzigiannakis, I., Tselios, C., Antoniou, A., 2017. On the deployment of healthcare applications over fog computing infrastructure. In: Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual, 2. IEEE, pp. 288–293.

Azimi, I., Anzanpour, A., Rahmani, A.M., Pahikkala, T., Levorato, M., Liljeberg, P., Dutt, N., 2017. Hich: hierarchical fog-assisted computing architecture for healthcare IoT. ACM Trans. Embed. Comput. Syst.(TECS) 16 (5s), 174.

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the internet of things. In: Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing. ACM, pp. 13–16.

Bravo, M., Diegues, N., Zeng, J., Romano, P., Rodrigues, L.E., 2015. On the use of clocks to enforce consistency in the cloud. IEEE Data Eng. Bull. 38 (1), 18–31.

Brownworth, A., 2017. How Blockchain Works. http://blockchain.mit.edu/how-blockchain-works. [Online; accessed 28-August-2018].

Bruneo, D., Distefano, S., Longo, F., Merlino, G., Puliafito, A., D'Amico, V., Sapienza, M., Torrisi, G., 2016. Stack4Things as a fog computing platform for Smart City applications. In: Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on. IEEE, pp. 848–853.

Calheiros, R.N., Toosi, A.N., Vecchiola, C., Buyya, R., 2012. A coordinator for scaling elastic applications across multiple clouds. Future Gener. Comput. Syst. 28 (8), 1350–1362.

Calheiros, R.N., Vecchiola, C., Karunamoorthy, D., Buyya, R., 2012. The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid clouds. Future Gener. Comput. Syst. 28 (6), 861–870.

Chang, C., Srirama, S.N., Buyya, R., 2017. Indie fog: an efficient fog-computing infrastructure for the internet of things. Computer 50 (9), 92–98.

Chang, C., Srirama, S.N., Buyya, R., 2019. Internet of things (IoT) and new computing paradigms. Fog Edge Comput. 1–23.

Chen, N., Chen, Y., Ye, X., Ling, H., Song, S., Huang, C.-T., 2017. Smart city surveillance in fog computing. In: Advances in Mobile Cloud Computing and Big Data in the 5G Era. Springer, pp. 203–226.

Craciunescu, R., Mihovska, A., Mihaylov, M., Kyriazakos, S., Prasad, R., Halunga, S., 2015. Implementation of Fog computing for reliable E-health applications. In: Signals, Systems and Computers, 2015 49th Asilomar Conference on. IEEE, pp. 459–463.

Dubey, H., Monteiro, A., Constant, N., Abtahi, M., Borthakur, D., Mahler, L., Sun, Y., Yang, Q., Akbar, U., Mankodiya, K., 2017. Fog computing in medical internet-of-things: architecture, implementation, and applications. In: Handbook of Large-Scale Distributed Computing in Smart Healthcare. Springer, pp. 281–321.

Gia, T.N., Jiang, M., Sarker, V.K., Rahmani, A.M., Westerlund, T., Liljeberg, P., Tenhunen, H., 2017. Low-cost fog-assisted health-care IoT system with energy-efficient sensor nodes. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International. IEEE, pp. 1765–1770.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R., 2017. iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Software: Practice and Experience 47 (9), 1275–1296.

He, J., Kryger, M.H., Zorick, F.J., Conway, W., Roth, T., 1988. Mortality and apnea index in obstructive sleep apnea: experience in 385 male patients. Chest 94 (1), 9–14.

Hu, P., Ning, H., Qiu, T., Zhang, Y., Luo, X., 2017. Fog computing based face identification and resolution scheme in internet of things. IEEE Trans. Ind. Inf. 13 (4), 1910–1920.

Initiative, M., 2017. Sleep apnea clustering. https://github.com/monarch-initiative/sleep-apnea-clustering. [Online; accessed 28-August-2018].

Jayavardhana Gubbi and Rajkumar Buyya and Slaven Marusic and Marimuthu Palaniswami, 2013. Internet of things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29 (7), 1645–1660.

Lab, M. M., 2015. App inventor. http://appinventor.mit.edu/appinventor-sources/. [Online; accessed 28-August-2018].

Mahmud, R., Koch, F.L., Buyya, R., 2018a. Cloud-fog interoperability in IoT-enabled healthcare solutions. In: Proceedings of the 19th International Conference on Distributed Computing and Networking. ACM, New York, NY, USA, pp. 32:1–32:10.

Mahmud, R., Ramamohanarao, K., Buyya, R., 2018b. Latency-aware application module management for fog computing environments. ACM Trans. Internet Technol. (TOIT) in press https://www.sciencedirect.com/science/article/pii/S0743731518301771.

Mahmud, R., Srirama, S.N., Ramamohanarao, K., Buyya, R., 2018. Quality of experience (qoe)-aware placement of applications in fog computing environments. J. Parallel Distrib. Comput. doi:10.1016/j.jpdc.2018.03.004.

Manigadde, S., 2018. Sleep Apnea. https://github.com/subrahmanyamanigadde/sleepapnea. [Online; accessed 28-August-2018].

McKinsey & Company, May, 2018. The Internet of Things: How to capture the value of IoT. https://www.mckinsey.com/featured-insights/internet-of-things/our-insights/the-internet-of-things-how-to-capture-the-value-of-iot

Microsoft, 2017. Windows performance toolkit. https://docs.microsoft.com/en-us/windows-hardware/test/wpt/.[Online; accessed 28-August-2018].

Mohamed, N., Al-Jaroodi, J., Lazarova-Molnar, S., Jawhar, I., Mahmoud, S., 2017. A service-oriented middleware for cloud of things and fog computing supporting smart city applications. 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE.

Muhammad, G., Rahman, S.M.M., Alelaiwi, A., Alamri, A., 2017. Smart health solution integrating IoT and cloud: a case study of voice pathology monitoring. IEEE Commun. Mag. 55 (1), 69–73.

Nigro, C.A., Dibur, E., Rhodius, E., 2011. Pulse oximetry for the detection of obstructive sleep apnea syndrome: can the memory capacity of oxygen saturation influence their diagnostic accuracy? Sleep Disorders, Hindawi 2011.

Rahmani, A.M., Gia, T.N., Negash, B., Anzanpour, A., Azimi, I., Jiang, M., Liljeberg, P., 2018. Exploiting smart e-Health gateways at the edge of healthcare internet-of-Things: a fog computing approach. Future Gener. Comput. Syst. 78, 641–658.

Slabicki, M., Grochla, K., 2016. Performance evaluation of CoAP, SNMP and netconf protocols in fog computing architecture. In: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, pp. 1315–1319.

Splunkbase, 2018. Nigel's performance monitor. https://splunkbase.splunk.com/app/1753/. [Online; accessed 28-August-2018].

Swan, M., 2015. Blockchain: Blueprint for a New Economy. O'Reilly Media, Inc, United States.

Vatanparvar, K., Faruque, A., Abdullah, M., 2015. Energy management as a service over fog computing platform. In: Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems. ACM, pp. 248–249.

Verba, N., Chao, K.-M., James, A., Goldsmith, D., Fei, X., Stan, S.-D., 2017. Platform as a service gateway for the fog of things. Adv. Eng. Inf. 33, 243–257.

Yangui, S., Ravindran, P., Bibani, O., Glitho, R.H., Hadj-Alouane, N.B., Morrow, M.J., Polakos, P.A., 2016. A platform as-a-service for hybrid cloud/fog environments. In: Local and Metropolitan Area Networks (LANMAN), 2016 IEEE International Symposium on. IEEE, pp. 1–7.

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

Yi, S., Qin, Z., Li, Q., 2015. Security and privacy issues of fog computing: asurvey. In: International conference on wireless algorithms, systems, and applications. Springer, pp. 685–695.

Zyskind, G., Nathan, O., et al., 2015. Decentralizing privacy: using Blockchain to protect personal data. In: Security and Privacy Workshops (SPW), 2015 IEEE. IEEE, pp. 180–184.

Shreshth Tuli is an undergraduate student at the Department of Computer Science and Engineering at Indian Institute of Technology - Delhi, India. He is a national level Kishore Vaigyanic Protsahan Yojana (KVPY) scholarship holder for excellence in science and innovation. He has worked as a visiting research student at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. Most of his projects are focused on developing technologies for future requiring sophisticated hardware-software integration. His research interests include Internet of Things (IoT), Fog Computing, Network Design, and Artificial Intelligence.

Redowan Mahmud received the BSc degree in 2015 from Department of Computer Science and Engineering, University of Dhaka, Bangladesh. He is working towards the PhD degree at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. He was awarded Melbourne International Research Scholarship (MIRS) and Melbourne International Fee Remission Scholarship (MIFRS) supporting his studies. His research interests include Internet of Things (IoT), Fog Computing & Mobile Cloud Computing.

Shikhar Tuli is an undergraduate student at the Department of Electrical Engineering at Indian Institute of Technology - Delhi, India. He has worked remotely with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia in the realization of the FogBus framework. His research interests include Embedded systems, Internet of Things (IoT), and VLSI technology.

Rajkumar Buyya is professor and future fellow of the Australian Research Council, and the director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored more than 425 publications and four text books including "Mastering Cloud Computing" published by McGraw Hill and Elsevier/Morgan Kaufmann, 2013 for Indian and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide. Microsoft Academic Search Index ranked him as the world's top author in distributed and parallel computing between 2007 and 2012. Software technologies for grid and cloud computing developed under his leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. He has led the establishment and development of key community activities, including serving as foundation chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. These contributions and international research leadership are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE computer society. Manjrasoft's Aneka Cloud technology developed under his leadership has received "2010 Frost & Sullivan New Product Innovation Award" and "2011 Telstra Innovation Challenge, People's Choice Award". He served as founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established over 45 years ago.