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

Shreshth Tuli[1,2], Redowan Mahmud[2], Shikhar Tuli[3], Rajkumar Buyya[2]

Abstract

The intention of facilitating simultaneous execution for both latency sensitive and computing intensive Internet of Things (IoT) applications is consistently boosting the necessity of integrating Edge, Fog and Cloud infrastructure. There exists a notable number of real-world frameworks for attaining such integration. However, the limitations of existing frameworks in terms of platform independence, security, resource management and multi-application assistance resist the potentiality of integrated environment. Therefore, in this paper, we develop a simplified but effective framework, named *FogBus* for implementing end-to-end IoT-Fog(Edge)-Cloud integration. FogBus offers a platform independent interface to IoT applications and computing instances for execution and interaction. It not only assists developers in building applications but also supports users in running multiple applications at a time and service providers to manage their resources. In addition, FogBus applies Blockchain, authentication and encryption techniques to secure operations on sensitive data. Besides, it is easy to deploy, scalable, energy and cost efficient. To demonstrate the efficacy, we also design a prototype for Sleep Apnea analysis through FogBus framework. The experimental results of this case study show that different FogBus settings can improve latency, energy, network and CPU usage of the computing infrastructure.

Keywords: Fog Computing, Edge Computing, Cloud Computing, Internet of Things(IoT), Blockchain.

1. Introduction

In recent years the concept of Internet of Things (IoT) has been expanded across numerous domains. It enables different sensors, digital machines and objects to perceive the external environment and connects them to the global Internet for exchanging data. It also supports integration and analysis of generated data through application software so that events of interest can be identified and necessary physical actions can be triggered through actuators. Thus, it paves the way for building smart systems with limited human intervention [1]. Based on the current trends, it is expected that by 2025 such systems will incorporate over 1 trillion IoT devices with 50% increased demand of latency sensitive applications[2]. So far, Cloud is considered as the fundamental computing paradigm to deal with these large number of geographically distributed IoT devices and host corresponding IoT applications. However, Cloud datacenters reside at a multi-hop distance from IoT devices that increases communication delay in both transferring the data and receiving the application service [3]. For latency-sensitive applications such as respiratory monitoring and Sleep Apnea analysis, this inconvenient interaction between IoT devices and Cloud datacenters is unacceptable and can degrade the service quality drastically. In addition, IoT devices can generate a huge amount of data within a minimal time. When large number of IoT devices simultaneously initiate transferring the data to Cloud datacenters through global internet, severe network congestion occurs. In this circumstance, to overcome the limitations of Cloud-centric IoT model, Fog and Edge computing paradigms are emerged [4]. Both of them prefer to utilize edge resources provided by local computing instances for executing real-time IoT applications. IoT devices with computing processors, raspberry pi circuits, personal computers, mobile phones, network switches, gateway routers, micro-datacenters, IOx devices, etc. can offer potential edge resources. Based on the capabilities of edge resources, many consider Fog and Edge computing similar and use them interchangeably. Conversely, others treat Edge computing as a subset of Fog computing which is in accordance with the perspective of this work [5].

Fog computing manages an intermediate layer between IoT-enabled systems and Cloud computing. The computing instances of Fog computing are commonly known as Fog nodes and deployed across the edge network in distributed manner. Through these nodes, Fog provides Cloud-like services such as infrastructure, platform and software closer to the IoT data sources and supports application execution. Consequently, it reduces service delivery time, network congestion and improves Quality of Service (QoS) and user experience [6]. However, unlike Cloud datacenters, Fog nodes are resource constrained and heterogeneous. With limited resources, it is not possible to accommodate every compute intensive IoT application at the Fog layer. Therefore, seamless integration of IoT-enabled systems with Fog and Cloud infrastructure is required so that both edge and remote resources can be harnessed according to dynamics and requirements of the applications [7]. In this integration, Cloud-based top-down approach for managing Fog-based re-

[1]Department of Computer Science,
Indian Institute of Technology (IIT), Delhi, India
[2]Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
Email: mahmudm@student.unimelb.edu.au
[3]Department of Electrical Engineering,
Indian Institute of Technology (IIT), Delhi, India

sources becomes infeasible when IoT-data is being 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 this environment, both internal and external operations get obstructed by the heterogeneity of computing instances and resources. In such circumstance, generalized techniques can overcome the impediments of node to node communication and application runtime environment. Nevertheless, the implementation of such integrated environment going beyond the infrastructure and platform-level diversity with decentralized resource management policies is a challenging task. It further intensifies due to coexistence of multiple decision making entities and their coordination, multi-dimensional scaling, unaware topology and security issues [8].

In literature, there exists certain number of works implementing real-world frameworks for integrating IoT-enabled systems, Fog and Cloud infrastructure [9] [10] [11] [12]. However, most of these frameworks fail to support simultaneous execution of multiple applications and platform independence. Moreover, they do not promote users and application developers to pursue their intentions jointly. In some cases, the frameworks exploit Cloud resources only for data storage while in other scenarios compel energy constrained IoT devices to process raw data. To reduce the management overhead, some frameworks apply centralized techniques that eventually degrade the QoS of integrated environment. Besides, the frameworks often confine their concentration on few security aspects which in consequence increases vulnerability of the whole integrated environment. Identifying such limitations of available frameworks, in this paper we develop a lightweight framework named *FogBus*.

FogBus allows an end-to-end implementation of integrated IoT-Fog-Cloud environment through a wide range of devices. FogBus boosts platform independence both in terms of application execution environment and node-to-node interaction. It assists developers in building applications, users in consuming services and providers in managing resources. Furthermore, FogBus facilitates execution of latency sensitive applications through Fog computing and compute intensive application at Cloud datacenters. Thus it supports various type of application simultaneously. Besides, to ensure data integrity, protection and privacy FogBus implements Blockchain and applies authentication and encryption techniques that consequently increases its reliability. The **major contributions** of this work are listed as:

- A lightweight and simplified framework named FogBus that integrates IoT enabled systems, Fog and Cloud infrastructure and harness both edge and remote resources according to application requirements.
- Exploration of platform independent application execution and node-to-node interaction overcoming heterogeneity within the integrated environment.
- Design of a Platform-as-a-Service (PaaS) model that assists application developers, users and services providers to pursue individual interests.
- Development of a prototype for Sleep Apnea analysis in integrated IoT-Fog-Cloud environment.
- Implementation of block chain technique to ensure data integrity while transferring confidential data.
- Performance evaluation of FogBus in terms of latency, energy, network and CPU usage.

The rest of the paper is organized as follows. Section 2 highlights the principle of several existing frameworks. In Section 3 the description of FogBus framework is provided. The implementation process of FogBus is enlightened in Section 4. In Section 5 and 6, a case study on Sleep Apnea analysis and performance of FogBus is discussed respectively. Section 7 provides future works to improve FogBus and concludes the paper.

2. Related Work

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

While developing a prototype-based framework for IoT-enabled health-care system, Amir et al. in [9] enlightens the architecture a smart gateway for facilitating local storage and data processing. In the framework, Cloud acts as a back-end system for data analysis and decision making. To secure the framework, operating system level techniques are applied. Another prototype framework for smart health-care is developed by Dubey et al. in [10]. Intel Edison and Raspberry Pi circuits are used in the framework as Fog nodes. Through role-based authentication, the framework ensures privacy of the data. In the framework Cloud is partially adopted for storing the data. Azimi et al. in [13] also discuss a prototype framework for health-care solutions. It is hierarchical and the health analytics are divided into two parts for placing separately in Cloud and Fog infrastructure. The framework follows IBM proposed MAPE-K model to conduct the computations that inherently supports execution of diverse applications. There, for security purposes encryption techniques are extended.

Moreover, Gia et al. in [14] present a low-cost remote health monitoring framework that facilitates autonomic analysis of IoT data and notification generation. The IoT devices are designed with computational capabilities so that they can pre-process raw data and forward to the Fog nodes for further processing consuming less energy. In Fog layer distributed database is managed for data categorization and security. Orestis et al. in [15] also develop a prototype framework that allows users to share health data and notify during emergency. Each operation within the framework are managed by a Spark IoT Platform Core residing at the Cloud. The framework uses encryption and authentication techniques for security.

Chen et al. in [16] and Razvan et al. in [17] develop separate prototype framework for smart city-based surveillance and gas-leak monitoring system respectively. In both frameworks the

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	
Amir et al. [9]	✓	✓	✓			✓	✓			✓	✓
Dubey et al. [10]	✓	✓				✓		✓		✓	✓
Azimi et al. [13]	✓	✓	✓			✓	✓	✓		✓	✓
Gia et al. [14]	✓	✓				✓	✓			✓	✓
Orestis et al. [15]	✓	✓	✓			✓	✓			✓	
Chen et al. [16]	✓	✓								✓	✓
Razvan et al. [17]	✓	✓								✓	✓
Hu et al. [18]	✓	✓	✓							✓	
Yangui et al. [11]	✓	✓	✓	✓		✓	✓	✓	✓		
Bruneo et al. [12]	✓	✓		✓		✓	✓	✓	✓		
Verba et al. [19]	✓	✓		✓		✓	✓	✓	✓		✓
Yi et al. [20]	✓	✓	✓	✓		✓		✓	✓		✓
Korosh et al. [21]	✓	✓		✓			✓	✓	✓	✓	✓
Chang et al. [22]	✓	✓	✓	✓	✓	✓		✓	✓		
Nader et al. [23]	✓	✓	✓	✓	✓	✓		✓	✓		✓
FogBus [this work]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fog infrastructure, as a kernel conducts necessary data processing and decision making operations. The remote Cloud is partially adopted in the frameworks for building a historical record of IoT-data. However, the authors neither mention any security features nor techniques for managing heterogeneity of the Fog nodes within the framework explicitly. Likewise, Hu et al. in [18] bypass the security issues of their developed prototype framework for face identification system. In the framework, centralized Cloud manage all resources of integrated environment and offload partial computational tasks to Fog infrastructure. After completing the tasks on Fog, only the results are sent back to Cloud for further analysis and storage.

In order to promote IoT, Fog and Cloud integration, a Cloud-centric PaaS framework is developed by Yangui et al. in [11] that automates the provisioning of applications. The PaaS facilitates developing diverse applications, their deployment and management of the Fog nodes. The framework can deal with the heterogeneity of the nodes. In addition, security features from Cloud Foundry architecture are extended in the framework. Similarly, Bruneo et al. in [12] propose a Fog-centric PaaS framework for deploying and executing multiple applications over computationally sound IoT devices. There, Fog infrastructure acts as a centralized programmable coordinator. The framework is designed in such a way that it can apply Cloud-based security features and deal diverse applications without getting affected by heterogeneity of the instances.

In addition Verba et al. in [19] propose a gateway architecture that offers PaaS for integrating Fog nodes and IoT devices. The gateways assist messaging based communication with authentication techniques. The framework supports horizontal integration of gateways and Cloud datacenters for application deployment and task migration. In another work Yi et al. [20] propose a comprehensive PaaS framework for integrated environment. To implement the framework, resource enriched Fog nodes such as Cloudlet and ParaDrop are required and each node including IoT devices should be virtualized. For securing the framework operations, existing authentication techniques are applied. Korosh et al. in [21] also develop a PaaS framework that can manage energy for home and micro grid levels over Fog infrastructure. It can deal with the heterogeneity of Fog nodes and IoT devices and ensure data encryption.

The PaaS framework proposed by Chang et al. in [22] utilizes users networking devices to execute IoT applications. In the 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 as well. For security, it runs a registry services. Nader et al. in [23] also discuss a service oriented framework for managing smart-city based services through Fog and Cloud infrastructures. In the framework, services are classified in two types. The first one manages the core operations of the framework including resource management, security etc. and another type incorporates the requirements of specific applications. The security of the framework is ensured by authentication and access control mechanisms.

In the aforementioned frameworks, security issues are exploited from limited perspective. Besides, the computing capabilities of both edge and remote resources are not fully leveraged. In some cases, pushing computation towards IoT devices or resource enriched Fog nodes increases overall deployment cost and energy consumption. In addition, most of the frameworks overlook the heterogeneity within the computing infrastructures. They often fail to support multi-applications in contemporary basis and miss to offer flexible working arrangement for application developers and service consumers simultaneously. However, our proposed FogBus combines the concept of prototype and platform-based frame-work so that it can uphold the interests of both the parties. It facilitates data integrity through block chain and assist user authentication and data encryption side by side. FogBus expands the execution platform of different IoT applications from resource constrained Fog nodes to Cloud datacenters going beyond their diversity.

3. FogBus framework

The FogBus framework incorporates diverse hardware and software elements to offer structured communication and platform independent execution interface within integrated IoT-Fog-Cloud environment. Detailed description of each element of FogBus is given in following sections.

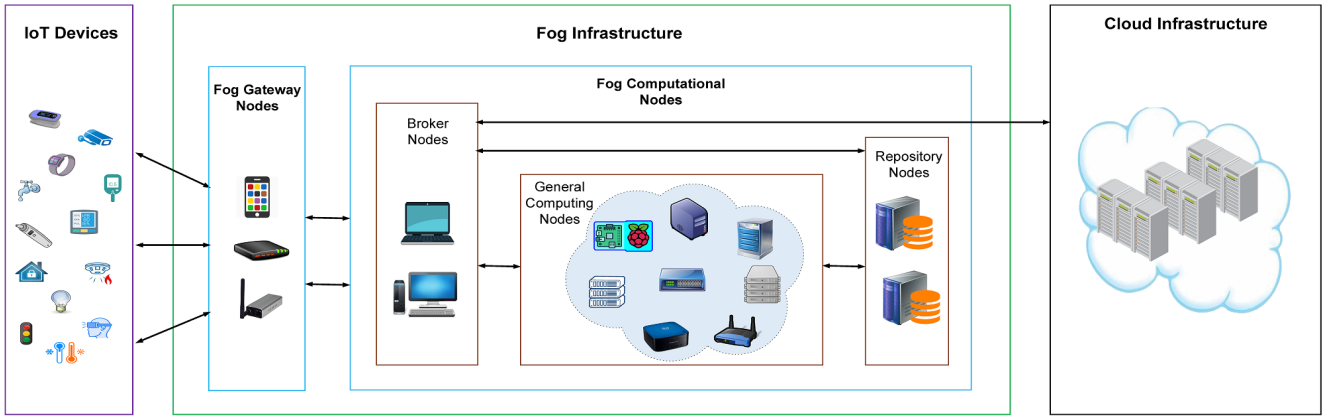


Figure 1: High level view of integrating IoT-Fog and Cloud through FogBus framework

3.1. Hardware Instruments

High level view of integrating various hardware instruments through FogBus is represented in Fig.1. In FogBus, the nature of interaction varies according to the types of hardware. The hardware instruments of FogBus include:

IoT Devices: IoT devices both, as a sensor perceive the external environment, and as an actuator converts any given command to physical action. 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 preprocess the generated raw data. FogBus allows IoT devices to connect with proximate gateways via wireless or wired communication protocols such as Zigbee, Bluetooth, NFC and 6LoWPAN. The sensing frequency of IoT devices can be tuned according to context of the system and the format of sensed IoT-data varies from device to device.

Fog Gateway Nodes (FGN): In FogBus framework, Fog Gateway Nodes (FGN) are the entry points of distributed computing infrastructure. FGNs assist IoT devices to get configured with integrated environment for placing and executing corresponding applications. Through FGNs, the FogBus framework offers user interface of applications so that the users can set authentication credentials, access the backend program, convey service expectations, receive service outcome, manage IoT-devices and request resources from computing infrastructure 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 particular IoT-enabled system. For large scale processing of the data, FGNs forward them to other computing instances of integrated environment. In attaining this purpose, FGNs maintain rapid and dynamic communication with accessible Fog nodes by the use of either Constrained Application Protocol (CoAP) or Simple Network Management Protocol (SNMP) [24].

Fog Computational Nodes (FCN): FogBus is designed in such way that it can deal with numerous Fog Computational Nodes (FCNs) simultaneously. FCNs are heterogeneous in terms of capacity and resource architecture. They are equipped with processing cores, memory, storage and bandwidth to conduct various FogBus operations. Based on these operations, FCNs can act in three different roles:

1. *Broker Nodes*: To handle the back-end processing of IoT-applications, FogBus facilitates the corresponding FGNs to connect with any of the accessible FCNs. This FCN initiates back-end processing of the application provided that required resources for the operation are available within it. Otherwise, it works as a broker node for the application. In this case, it communicates with other FCNs and Cloud datacenters on behalf the FGN to provision required resources for executing the back-end application. Sometimes, it distributes the computational tasks over multiple FCNs and seamlessly monitors, synchronizes and coordinates their activities. FogBus supports such broker nodes with adequate security features and fault tolerant techniques so that they can ensure reliability in communication and exchanging data among FGNs, FCNs and Cloud datacenters.
2. *General Computing Nodes*: For security issues, FogBus does not expose all FCNs directly to the FGNs. They are used for general computing purposes and accessible via broker nodes. The broker nodes also explicitly manage their resources and forwards the data along with executable back-end applications for processing. A general computing node can simultaneously serve multiple broker nodes without degrading consistency of their individual operations. In addition, computing nodes form clusters among themselves under the supervision of specific broker node while executing distributed applications.
3. *Repository Nodes*: Apart from conducting, broking and computing operations, some FCNs manage distributed database to facilitate data sharing, replication, recovery and security. The repository nodes offer interfaces for instant access and analysis of historical data. They also 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.

Cloud Datacenters: In some cases when Fog infrastructure becomes overloaded or service requirements are latency-tolerant, FogBus extends resources from Cloud datacenters to

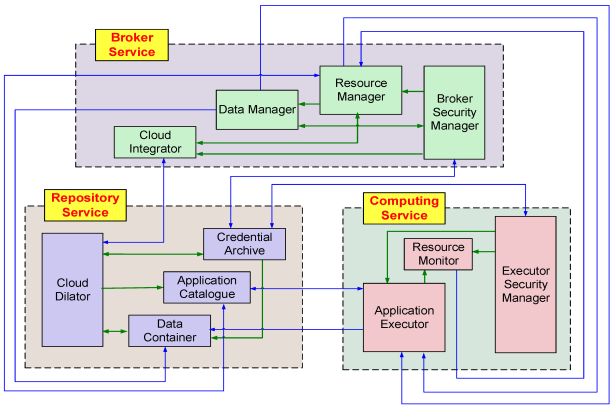


Figure 2: Interaction of different software components within FogBus framework

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 platform independent software components. These components are broadly classified into three types of system services. The Broker service manages all 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 general computing node. When 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. However, these services are interrelated. The interaction among different FogBus software components are represented 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 particular FGN, the Broker Security Manager verifies them in association with Credential Archive of Repository Service. The Credential Archive also assist this component with required security certificates for remote Cloud integration. The Broker Security Manager itself 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. Besides, 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 general computing nodes through Resource Monitor of Computing service. The Cloud Integrator assists the Resource Manager with contextual data of Cloud-based instances such as virtual machines and containers. After attaining all information regarding resources and applications, Resource Manager provisions required resources on FCNs and Cloud for applications. In this case, for FCNs, Application Executor of Computing service and for Cloud, their internal software system helps the Resource Manager. Moreover, FogBus facilitates service providers to apply various policies in Resource Manager, while provisioning resources for applications. In addition, it maintains a resource configuration file that tracks the addresses of FCNs and Cloud instances along with deployed applications so that subsequent data of the 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 these 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 in encrypted manner on Data Container of Repository service for further usage. 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 of the stream to the processing destination.

Cloud Integrator: All interactions of FogBus framework with Cloud is handled by Cloud Integrator. It notifies the context of Cloud instances to the framework and forwards the storage and resource provisioning commands to the Clouds. Through this component, FogBus not only offers interface to the providers for developing customized Cloud-integration scripts but also facilitates access to third-party software systems to deal with multiple Cloud datacenters simultaneously.

3.2.2. Repository Service

Credential Archive: Users authentication credentials are set during IoT device configuration and 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 of the corresponding nodes.

Application Catalogue: This component is responsible for maintaining the details about various types of applications including their operation, 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 synchronizes the applications on allocated resources in association with the Application Executor of Computing service.

Data Container: Data received from IoT devices is 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 help 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 assist 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. In this case, 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 of computing resources are monitored by this component in association with the Application Executor. Its perceived information helps Resource Manager to provision resources for different applications. It also tracks the runtime QoS requirements of the application and the performance of the resources in meeting them. When allocated resources perform less than expectations, in dealing with the application or their uncertain failure occurs, this component immediately notifies the Resource Manager to initiate required actions such as dynamic resource provisioning, application execution migration and intermediate data storage.

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 form Application Catalogue to 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 to Data Container for making the framework fault tolerant.

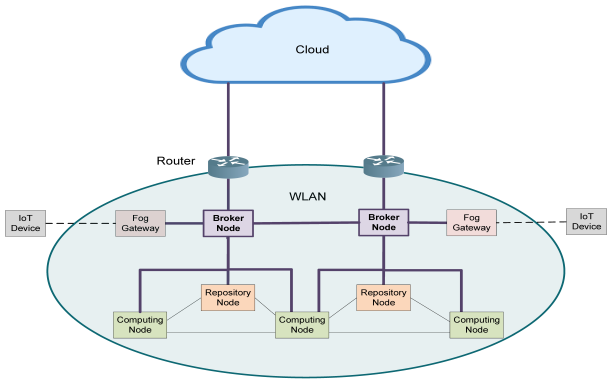


Figure 3: Network Structure of FogBus framework

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 very necessary. Besides, it is also required to ensure that hardware instruments do not become overwhelmed with the communication burdens. 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 the 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 are connected with a common wireless local area network (WLAN) that is managed by one or multiple routers.

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 own chain of blocks and separate database on the workers. In addition, the software

components running at the masters facilitate Fog infrastructure to incorporate with multiple Cloud datacenters simultaneously.

Reliability: The facility of running multiple masters implicitly eradicates the inherent single point dependency limitation of master-worker model within FogBus. Besides, the framework allows each master to replicate their image over one of its worker nodes. It is done so that during uncertain failure of that master, corresponding worker gets 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. In addition, the masters 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 that worker's information with other workers so that residual data processing can be started immediately. If all the workers of a master are overloaded, workers of other masters are taken into consideration. In this case, all the masters maintain an internal communication among themselves. Thus, the computation facility remains always available within the FogBus framework.

Security: The inclusion of FGNs and FCNs in FogBus provided network to 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. Besides, in FogBus, multiple communication links 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: The FogBus framework utilizes the WLAN bandwidth dedicatedly for a specific system. Since the network resources are not shared to external entities, the overall performance of the system from network perspective does not degrade. 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. Besides, FogBus framework 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 reduces the capital and the operational expenditure for the service provider.

4. Implementation

Protocol, execution environment, scripting and programming language used in FogBus are supported by all hardware in the integrated environment. It eventually helps FogBus to function going beyond the infrastructure heterogeneity. The implementation of different FogBus elements are described as follows.

4.1. System Services

The system services of FogBus framework are 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 Fog nodes within the WLAN. Usually these PHP based web programs can function in every operating system such as Unix, Windows, Linux and NetWare. 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. In FogBus, an Apache server is set in each FCN to run the web programs of corresponding system services. In addition, MySQL servers are placed in Broker and Repository nodes to manage databases and their operations. However, each system service of FogBus is divided into service interface and management activity. Apart from Blockchain, the service interface and management activity of each system service handles other security aspects of the FogBus framework. Moreover, in masters, the service interface assists in receiving data and user's specifications from the gateway devices and representing 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 policies, updates the configuration files and forwards commands for the workers. The worker's service interface functions as a receptor of the corresponding node and is responsible to decode the output file of applications to the masters. Based on the received signals, the management activity of worker functions in: monitoring of the resources, circulating their status to the masters, allocation of resources and storing data in relational databases. It also creates input file for backend program of applications with the received data from masters.

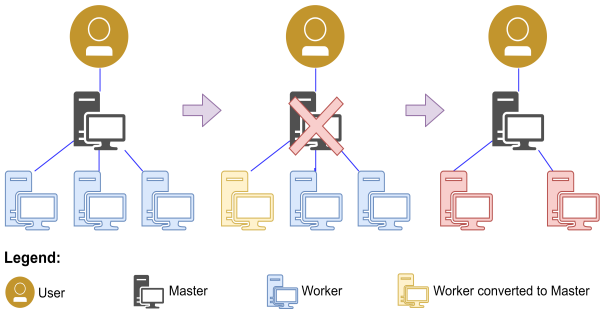


Figure 4: Ensuring reliability in FogBus framework

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 [25]. 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. Besides, 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 and its proof-of-work is sent to other nodes for linking with their local chains. In this operation digital signature is used so that source of the block at the destination can be verified. However, in such context, 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 becomes invalid. To make the chain valid again, hash of the invalid blocks requires to recalculate. Besides, the proof-of-work of each block requires to be generated again. Both of these operations are time consuming and compute intensive. Moreover, this fraudulent manipulation of data in a chain 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 constraint [26].

In FogBus, the masters create the blocks from received data and calculate the hash of each block using SHA256 algorithm. 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 are able to 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. Besides, each block and its hash are verified at the workers using the associated 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 service providers to track the data/block flow through the service interface of masters by displaying the latest hashes of the Blockchain copy at each worker. Thus, it helps users and service providers to take necessary actions on suspicious activity within the FogBus network. In FogBus, the Blockchain is developed in Java programming language. Compiled Java programs usually run on Java Virtual Machine, JVM, which can be easily installed across diversified platforms. Hence, the Blockchain utility of FogBus can operate in wide range of operating systems. In different FCNs of FogBus, this utility directly interacts with the corresponding system services.

4.3. Cloud Plugin

In FogBus, the service interface of 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 becomes activated. For other operations such as storage and distribution, the management activity of 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 third-party Cloud Plugin services, the master is required to configure 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 [27]. 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 [28].

According to the built-in resource provisioning policy of FogBus, at first Fog infrastructure is exploited to process data, later Cloud infrastructure is referred. For the second case in FogBus, the management activity of a master stores the data in a Cloud input file. The Aneka-based Cloud plugin at the master parses this file in every 500 millisecond 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 framework supports the execution and deployment of applications of different IoT-enabled systems. In FogBus these applications are divided into user interface and backend program. Although applications are not the part of FogBus software components, FogBus offers developers some guidelines to build their user interface and back-end programs aligned with the features of FogBus framework. The required specifications of user interface and back-end program of applications are described as follows.

4.4.1. User Interface

The user interface 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 user interface should be supported by these platforms. Moreover, for some applications, user interface requires to store data temporarily. On that note, the developers should use compatible database system and schema to these platforms. Besides, the user interface 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 user interface should support both general and low energy Bluetooth

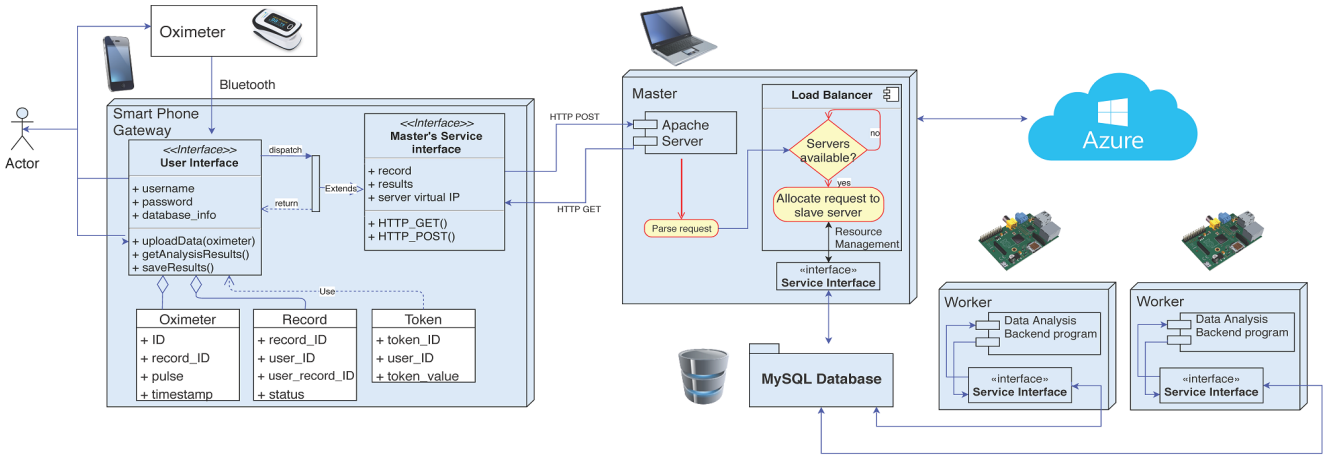


Figure 5: FogBus framework enabled system model for Sleep Apnea analysis

interactions. In FogBus framework, user interface is directly correlated with the service interface of the masters for forwarding IoT data and user information, and receiving the service outcome. For simplicity of the interaction, the user interface can be designed in such a way that easily parses the web programs of service interface running at the master.

4.4.2. Backend Program

In FogBus, the backend program of applications is executed in the FCNs. Since the FCNs are distributed, to fully leverage their capabilities it is preferable to build the backend program in distributed manner. In this case, modular development of backend program can be applied by the developers. In addition, the execution of backend program should not be obstructed by the heterogeneity of FCNs. To address this issue, developers can use cross platform programming languages to develop the backend program. While developing the backend program some specific points within the script should be specified so that application's intermediate data on those points can be stored during execution. Besides, the backend 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 of time during sleep. It reduces oxygen level in blood of the patient, downs the heartbeat rate and resembles that the patient has stopped breathing. In this case, the brain is triggered to wake up for forcing the breathe. It can happen very frequently and create severe obstruction in sound-sleep of the patient. If the brain fails to respond quickly or oxygen saturation becomes significantly lower specially for aged and asthma patients, Sleep Apnea could provoke cardiac failure or brain stroke of the patients. However, Sleep Apnea is a very common disease although most of the people either ignore or unknown to it. To determine the intensity of Sleep Apnea on a person, it is required to monitor his/her 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 invites other complications [29].

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 represents the Sleep Apnea intensity in proportional manner. Currently, to conduct the Sleep Apnea analysis, hospital or laboratory-based machineries are required which are expensive for each individual to own. 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 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 Model

The system model for FogBus-enabled Sleep Apnea analysis prototype is represented 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.

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.

Worker Node: Raspberry Pi 3 B+, *ARM Cortex-A53 quad-core SoC CPU @ 1.4GHz 1GB LPDDR2 SDRAM*, IEEE 802.11, 64-bit, Raspbian Stretch, Apache HTTP Server 2.4.34, Java SE Runtime Environment (JRE) 1.6, MySQL 5.6.

Figure 6: Real-world implementation of FogBus-based Sleep Apnea analysis

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

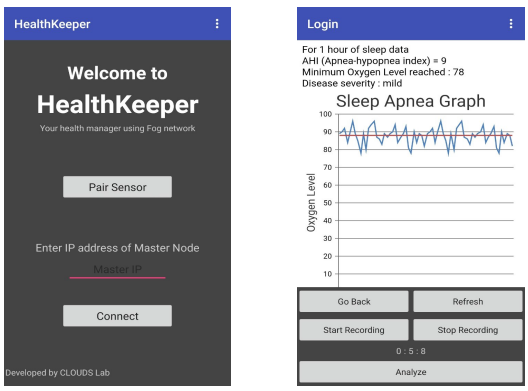
Fig. 6 depicts the real implementation of this system model.

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 interface and a data analytic. Description of the installed package is given below.

5.2.1. Android Interface

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. 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



(a) (b)

Figure 7: (a) Home and (b) Session Screen of the android interface

sending data manually through the HTML form, the interface records and transmits data automatically. Here, 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 of the master and displays the result to the operator once they become available to the master.

5.2.2. Data Analytic

The data analytic for Sleep Apnea analysis encapsulates two open source programs found in [30][31]. 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. Usually, AHI < 5 per hour refers to no or minimal Sleep Apnea. It becomes mild when AHI ≥ 5 but < 15 per hour. Moderate Sleep Apnea occurs if AHI ≥ 15 but < 30 per hour. Sleep Apnea turns into severe for AHI ≥ 30 per hour. However, as additional information, the data analytic stores the minimum oxygen level for the given period of time. 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 data 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 represented 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 service interface of the master, this acknowledgement becomes visible to the operator.

After recording the data for a certain period of time, 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

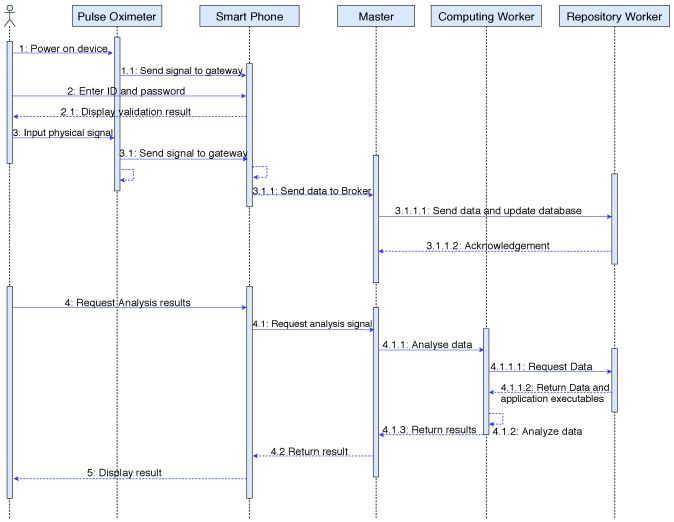


Figure 8: Sequence of communication during Sleep Apnea analysis

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.

6. Performance Evaluation

6.1. Experimental Setup

The prototype for Sleep Apnea analysis discussed in Section 5 is used to study the performance of FogBus in terms of latency, energy, processor (CPU), memory (RAM), storage (Cache) and network usage. For experiments, data from multiple pulse oximeters are recorded for a specific period of time, later the master sequentially generates analysis tasks for each recorded data chunks to the computing worker. Here, each experiment scenario is modeled under the following settings.

1. *With / Without Interval*: In With Interval setting, master sends the next analysis task to its computing worker after 5 seconds of receiving the outcome for previous task. This time interval helps both master and computing worker to reduce their overhead. On the contrary, in Without Interval case, master sends the next task to its computing worker as soon as the outcome of the previous task becomes available. It ensures that the FogBus framework remains consistently active and there exit 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 differs from each other based on the status of Blockchain security feature in the FogBus-based prototype.
3. *Fog / Cloud Only*: FogBus supports application execution across diverse computing infrastructures. This experiment setting refers whether the application execution is solely conducted on Fog or Cloud 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. Apart from the system model parameters specified in Section 5.1, additional parameters used for the experiments are given in Table 2.

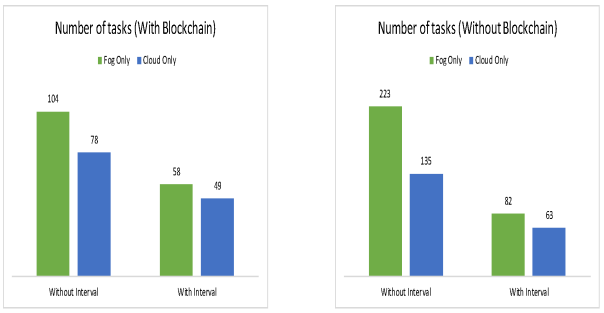
Table 2: Experiment parameters

Parameter	Value
Analysis Task:	
Duration of sequential task generation	5 minute
Data recording time per task	3 minute
Pulse Oximeter:	
Signal length	18 KB
Sensing frequency	2 signal per second
WLAN:	
Download Speed	7 Mbps
Upload Speed	2 Mbps

6.2. Result Analysis

6.2.1. Number of Tasks

Fig. 9 depicts the number of tasks generated in FogBus on different experiment settings. It is observed that the number of tasks is higher in the Fog Only setting compared to the Cloud one. It happens due to receiving outcome of the previous task quickly from the Fog infrastructure. During Without Interval setting, this value rises significantly than the With Interval setting since tasks are generated continuously by the master. It is also noticed that, if Blockchain feature of FogBus is turned off, comparatively higher number of tasks 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 task. Based on these observations, it is understood that if there exists higher amount of tasks to be handled, 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 tasks and the size of data chunk for individual task. It can be managed by tuning the interval between subsequent tasks creation.



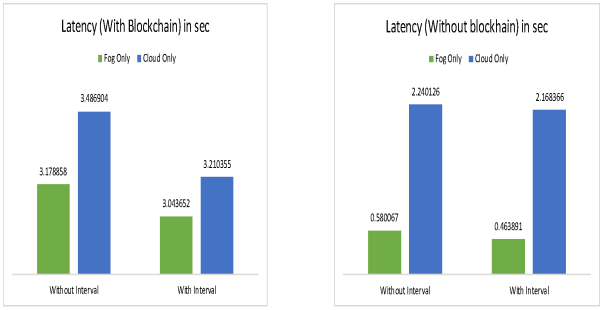
(a) (b)

Figure 9: Number of tasks (a) With and (b) Without Blockchain

6.2.2. Latency

Fig. 10 represents the impact of different settings of FogBus on service delivery latency. Here service delivery 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. Besides, if the size of data chunk for a particular task 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 task 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 case. This latency becomes much lower on disabled state of Block chain feature since its management add some more time to complete the tasks. 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 tasks having stringent service delivery deadline.



(a) (b)

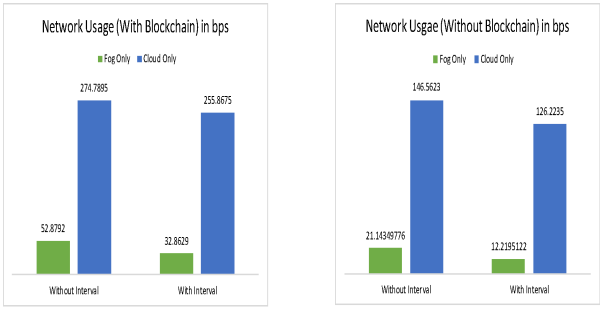
Figure 10: Latency (a) With and (b) Without Blockchain

6.2.3. *Network Usage*

Network usage in different settings of FogBus are represented in Fig. 11. In this experiment, Fog Only setting provides improved performance compared to Cloud Only 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 tasks are generated and their associated data and information are exchanged. In this case, tuning of subsequent task 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 a particular IoT-enabled system.

6.2.4. *Energy*

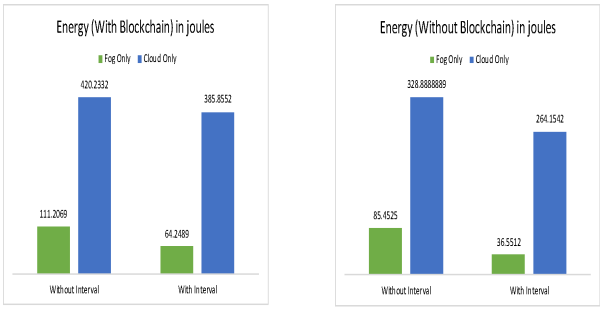
Fig. 12 represents how different settings of FogBus influence energy consumption of the infrastructures. In Cloud Only setting the Fog nodes are used for networking and Cloud VMs conducts the computation whereas in Fog Only setting both the networking and computation are handled by Fog nodes. Since, Cloud VMs consume much more energy compared to the Fog



(a) (b)

Figure 11: Network usage (a) With and (b) Without Blockchain

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 task creation assists to improve the energy usage of the infrastructure. However, it leads FogBus to process less number of tasks which can be overcome by efficient tuning of the interval time. Nevertheless, these configurations help FogBus to execute applications under the energy constraints.

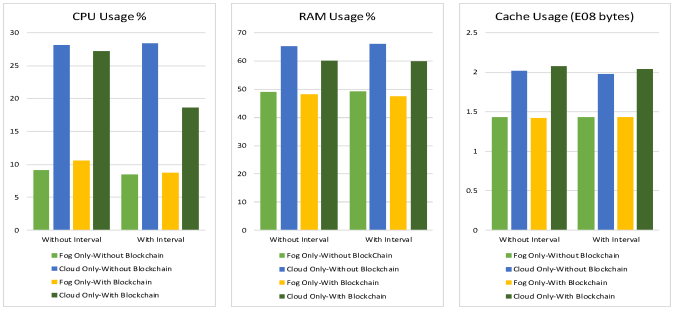


(a) (b)

Figure 12: Energy consumption (a) With and (b) Without Blockchain

6.2.5. *CPU, RAM, Cache Usage of Master*

The Fig. 13 shows the CPU, RAM and Cache usage of mater for various FogBus settings such as Fog Only-Without Blockchain, Cloud Only-Without Blockchain, Fog Only-With Blockchain and Cloud Only-With Blockchain. The parameter values for Fog Only setting are much lower compared to Cloud Only case since it reduces the overhead of running Cloud Plugins and storing the Cloud communication attributes. Even Without Blockchain, these parameters value also decrease as hash and proof-of-work creation for each data block are eliminated. Moreover, on any resource constrained master, these settings can ensure acceptable performance of the FogBus framework. However, since the software components of FogBus do not release their allocated resources after operations by themselves, the RAM and the cache usage for master in all setting remain



(a) (b) (c)

Figure 13: (a) CPU, (b) RAM and (c) Cache utilization of master

almost same. In this case, the interval in subsequent task generation does not improve the state of resources. It is left to be resolved in the next version of FogBus.

7. Conclusion and Future Works

In this work, we implement the FogBus framework that can integrate different IoT enabled systems to both Fog and Cloud infrastructures simultaneously. The framework is lightweight and can harness both edge and remote resources for IoT application deployment, monitoring and management. FogBus is developed is cross platform programming languages that helps to overcome the heterogeneity of the infrastructure during application execution and end-to-end interaction. Besides, 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 tune their service expectations, 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 secure data transfer across unreliable network, encryption techniques are applied in FogBus. Based on the principles of FogBus a cost efficient prototype for Sleep Apnea analysis is also developed in this work. Applying different FogBus settings on the prototype, it is demonstrated that FogBus performs well when large number of tasks are required to be processed, the execution of tasks are latency sensitive, network resources are not abundant, energy usage is constrained and computing instances are not resource enriched. However, it is the very first version of FogBus and provides a larger scope for further improvement in the following aspects:

Resource management policies: FogBus provides flexibility to apply customized provisioning polices while allocating resources for different applications. Resource management policies can be developed targeting load balancing among the computing infrastructures, the QoS enhancement and others.

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.

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.

The Source Code of FogBus software is available on `https://github.com/Cloudslab/FogBus`

References

[1] Jayavardhana Gubbi and Rajkumar Buyya and Slaven Marusic and Marimuthu Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645 – 1660.

[2] McKinsey & Company, The Internet of Things: How to capture the value of IoT (May, 2018).

[3] M. Afrin, M. R. Mahmud, M. A. Razzaque, 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), 2015, pp. 495–498.

[4] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog Computing and Its Role in the Internet of Things, in: Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, ACM, 2012, pp. 13–16.

[5] R. Mahmud, R. Kotagiri, R. Buyya, Fog computing: A taxonomy, survey and future directions, Springer, 2018, pp. 103–130.

[6] R. Mahmud, S. N. Srirama, K. Ramamohanarao, R. Buyya, Quality of experience (qoe)-aware placement of applications in fog computing environments, Journal of Parallel and Distributed Computingdoi:https://doi.org/10.1016/j.jpdc.2018.03.004. URL http://www.sciencedirect.com/science/article/pii/S0743731518301771

[7] R. Mahmud, F. L. Koch, R. Buyya, Cloud-fog interoperability in iot-enabled healthcare solutions, in: Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN '18, ACM, New York, NY, USA, 2018, pp. 32:1–32:10.

[8] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, R. Buyya, 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) (2017) 1275–1296.

[9] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, P. Liljeberg, Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach, Future Generation Computer Systems 78 (2018) 641–658.

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

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

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

[13] I. Azimi, A. Anzanpour, A. M. Rahmani, T. Pahikkala, M. Levorato, P. Liljeberg, N. Dutt, HiCH: Hierarchical fog-assisted computing architecture for healthcare IoT, ACM Transactions on Embedded Computing Systems (TECS) 16 (5s) (2017) 174.

[14] T. N. Gia, M. Jiang, V. K. Sarker, A. M. Rahmani, T. Westerlund, P. Liljeberg, H. Tenhunen, 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, 2017, pp. 1765–1770.

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

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

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

[18] P. Hu, H. Ning, T. Qiu, Y. Zhang, X. Luo, Fog computing based face identification and resolution scheme in internet of things, IEEE transactions on industrial informatics 13 (4) (2017) 1910–1920.

[19] N. Verba, K.-M. Chao, A. James, D. Goldsmith, X. Fei, S.-D. Stan, Platform as a service gateway for the Fog of Things, Advanced Engineering Informatics 33 (2017) 243–257.

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

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

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

[23] N. Mohamed, J. Al-Jaroodi, S. Lazarova-Molnar, I. Jawhar, S. Mahmoud, A service-oriented middleware for cloud of things and fog computing supporting smart city applications, in: 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, 2017.

[24] M. Slabicki, K. Grochla, Performance evaluation of coap, snmp and netconf protocols in fog computing architecture, in: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, 2016, pp. 1315–1319.

[25] G. Zyskind, O. Nathan, et al., Decentralizing privacy: Using blockchain to protect personal data, in: Security and Privacy Workshops (SPW), 2015 IEEE, IEEE, 2015, pp. 180–184.

[26] M. Swan, Blockchain: Blueprint for a new economy, " O'Reilly Media, Inc.", 2015.

[27] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds, Future Generation Computer Systems 28 (6) (2012) 861–870.

[28] R. N. Calheiros, A. N. Toosi, C. Vecchiola, R. Buyya, A coordinator for scaling elastic applications across multiple clouds, Future Generation Computer Systems 28 (8) (2012) 1350–1362.

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

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

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