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 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 components for execution and interaction. It not only assists developers in building up 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 and model different scenarios. The experimental results of this case study show that FogBus outperforms the benchmark approach in terms of latency, energy, network and CPU usage.

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 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 of building smart systems with limited human intervention. 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[1]. 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. 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 is occurred. In this circumstance, to overcome the limitations of Cloud-centric IoT model, Fog and Edge computing paradigms are emerged. Both of them prefer to utilize edge resources provided by local computing components for executing real-time IoT applications. IoT devices with computing processors, raspberry pi circuits, personal computers, mobile phones, network switches, gateway routers, proxy servers, 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 accord with the perspective of this work.

Fog computing arranges an intermediate layer between IoT-enabled systems and Cloud computing. The computing components 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. However, unlike Cloud datacenters, Fog nodes are resource constrained and heterogeneous. With limited resources, it is not possible to accommodate every compute intensive IoT applications 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. In this integration, Cloud-based top-down approach for managing Fog-based resources

[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

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 provisioning resources following distributed bottom-up approach. Moreover, while placing and executing applications on the integrated environment, both internal and external operations get obstructed by the heterogeneity of computing components and resources. In this 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 entity and their coordination, multi-dimensional scaling, unaware topology and security issues.

In literature, there exists certain number of works implementing real-world frameworks for integrating IoT-enable systems, Fog and Cloud infrastructure [2] [3] [4] [5]. 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 constraint 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 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 up applications, user 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 block chain 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 provides 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 principal 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 some future direction to improve FogBus and finally Section 8 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 healthcare system, Amir et al. in [2] 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 healthcare is developed by Dubey et al. in [3]. 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 [6] also discuss a prototype framework for healthcare 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 introduced MAPE-K model to conduct the computational tasks. The MAPE-K model inherently supports execution of diverse applications. There for security purposes encryption techniques are extended.

Moreover, Gia et al. in [7] 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 catagorization and security. Orestis et al. in [8] also develops 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 [9] and Razvan et al. in [10] 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. [2]	✓	✓	✓			✓	✓			✓	✓
Dubey et al. [3]	✓	✓				✓		✓		✓	✓
Azimi et al. [6]	✓	✓	✓			✓	✓	✓		✓	✓
Gia et al. [7]	✓	✓				✓	✓			✓	✓
Orestis et al. [8]	✓	✓	✓			✓	✓			✓	
Chen et al. [9]	✓	✓								✓	✓
Razvan et al. [10]	✓	✓								✓	✓
Hu et al. [11]	✓	✓	✓							✓	
Yangui et al. [4]	✓	✓	✓	✓		✓	✓	✓	✓		
Bruneo et al. [5]	✓	✓		✓		✓	✓	✓	✓		
Verba et al. [12]	✓	✓		✓		✓	✓	✓	✓		✓
Yi et al. [13]	✓	✓	✓	✓		✓		✓	✓		✓
Korosh et al. [14]	✓	✓		✓			✓	✓	✓	✓	✓
Chang et al. [15]	✓	✓	✓	✓	✓	✓		✓	✓		
Nader et al. [16]	✓	✓	✓	✓	✓	✓		✓	✓		✓
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 [11] 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 [4] 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 [5] 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 computational components.

In addition Verba et al. in [12] proposes 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. [13] 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 [14] 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 [15] 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 incorporate user's handheld devices as well. For security, it runs a registry services. Nader et al. in [16] 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 application. The security of the framework is ensured by authentication and access control mechanisms.

In the aforementioned frameworks, security issues are not exploited from different 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. Detail description of each elements of FogBus is given in following sections.

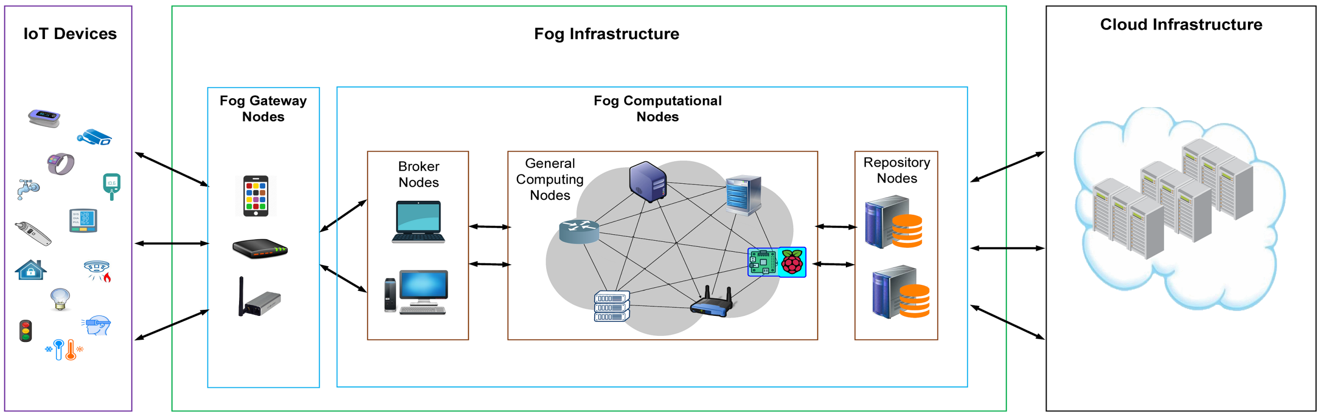


Figure 1: High level view of integrating IoT-Fog and Cloud through FogBus framework

3.1. Hardware Instruments

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

IoT Devices: IoT devices both as a sensor perceives the external environment and as an actuator converts any given command to physical action. Usually IoT devices are energy and resource constraint and act as mere data producer or consumer. In some case, IoT devices are equipped with limited computation capabilities to preprocess generated raw data. FogBus allows IoT devices to connect with proximate gateway machines 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, FogBus offers users the front-end interface of applications so that they can set authentication credentials, access the back-end application, 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 these data, FGNs forward them to other computing components of integrated environment. In attaining this purpose, FGNs maintain rapid and dynamic communication with accessible Fog nodes by dint of either Constrained Application Protocol (CoAP) or Simple Network Management Protocol (SNMP).

Fog Computational Nodes (FCN): FogBus is designed in such way that 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 communicate 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 manages 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 manages distributed database to facilitate data sharing, replication, recovery and security. The repository nodes offer interfaces for instant access and analysis of historical data. In addition they maintain meta-data of various applications including their application model, runtime requirements and dependency. Moreover, repository nodes can preserve some intermediate data during application execution so that data processing can be initiated immediately from any anomaly-driven stopping point.

Cloud Datacenters: In some cases when Fog infrastructure becomes overloaded or service requirements are latency-

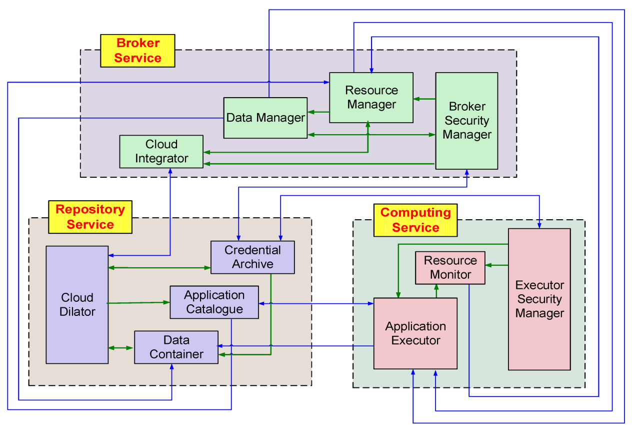


Figure 2: Interaction of different software components within FogBus framework

tolerant, FogBus extends resources from Cloud datacenters to execute back-end IoT applications. Through Cloud datacenters, FogBus expand the computing platform for IoT applications across the globe. In association with Fog repository nodes, they facilitate extensive data storage and distribution so that access and processing of data become location independent.

3.2. Software components

In order to simplify the IoT-Fog-Cloud integration, FogBus provides various software components. These software components are classified into three categories. The broker service manages all the functionalities of a broker node whereas the computing service is responsible to control the operations of a general computing node. In some cases when broker node itself initiates the execution of back-end application, the computing service is triggered within it. Conversely, Repository service can run across all the Fog nodes. However, the software components 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: On reception of users 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 and maintain separate chain for individual data streams. 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 framework and offers other components flexible accesses to their required information.

Resource Manager: This component is responsible for selecting suitable resources to execute applications. Resource Manager identifies the requirements of different applications from Application Catalogue of Repository service. It perceives the resource status within each broker node and general computing nodes through Resource Monitor of Computing service. The Cloud Integrator assists the Resource Manager with contextual data of Cloud-based virtual machines. 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 help the Resource Manager. Moreover, FogBus facilitates service providers to apply various policies in Resource Manager while provisioning resources for applications. In addition, this component maintains a resource configuration file that tracks the addresses of FCNs and Cloud virtual machines along with deployed applications so that subsequent data of the streams can directly be sent to the allocated resources for processing.

Cloud Integrator: All the interactions of FogBus framework with Cloud is handled by Cloud Integrator. It notifies the context of Cloud virtual machines to the framework and forwards the command for storage and resource provisioning to the Clouds. Besides, this component can interact with multiple Cloud data centres simultaneously

Credential Archive: Users authentication credentials those are set during IoT device configuration with integrated environment, are preserved in the Credential archive. Credential Archive distributes the security keys and details of each data blockchain generated by the broker nodes to others. This component also provides the Secure Socket Layer (SSL) and Transport Layer Security (TLS) certificates for Cloud integration to the broker nodes.

Data Manager: IoT device sensed and pre-processed data are received by Data manager component running on specific broker node. With these data, blocks and their chain are created for maintaining the data integrity. 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.

Executor Security Manager: The seamless secured interactions of general computing nodes with specific broker node are managed by the Executor Security Manager. This operation is managed based on the received security keys from Credential Archive of Repository service.

Application Catalogue: Another task performed by the Network Manager is to synchronize the Jar file used for analysis. This service sends a request to the manager page of each worker node to synchronize the executable jar file from master. The worker node then downloads the masters copy of the executable and overwrites its own with it.

Data Manager: This module performs the most of the software functionalities. It takes data as input using GET request, after which if analysis request is sent then the data is sent to the FogBus framework for computation. This module also handles the Fail over and Load Balancing schemes of the software. The PHP method file_get_contents() allows us to get a string form of the webpage passed as the argument. An error operator lets it return FALSE in case of errors, and in this case that IP is removed from the set of worker IPs for that data and a 'worker compromise' error is displayed. The next time the worker IPs are accessed for load, this is checked again and if the particular worker node is available, it is taken into the load balancer. If the load of any worker is <80%, then the task is given to the worker with least load. The task is sent using the GET method to the Worker Module of the worker with that IP address. Otherwise, randomly either the master or Aneka gets the task. If the task is sent to master, then it is sent using GET method the Worker Module in the local machine. If the task is sent to Aneka, then it is sent using the GET method to the Aneka Interface. The load balancing strategy used is shown in Figure 6. This module also sends the block details to the other worker nodes. It saves the data received from the sensors to a data file which the Master Interface java application uses to form a new block. The new block hash values and proof-of-work with public key and signature are shared to each node for updating blockchains. If any node reports error in terms of blockchain tampering or signature verification, then the blockchain in majority of the network is copied to that node. Also dispay the results

Cloud Integrator: The framework at the Aneka side can be described in terms of following major modules. This Module acts as an interface between the Aneka software and the master session script. It receives data to be analyzed from the session script and saves it to the data file. When analysis is complete, the contents of the result file are displayed on the webpage to be parsed by Master Node.When the Aneka software gets the data for analysis it itself decides to which container the data needs to be sent if it is a Task based implementation or can be sent to multiple containers when in Thread based implementation. The Aneka code (in C#) parses the data file every 500ms and checks if pending analysis exists. If yes then it forms Aneka tasks/threads and launches it to other Aneka containers which might be on the cloud or in local network. Here also blockchain and digital signature management have been implemented

3.2.2. Computing Services

The framework at the Worker Node side can be described in terms of following major modules.

- *Resource Monitor* It is a PHP based module that keeps a real time information of the load on the worker node. This Monitor is accessed by the Master Node for it's load balancing scheme implementation.

- *Network Manager* The workers manager page allows user to set the Master IP address which is used for synchronizing the executable Jar file. It also displays the current set IP address of the Master which is saved in the configuration file and the analysis file name that is being used for computation on the data. The Network Manager also keeps track of the Master Node's public key that is used to sign the data sent to the worker.

- *Worker Module* This module receives data to be analyzed using the GET method. It acts as an interface between the Master Node and the Java based analysis application.

- *Analysis Application* It performs the computation on the data sent by the Master node and sends results to the Worker Module to be forwarded to the Master Node.

- *Worker Interface* This Module takes in the updated block details from the master node, analyzes them and adds them to it's blockchain if validation checks pass. If the Worker Interface application throws any exception in terms of data tampering or signature verification, it is reported to Master node. Data validation includes the following checks:

  – Checking the source:

    ∗ Public Key received matches the registered public key of master node

    ∗ Signature Verification of data received with the public key

    ∗ The tuple of data, public key and signature is not same as last

  – Checking the block:

    ∗ Hash of the block matches the calculated hash value

    ∗ Proof-of-work follows the required pattern

    ∗ Previous hash corresponds to the hash of the latest block

  If the source verification fails then block is ignored and data breach is informed to the master node. If block details check fails then the errors are reported to master. If the block is valid, it is added to the blockchain and no errors are reported to Master.

3.3. Network Structure

User Interface: The means by which the user and the FogBus system interact to access, use or instruct some or many nodes across the network. Figure 1 depicts how different physical components, as described in section 3.2, are connected in the FogBus model. This figure only shows a connection of Broker and worker nodes for one geographical location. The FogBus framework allows *n* number of Brokers for *n* different geographical locations. This *n* can be scaled to large values as described in the next subsection. The User connects his/her device to the local network and interacts with the Master Node using an interface which can be an Android or other smart device. This interface allows communication between sensor/actuator with the Master Node.
The Master Node, also referred to as the Broker Node, collects data from the sensor and uses the data itself for analysis or sends it to one of the following:

- Worker Node in local Network

- Private Cloud
- Public Cloud or data center

All communication among the Master Node, Worker Nodes, Private Cloud and User is facilitated by the Network Router. Additional workers can be installed in the network using Switches. Communication of the Master Node and the Public Cloud is facilitated with Internet, and the Network Router directs Internet data packets through the connection provided by the Internet Service Provider.
The Master Node collects results/actions from the devices mentioned above and send it to user interface or actuator. The master or worker nodes can also receive data directly from data sets and perform actions accordingly.
It is in the hands of the user, which service to run and which master to connect to. There can be Master nodes dedicated for specific tasks and in IoT environment multiple services can even run simultaneously requiring connections with multiple masters and/or multiple service running on same master node.

3.3.1. Scalability of network architecture

Scalability of an IoT framework is highly crucial to allow addition of worker nodes easily when the computational demand increases. The traditional Master/Slave system lacks this feature as inherent to it there is one master and several workers connected to the same Master. This can lead to deadlocks when there are several simultaneous requests and the data is being sent to or received by several worker nodes concurrently. To tackle this problem the FogBus framework is based on a model that allows dynamic interconversion between broker and worker nodes to allow scalability on demand. As shown in Figure 2, the FogBus framework allows networks to be scaled by allocating Broker privileges to worker nodes. If many worker nodes are connected to the Broker then some Worker nodes automatically get broker privileges. As shown, initially there are 3 worker nodes connected to the broker. Then, 5 more worker nodes are added. Some worker nodes convert themselves to Brokers and the number of workers per broker reduces from 8 to 2. This approach is possible only because of the platform independence property of the FogBus architecture. This solution for scalability does not hinder with Blockchain management as the blockchains of the Worker Nodes are continued and blockchain verification takes place in the regional network encompassing one Broker and it's Worker nodes. Now, due to multiple Broker nodes that the User can connect to, the tasks can be sent based on the cumulative regional workloads or using a round robin scheme. If by this approach there are several Broker nodes for the User, there can be even more levels that have nodes that act as intermediaries between the user and the broker nodes.

3.3.2. Sequence of Communication

A defined sequence of tasks and communication of data helps reduce failure rate and makes it easier to understand the flow of data through the system. Figure 2 below shows the sequence of tasks and data flow in the network.

4. Implementation

The software has been built using platforms that are supported by a wide range of devices including those used for IoT application. The main software is divided into four different components:

1. Interface Web Scripts (Based on PHP)
2. Blockchain Interface (Based on Java)
3. Aneka analysis code (Based on C# and .NET)
4. Analysis Jar file (Based on Java)
5. User Interface (Android Application, or web-based interface)

The web scripts require an Apache server to be set up in all nodes and MySQL server in master. Each of these have different roles and are distributed among different nodes. All local network and public cloud interaction is based on Blockchain to prevent hackers from accessing and manipulating data.
Interaction with the user plays a key role in obtaining the user requirements and also acts as an intermediary between the network and sensors/actuators. It is crucial for the user interface to be accessible from a wide range of devices and be friendly for interaction. Some requirements of this interface include:

- Set the architecture of the network in terms of worker nodes
- Connect to Sensors and actuators for real time data sharing
- Allow modification and extension of different components for application specific optimization
- Ability to set/modify the resource management policy
- Display/Modify/Erase data collected
- Enable or Disable Master as a worker
- Enable or Disable Aneka tasks to be spun

The current implementation of the FogBus model uses PHP based web scripts as an interface with the user and also backend interface among master and worker nodes. An abstract level depiction of the implementation is shown in Figure 4.

Java and PHP (based on HTTP) have been chosen for all local interface because they are cross platform languages. Compiled Java program runs on all platforms for which there exists a JVM (Java Virtual Machine), which includes large number of Operating systems (Windows, Linux, Mac OS X). PHP scripts can run almost everywhere (Unix, Windows, Linux, Embedded Systems, Risc, NetWare). Most embedded and IoT devices come with networking protocol stacks for HTTP communication, and can be easily installed in those that don't.
Due to the cross platform nature of the languages used to develop the FogBus framework, it can be deployed in a plethora of devices.

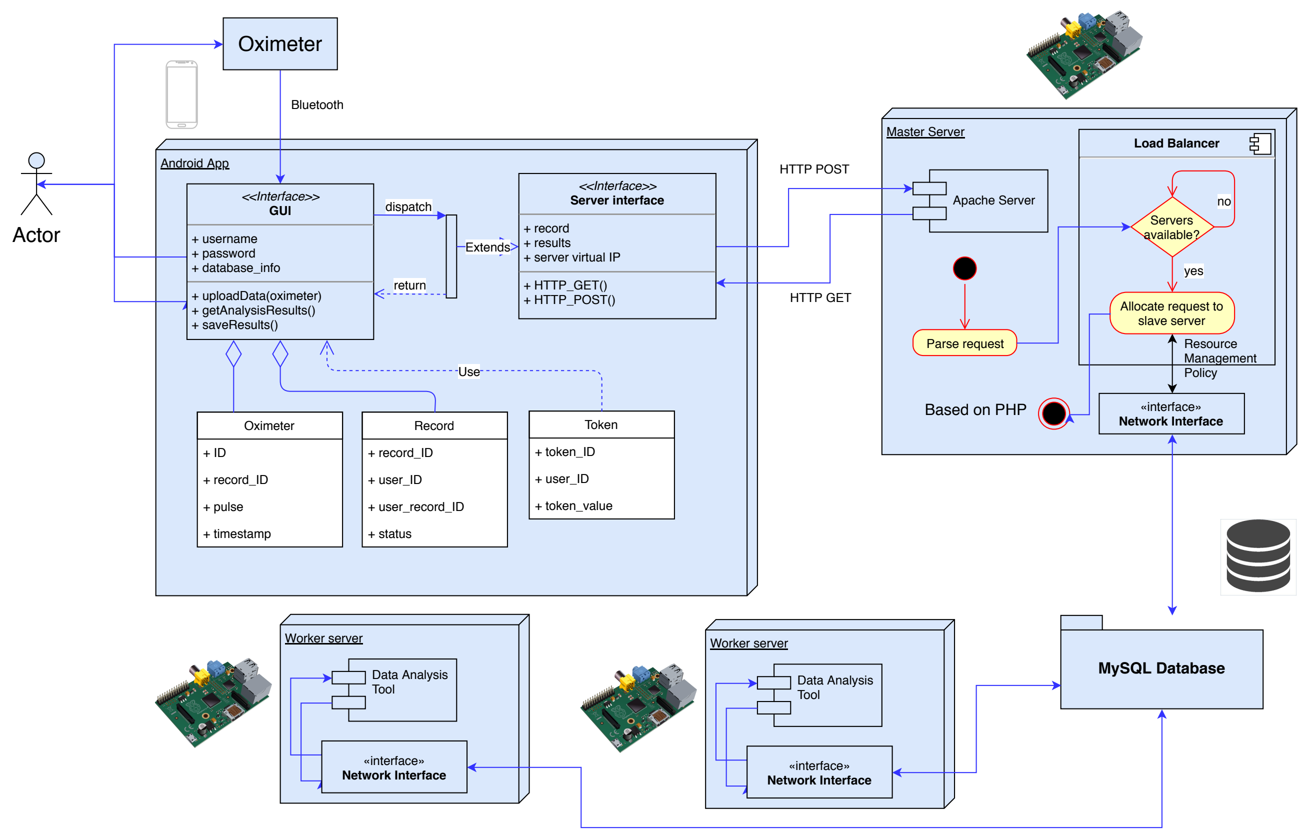


Figure 3: Sequence of communication among different components

5. Application Case Study : Sleep Apnea Analysis

The FogBus framework has been deployed and tested with a real-life application: Sleep Apnea Analysis. Sleep Apnea is a disease in which air stops flowing into the lungs for 10 seconds or longer while in sleep. This reduces the oxygen level and sensing that the patient has stopped breathing, the brain triggers to wake up just enough to breathe. In some cases, this can happen over 30 times in an hour. Many people have sleep apnea, (also known as sleep apnoea) but may not even know it. In fact, sleep apnea affects more than three in 10 men and nearly two in 10 women, so it's more common than you might think.
Sleep Apnea analysis is very difficult and cumbersome. An overnight or laboratory-based sleep study requires you to stay overnight at hospital in a sleep unit or laboratory usually consisting of a number of private, quiet, single rooms. You will sleep with sensors hooked up to various parts of your body. Physicians usually recommend this test for more complicated or difficult to diagnose cases.
The doctors use a term Apnea Hypopnea Index (AHI) and the Electrocardiogram (ECG) data to diagnose if a person is suffering from Sleep Apnea or not. Using open source algorithms, we deployed an analysis application that could analyze data from a Bluetooth enabled Pulse Oximeter and give diagnosis results based on the data.
This was divided into two parts: Data analysis application and the Android Interface with Bluetooth Oximeters.

5.1. Physical Infrastructure

The devices on which the FogBus framework was deployed:

1. Master Node: Dell XPS 13 laptop, OS: Windows 10
2. Worker Node: 2x Raspberry Pis, OS: Raspbian Stretch
3. User Interface: Android Device, OS: Android 8.0
4. Sensor: Pulse Oximeter, Data encoding UTF-8
5. Public Cloud: Microsoft Azure B1s Machine, OS: Windows Server 2016

All nodes and android device were connected using a local hotspot.

5.2. Data Analysis Software

References:
https://github.com/subrahmanyamanigadde/sleepapnea
https://github.com/monarch-initiative/sleep-apnea-clustering
The sleep apnea analysis app was developed using a combination of the above two open source repositories.
The analysis application takes the data file as input and stores results in the result file. When new data is found in the data file, it parses it and splits the string with , as a delimiter, and converts all strings to integers. Two data sets are parsed in this way: heart rate and blood oxygen level. For oxygen level, in a broad sense, the analysis is done in the following way:

- There is a dip Boolean variable, which stores whether there is a dip in oxygen level, a count which stores the number of times dip changes to true and a min which stores minimum oxygen level

- Whenever the oxygen level goes below 88, dip turns to true and stays true till oxygen level

comes above 88. This is verified with an increase in the heart rate corresponding to or with an offset with the timestamps along a dip in oxygen level.

- This count becomes the AHI (Apnea - Hypopnea Index), used to determine the disease severity
- Based on standard thresholds, the disease probability of the sleep apnea is determined

For the heart rate data, the analysis is done using the following method:

- The Minimum and Maximum heart rates are determined in the input data
- The average heart rate and average rise or fall of the heart rates is determined
- Those close to the dips in oxygen level are filtered
- Based on some thresholds the ECG diagnosis is determined

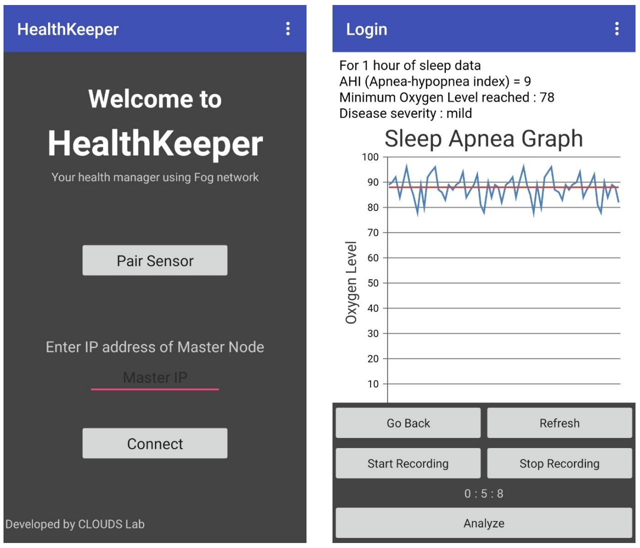
Using both the diagnosis results, disease severity of the patient can be evaluated.

5.3. *Android Interface*

References
MIT App Inventor: link
App Inventor JavaBridge: link
The android app (named HealthKeeper.apk) allows the android device to act as an intermediary between the Oximeter sensor and the Master server. Rather than sending data manually through the GET HTML form, this app records and sends data automatically. The app has been developed on an open source platform: MIT App Inventor which can be seen at this link. The source file for the android application can be found here. The application is divided into two screens namely: Welcome screen and Session screen. The Welcome screen asks user to pair the Bluetooth Oximeter with the Android device and enter the Master servers IP address.
The Session screen is the main screen that handles all interaction with the master server. An empty list is initialized and timer is reset when recording starts and is appended with data values every second. When analysis request is made, the list is sent to Master for distribution to a worker or aneka.

6. Performance Evaluation

In this section we have tested the FogBus framework with the Sleep Apnea Analysis case study on different scenarios. Each scenario was built using a combination of the following test case options:



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

Figure 4: Welcome and session screens of the Android Application

1. *Maximum Load / Constant instruction rate*: In the maximum load case, next task was sent as soon as the previous task was over to ensure that the FogBus framework is continuously on work and has no idle time. The 'Maximum Load' case can also be called as 'No Interval' case. In the constant number of instructions case, every task was sent at a gap of 5 seconds. It can also be called as 'With Interval' case. This would mean a total of 60 tasks (for a 5 minute test) but due to the overhead of master node and network propagation 60 was the maximum limit.
2. *With / Without Blockchain*: As the names of scenarios suggest the test cases were based on whether the blockchain security was enabled on not. If the blockchain is disabled there is a significant reduction in latency, network usage, and energy consumption.
3. *Fog Only / Cloud Only / Master Node Only*: This was whether the tasks were sent to only the Raspberry Pis or Azure through Aneka or Master itself performed all computation.

As the test configurations were built as a combination of these test case options, the total number of configurations were $2 \times 2 \times 3 = 12$. Each configuration was tested for 5 minutes as the following parameters were observed and recorded:

1. Number of tasks performed in 5-minute duration
2. Power consumption (Master + workers + Azure)
3. Energy per task (Master + workers + Azure)
4. Bytes sent / second by master
5. Bytes received / second by master
6. Network usage per task (Based on the bytes sent + received per task)
7. Average task latency (i.e. the time duration between when the master sends data for analysis to worker/Aneka/itself and when it gets results)
8. Master CPU Usage %
9. Master RAM Usage %

10. Master Cache Usage

The data parameters were recorded using Microsoft Performance Monitor© at the Master and Azure machines, and using NMON Performance Monitor at the Raspberry Pi. The green bars in graphs are for Fog Only case and blue bars are for Cloud Only case. Each task had two data sets: Oxygen level and Heart rate, each data set having 15 integer values.

6.1. Number of Tasks

The number of tasks performed in a constant duration of 5 minutes can be seen in the graphs for the different cases. We observe that the Number of tasks is higher in the Fog Only case than the Cloud Only case. For the Maximum Load case (i.e. tasks sent with no interval) the number of tasks that are performed depends on the Network propagation and time required by the node to perform analysis. We know that the network propagation time is lesser for fog devices, and execution time is lower for cloud. As the results show us, the latency is lower for the Fog Only case compared to the Cloud only case, and as $Latency = Network\ propagation\ time + Execution\ Time$, it is apparent that the difference in network propagation times in fog and cloud cases is more than the difference in computation time. This shows that the Fog Only case is better for performing high number of tasks.
We also see that without blockchain the framework is able to perform much higher number of tasks. This is because of lower workload and lesser amount of data to be shared among nodes.

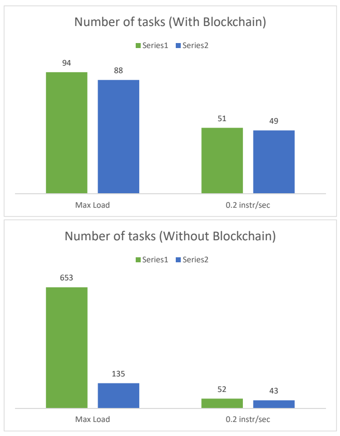


Figure 5: Number of tasks performed in 5 minutes

6.2. Latency

Latency is another critical parameter for mission critical applications like healthcare, robotics, etc. Latency is inversely proportional to the Number of tasks. The results show that the latency in Fog case is much lesser than the Cloud Only case. Also similar to the previous parameter, without blockchain the latency is lower due to signature verification and proof of work validation that needs to be done with blockchain. Results show that computing in the Fog Only case not only cuts down response time but also safeguards the application from scalability issues (leading to delayed responses) arising due to large topologies and high data-generation rates.

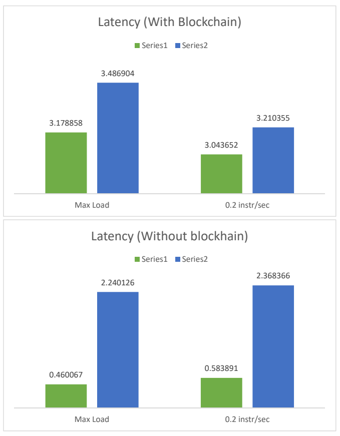


Figure 6: Latency

6.3. Network Usage

Network usage is a significant factor in such systems. It is not only important for the Network Bandwidth requirements but also the data I/O capabilities of devices. If the Network Router can not support the required I/O of the framework then it can lead to network congestion and even breakdown of the full system. Thus it is important for the framework to balance Network usage and keep is as low as possible. Network Usage is calculated as: $(Number\ of\ bytes\ sent\ /\ second\ +\ Number\ of\ bytes\ received\ /\ second)/(Number\ of\ tasks)$, where bytes sent or received are by master node. Network Usage is lower in the Fog Only case as Aneka overhead of Network communication is not present in this situation. Also, without blockchain a lot of data sharing is eliminated which is there in the blockchain case for signature verification, blockchain hash data, public keys, etc.

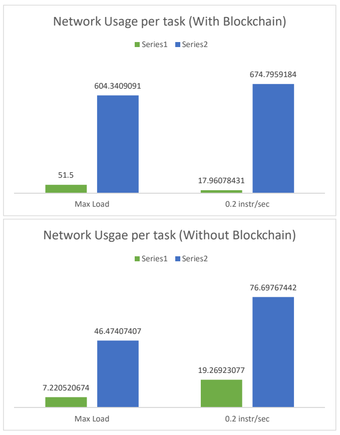


Figure 7: Network Usage

6.4. Energy

Energy consumption is a very important parameter for deploying such network of computing devices. After the installation cost, the Energy consumption plays an important role in the maintenance costs of the system. In these tests the Energy coonsumption per task was calculated as $(Average\ Power\ Consumption \times 300\ seconds)/(Number\ of\ tasks)$. Energy consumption is also lower in the Fog case which is because the cloud power consumption is much higher as compared to edge devices. Average power consumption of a Raspberry Pi is 1.5W and that of a Windows Machine is 115W which is around 77 times the former. This makes maintenance of Fog devices much easier due to low expenditure on energy consumed.

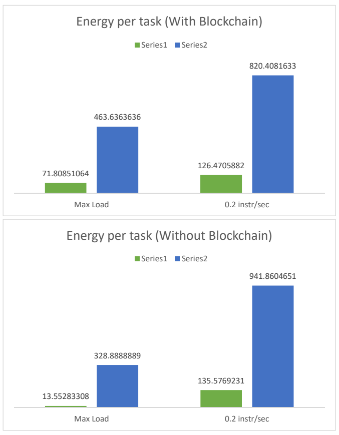


Figure 8: Energy Used per task

6.5. Master Node CPU, RAM, Cache Usage

The Figure 13 shows the Master CPU, RAM and Cache usage for the Master node for each case. Series1: Fog Only without blockchain, Series2: Cloud Only without blockchain, Series3: Fog Only with blockchain, and Series4: Cloud Only with blockchain. The parameter values are much lower in the Fog Only case due to elimination of Aneka overhead. Even without blockchain these parameters are lower due to elimination of hash creation, mining, etc tasks.

6.6. Discussion

From the results it is clear that Without blockchain case is much better in terms of latency, Network Usage, CPU Usage, etc. This is because of the much lesser load on the nodes for sharing data and maintaining blockchains.
We see that for the Sleep Apnea analysis case study, the Fog domain is much more favorable. Due to lesser latency, lower network usage, lesser energy consumption, etc. the Fog environment is much more suited for this type of applications. For application that involve high computation like calculating Fast Fourier Transform or involving Matrix Multiplication, the Fog devices may not be able to perform well in such calculation and the Cloud would be more suitable. Due to the light-weight computation requirement of Sleep Apnea Analysis application that

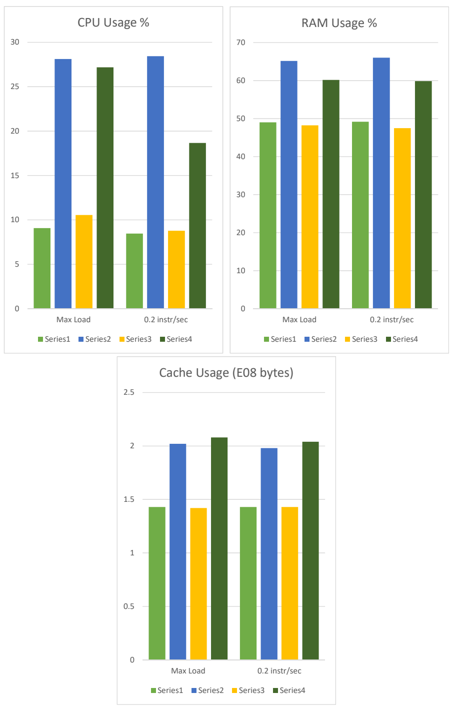


Figure 9: Master CPU, RAM and Cache Usage

we have deployed in our case study; the Fog domain is more suitable.

7. Future Work

This software forms a base for setting up Fog Computing Environment which is not OS or architecture specific. Though the framework is complete by itself and fully functions to perform real-time, sensor-based Sleep Apnea analysis and can be used for other applications as well, but has a large scope of improvement and further developments:

7.1. Load Balancing Scheme

As discussed earlier, the current load balancing scheme is naive and can be greatly improved. Currently, the load balancing scheme focuses on task distribution based on CPU load, whichever device has the least load, the task is given to it. There can be a cumulative ranking parameter which takes into account many other factors as per requirement. These factors can be and are not limited to: Network Bandwidth, Memory Load, etc. Different weights can be given to these parameters are per the scenario.

7.2. Data Integrity

Health Analysis data is important for patients as their treatment and lives depend on it, thus it is important to save this data from fraudulent manipulation or sabotage from hackers. Thus, to maintain data integrity many techniques like Blockchain can be implemented to ensure that data is secure.

7.3. Data Privacy

Some applications require data to be secure as well as kept private. The current system is prone to attacks and unwanted display of data to others. Privacy policies like encrypting the data can be used to ensure that the data cannot be seen by others.

7.4. Data Authenticity

The current system allows any device to connect to master and share or view data. This can be used by hackers to forge DDoS or similar attacks. As Fog platforms also contains low range devices with limited threshold management, such attacks even at a low scale can destroy such devices. Thus, a signature-based validation technique can be used to ensure user and data authenticity.

8. Conclusions

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.