Received 26 April 2016; | Revised 6 June 2016; | Accepted 6 June 2016

DOI: xxx/xxxx

SPECIAL ISSUE PAPER

FogBus: A simplified framework for implementing Fog, Edge and Cloud computing applications/research secured using Blockchain Technology

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

[1]Cloud Computing and Distributed Systems (CLOUDS) Laboratory Department of Computing and Information System, The University of Melbourne, Parkville, Victoria, Australia
[2]Department of Computer Science, Indian Institute of Technology, Delhi, India
[3]Department of Electrical Engineering, Indian Institute of Technology, Delhi, India

Correspondence
*Shreshth Tuli, Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, 122002, India. Email: shreshthtuli@gmail.com

Summary

Fog and Edge computing are rapidly transforming IoT and networking ecosystems. They are being used in different ways to improve the communication and simplify day-to-day tasks of public. However no flexible implementation framework/platform is available to realize such an environment. In this paper, we present "FogBus", a simplified framework for implementing Fog, Edge and Cloud computing applications/research which is secured using Blockchain Technology. The framework is built using widely used platforms like Java and web interfaces, etc. to allow the diverse range of embedded devices to be used to realize a fog environment. We describe the framework specifications and the model on which it is built. We describe various functional components of the system and its implementation details. The system has been tested on a real scenario of Sleep Apnea analysis and deployed on Raspberry Pi's for computation. Different experimental results have been shown later which act as performance evaluation metrics for the framework. In the end we also discuss various future directions and conclude our work.

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

1 | INTRODUCTION

[0]**Abbreviations:** ANA, anti-nuclear antibodies; APC, antigen-presenting cells; IRF, interferon regulatory factor

2 | RELATED WORK

3 | FRAMEWORK

3.1 | Model

FogBus is a lightweight and platform independent framework which allows an end-to-end implementation of fog, edge and cloud computing environment. It is designed to run on almost all IoT/embedded devices and support a wide range of machines in terms of their computation capabilities. A driving factor for developing this framework independent of operating system and machine architecture is to allow the diversity of embedded devices in the IoT world to be used in the network. The framework integrates all edge devices (using a lightweight balancing framework for small-scale computation) and also cloud/other heavy devices (using the Aneka framework for heavy computation) and provides an end to end solution for integrating computation at different levels to provide better latency (by allocating simple and small-scale work to edge nodes) and high computation power (by allocating heavy computation work to cloud), which in effect to the end user feels like high computation with low latency. This is the fundamental ideology on which fog computing is based.
The framework is secured using Blockchain technology and digital signature management to ensure data remains secure and can not be tampered by hackers. Due to edge devices being fragile, the Blockchain structure had to ensure that critical data is not stolen and/or gets leaked when an edge device is compromised.

The model is developed to provide a framework the aims of which include:

1. Allow input of data from sensors and output of information/actions to end user/actuators
2. Provide an end-to-end and seamless communication between different fog and cloud resources
3. Provide a platform to easily form an integrated system that can use this interaction to provide services to the end user
4. Implement resource management policies and allow easy extension/implementation of such policies for efficient load distribution
5. Set up a friendly interface between the framework and end user
6. Keep all data secured, and ensure data sources are legit and existing data is not tampered by hackers

3.2 | Physical Components

In the FogBus framework, all devices interact at different levels with different devices. The overall model is based on the hierarchal Master-Slave design where the Master node receives data from sensor for analysis/computation. The Master node then sends this data as a task to different workers: Local worker, Private Cloud or Public Cloud. Therefore, the core components of network include Master Node, Worker Nodes, Sensors and Cloud. All other components including gateways, switches and other interface devices 'glue' the core components together.
The core physical components in the FogBus framework include:

1. *Master Node*: It is the main physical computer which distributes tasks among the Worker nodes based on some load balancing scheme. It directly controls the Worker Nodes.
2. *Worker Nodes*: They are working machines in the form of dedicated computers or Virtual Machines(VM) that take task details from the Master Node, perform the tasks and return results.
3. *IoT Sensors*: Devices which measure physical properties of their surroundings, record the data and send to the Master Node through an interface.
4. *Cloud*: A network of physical or virtual servers that store data and act as computational resource spread across large number of data centers across the globe. Three main types of Clouds exist:
    - *Private Cloud*: Private cloud is cloud infrastructure operated solely for a single organization, whether managed internally or by a third-party, and hosted either internally or externally.
    - *Public Cloud*: A cloud is called a "public cloud" when the services are rendered over a network that is open for public use. Public cloud services may be free.

- *Hybrid Cloud*: Hybrid cloud is a composition of two or more clouds (private, community or public) that remain distinct entities but are bound together, offering the benefits of multiple deployment models

Other components that help in the communication between the core components and help in functioning of the network include:

1. *Network Router*: A Router is a network device that directs data packets in the network to the required source destination which is characterized by the IP (Internet Protocol) address of the destination. It allows communication among the Master node, Worker nodes and Cloud.
2. *Database*: A structured set of data that is held in one computer or distributed among many computers so that all nodes having access to the database can share data.
3. *Switch*: A switch is a bridge that connects together many devices. They mostly act as network connection point for nodes.
4. *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.

3.3 | Network Structure

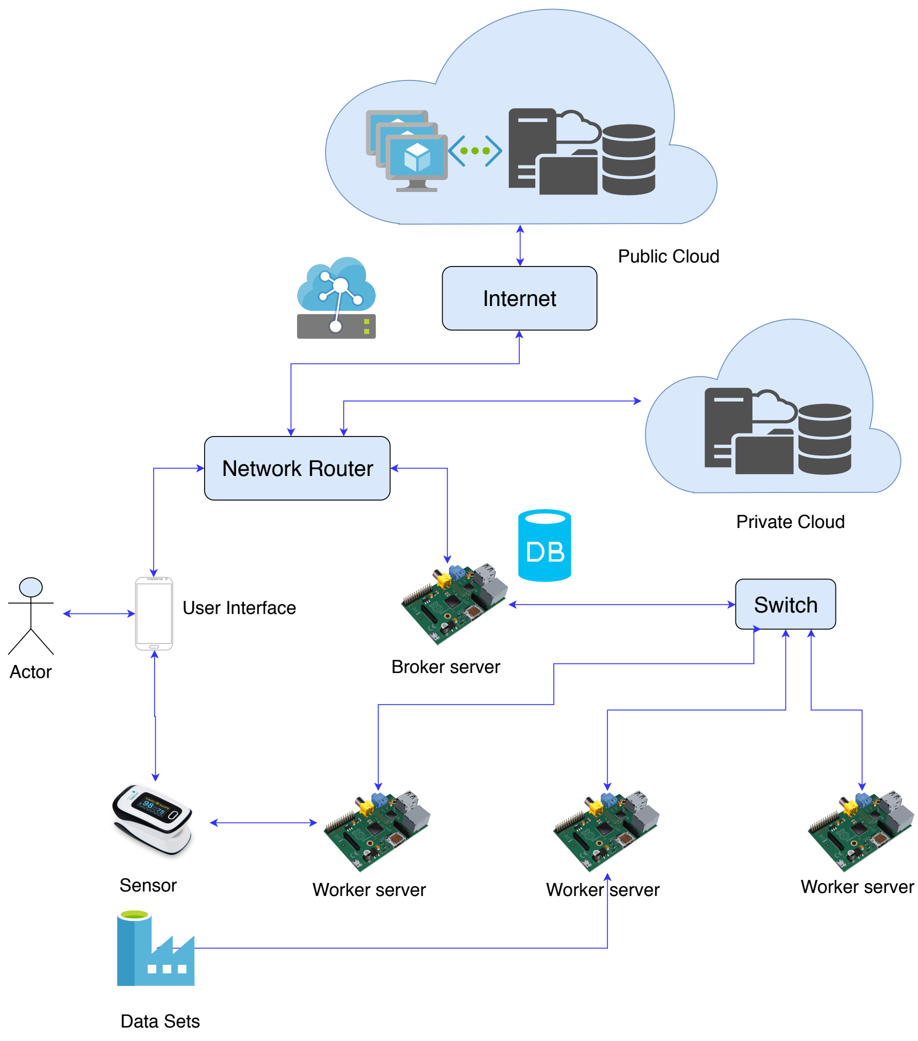


FIGURE 1 Network level structure of the FogBus framework

Figure 1 depicts how different physical components, as described in section 3.2, are connected in the FogBus model. The User connects hos/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.

3.4 | 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.

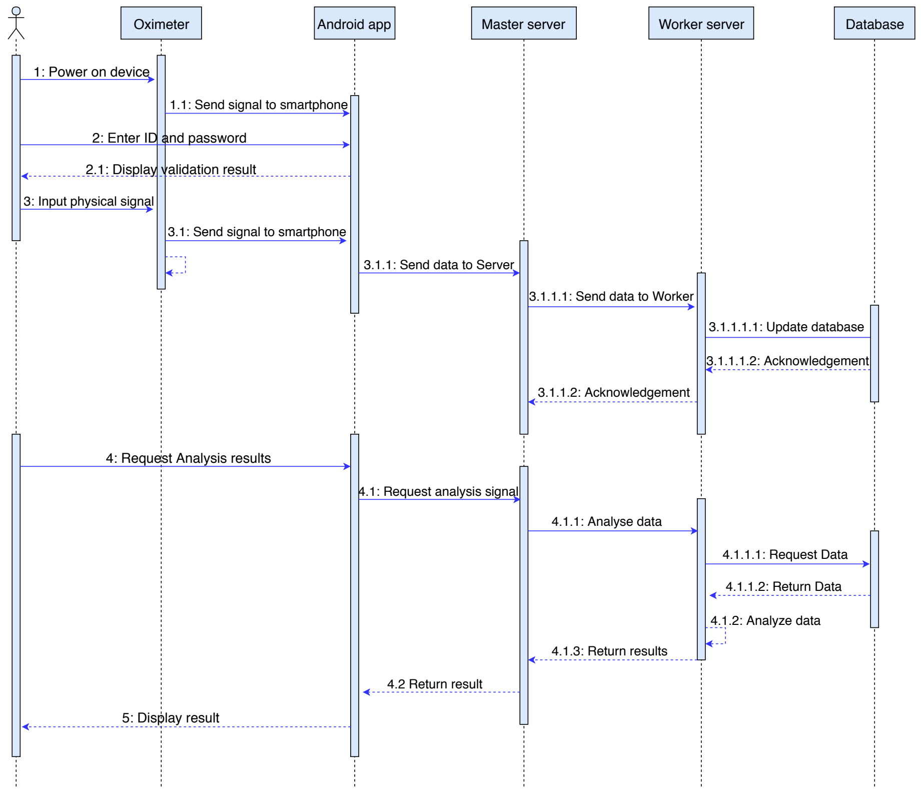


FIGURE 2 Sequence of communication among different components

4 | FUNCTIONAL COMPONENTS

4.1 | Resource Manager

As devices at different levels of the network, as shown in the 'Network Level Model', offer range of computational capabilities and latencies, it is important to distribute work accordingly to ensure optimum QoS. The devices at the 'edge' of the network provide very low latency due to close proximity with the user, sensors and actuators but have tendency to have significantly low computation power and thus higher execution times. The devices far away from users like those on the cloud provide significantly higher latency but due to their much higher computation capabilities provide much lesser execution time. There is thus a trade-off between network propagation latency and computation latency. Depending on different scenarios, different resource management techniques are used.

There is scope of considerable reduction of latency caused due to propagation of data if low-computation workload is allocated to edge devices. Even though their execution time would be higher but the critical factor here would be the time caused by data transfer over the network. On the other hand, significant reduction in time can be achieved if high-computation workload is sent to more powerful machines like cloud because in such tasks edge devices might lead to deadlocks and cause the whole system to collapse.

The Resource manager of the model should decide the following:

1. Allocate task to a local worker node or not: As the latency offered by local machines is much lower in terms of the network propagation of data this is a crucial decision to be taken by the master node. Sending to cloud would require considerable overhead and might not be suitable for real-time and mission critical tasks like health-care, robotics, etc.
2. Allocate task to itself or not: Some small tasks can be performed by the master itself. As the master deals with the load balancing and other tasks like maintaining integrity of data, it might be that the master node is itself busy. Even then some small tasks may arise on rare occasions which require immediate results. At such stages, the master may decide to perform it itself to eliminate data propagation over the network and other authentication and handshaking protocols.

It is a design decision in the implementation strategy how these decisions are taken. Some parameters that are important for taking such decisions include:

1. *Quality of results*: Each application has its own quality thresholds. As the required quality of results increases, more complex algorithms may be required. Sometimes the algorithm remains same but more number of iterations are required to reach the required confidence threshold. This has a direct impact on the computation difficulty and how well and fast a machine is able to perform repetitive tasks.
2. *Deadline*: The time in which the results are required plays a crucial role in determining the load balancing scheme. As hinted earlier, devices at different levels have different execution times and data propagation times. Based on data flow rate and application runtime it is important to balance tasks among different levels of devices.
3. *Frequency of data sharing*: Closely related to point 2, some tasks might require small amount of data to be analyzed but with high frequency for example —–. In such cases execution time needs to be very low and network bandwidth very high. The device themselves should allow high I/O data flow.
4. *Computation Complexity of Application*: The order of complexity of application determines the number of task request to be sent to cloud. For very high complexity tasks it might me suitable to only use cloud resources to prevent bottlenecking of the software because of low computational capabilities of edge nodes.
5. *Cost of Computation*: Different levels have different currency for computation. Devices over the cloud are metered and are charged based on time of usage. Embedded devices in IoT applications are charged based on the initial physical component cost and then the cost of energy expenditure for functioning of these devices. By normalizing these different currencies using conversion factors, one can determine the cumulative cost for running the framework per unit time/task/user. Inherent to the model requirements, but depending on the scenario, this cost needs to be minimized.

Different load balancing algorithms can be implemented that specialize in improving some of these parameters based on user needs. A weighted sum of quantifiable factor of each of these parameters and the weights themselves would determine which balancing technique should be used for a specific situation. The *Session Manager* in the Master Node with the help of Workers' *Resource Monitors* handle the resource management in the FogBus framework. See *Implementation* for more details.

4.2 | Blockchain and Digital Signature

Maintaining integrity of data and ensuring that data is not sent by an unregistered source is very important for the credibility of the system. This is extremely important in systems that maintain real time medical records, financial transactions and other secured platforms.
For the integrity of data and prevention from tampering of existing data the model uses Blockchain Technology. Technically, blockchain is a suite of distributed ledger technologies that can be programmed to record and track anything of value.
Whenever new data is received by the Master node, it packs it to form a "block". This block hash a SHA256 hash value which is created using the data, block index and a nonce value. The Master node, whenever creating a block mines the block to create a "proof-of-work" which allows the hash to have a particular form/pattern. The Master also creates a random public/private key pair that allow to form a unique signature with the original data. The private key is not shared with the worker node and only data, signature and public key are shared. With this the Worker node is able to verify that the data is from a legitimate source as it saves the public key of the Master node. If any other key is used, that data is rejected. Also, the public/private key pair is kept dynamic per block generation to prevent hackers to generate private key using brute force techniques. Even if one is able to generate a private key for a transaction, it would change for the next and thus become hack proof.
The user is also given ability to track the data/block flow through the worker nodes by displaying the latest hashes of the blockchain copy at each node. This ensures that the user is able to see and automatically take action on which nodes are more prone to attack and/or are being targeted by hackers.
Thus:

1. Dynamic Public and Private Key Generation
2. Digital Signature Verification
3. Blockchain maintenance and validation
4. Proof of work sharing and verification
5. Automatic majority chain deployed in minority workers
6. User displayed the latest hashes of each node

Points 1 and 2 ensure that the data recorded is from genuine source and points 3, 4 and 5 ensure data integrity. The *Master Interface* and *Worker Interface* modules in respective nodes handle the blockchain and signature verification in the FogBus framework. See *Implementation* for more details.

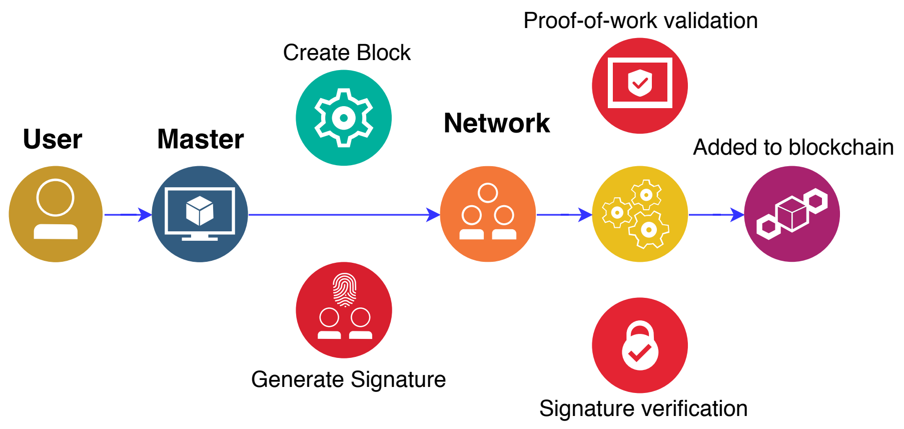


FIGURE 3 Simple Load balancing scheme implemented in code

4.3 | Data Input/Output

There exist several ways of interacting and sharing data from sensors and among fog nodes. It is crucial that this data sharing mechanism is failsafe in terms of data theft and fraudulent manipulation. The streams of data can also be dedicated. Another important part is that the protocol followed for communication among fog devices should be supported by all nodes, complex and rarely used encryption standards may restrict diversity of devices that can be used.
The sharing of data with sensors and actuators needs to be as close to real time as possible. For mission critical application proper data transfer frequencies are important. For the case of static transfer frequencies, very high frequencies can lead to computation lag from the worker nodes, very low frequency can cause reduction in the Quality-of-service (QoS).
The current model uses HTTP protocol for communication between different nodes in the local network and the Aneka's protocols for task distribution to cloud and other devices. All data is currently transferred in plain text using the HTTP REST APIs specifically GET and POST. The *User Interface* and the *Session Manager* collectively handle the Data I/O in the FogBus framework. See *Implementation* for more details.

4.4 | Computation over Data

Another important criteria for model development is that it is generic in terms of the service it provides. In the IoT world there are several applications in which Fog, Edge and Cloud Computing can be used, and there are several others that haven't even been thought of. It is important that the frame work allows services to be used that can be changed, modified and/or extended easily. The computation/analysis requirement of the service also affects the Resource management policy and data transfer rate requirements. Based on different services and end-user requirements, performance metrics can be designed and the framework can be optimized accordingly.
The model uses Java runtime environment for data analysis due to it's ability to be run in diverse machines. The *Analysis Application* in the Workers does the computation over data. See *Implementation* for more details.

4.5 | Data Storage

Many applications require data to be stored for future use. This data being shared among multiple nodes is prone to attacks. It is thus important to implement strategies/techniques that keep data resilient and prevent it from tampering. Not only keeping saved data secure is important, but the methods used for data extraction and sharing among nodes is also a concern. The *Network* and *Session Managers* handle the data storage in the FogBus framework. See *Implementation* for more details.

5 | 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.

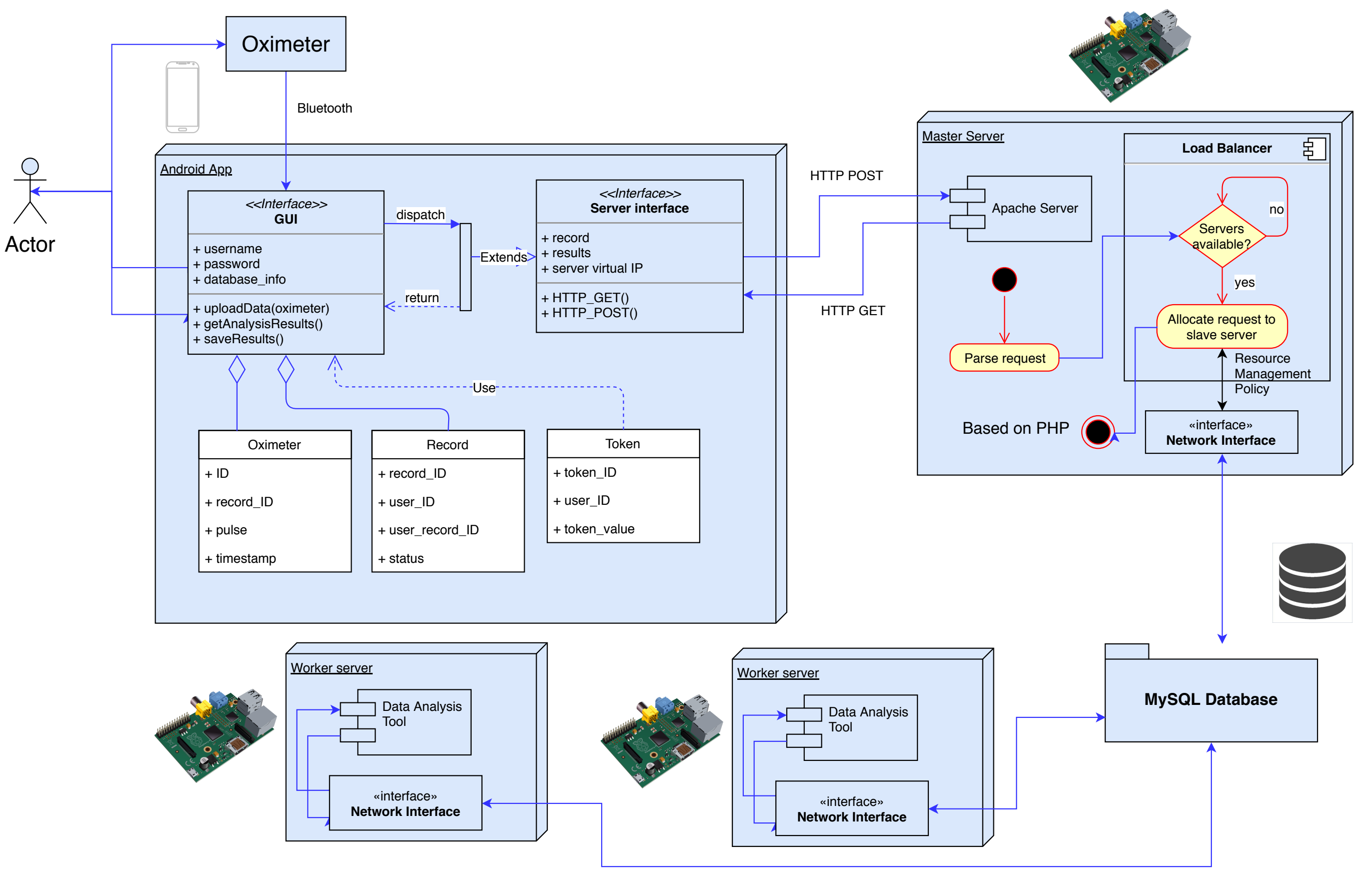


FIGURE 4 Sequence of communication among different components

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.

5.1 | Master Node

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

5.1.1 | Login Verification Module

This module takes the user login credentials and verifies them with the database. An HTML form allows users to input login information including username and password is displayed first. This form takes in the login information and sends it to Master node using the POST method. The script then checks the input information with the MySQL Database. Using the *mysqli_connect* command, the database name and table name are verified. After the form, the database and table connection success are shown on the web page. The login information is checked using the SQL query to find entry with the given details. If the number of returned rows is greater than 0 then the login is successful and page navigates to Navigation Module. If the login details are not found in the table then a prompt is displayed to allow re-entering the login details.

5.1.2 | Navigation Module

This acts as a menu or navigation page for the user and allows user to go to either the Network Manager or the Session Manager.

5.1.3 | Network Manager

This module manages all the worker IPs for the Session Manager. It maintains the configuration file in which every line contains one IP address of a worker. The configuration file also holds whether Master and Aneka could be given the task of analysis or not. A reset command allow the configuration file to be reset to default state where there are no registered worker IP addresses and both Master and Aneka are enabled for task processing. Whenever a new IP address is set it is appended to the configuration file. 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 master's copy of the executable and overwrites its own with it. This Manager also lets user configure other worker nodes automatically by sending it's IP address and latest public key to other nodes for updating. The configuration files of the worker nodes is updated and manual configuration update is not required.

5.1.4 | Session manager

This module performs the most of the software functionalities.. The user can manually input the data for send data via the GET request to the master node. After inputting the data, if the "Analyze" button is clicked, then the content of configuration file is parsed to obtain list of worker IP addresses.
Variables $toMaster and $toAneka store if the task is to be given to the Master or Aneka or not. This module also handles the "Fail over" and "Load Balancing" schemes of the software. The variable $ips is an array of the IP addresses and $loads is their corresponding loads. 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. If any HTTP request for load returns FALSE then variable $my_var is set to 100, and error message is displayed on the screen. Setting load to 100 has the effect that this node would never get the task and hence removed from load balancing IPs. 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.
Using a for loop, the $loads array is populated by accessing the Resource Monitor of the workers corresponding to the IP address. If the load of any worker is < 80%, then the variables $toMaster and $toAneka are set to false. In effect, when master is enabled as worker, this sets $toMaster to true only when all worker nodes have more that 80% load. Same is the case for Aneka container. If the task is not given to the master or Aneka, i.e. $toMaster and $toAneka are false, then the IP of the worker with minimum load gets the task. 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 5. 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.

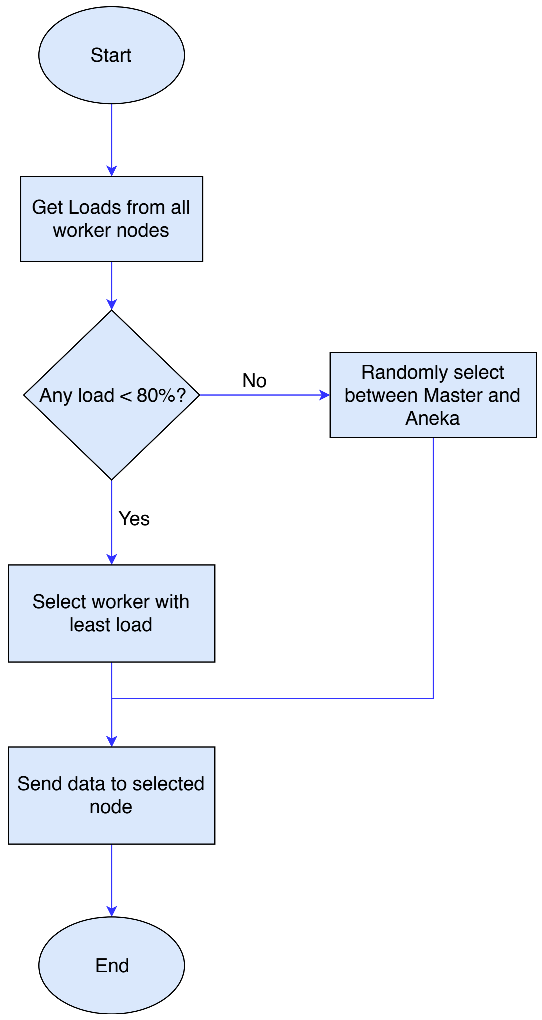

Start
Get Loads from all worker nodes
Any load < 80%?
No
Randomly select between Master and Aneka
Yes
Select worker with least load
Send data to selected node
End

FIGURE 5 Simple Load balancing scheme implemented in code

5.1.5 | Master Interface

This module acts as the blockchain interface for the Master node. It forms new blocks and generates random public/private key value pairs and shares with worker nodes. It also maintains its own blockchain and sends data, hash values and proof-of-work to other nodes for verification. All hashes are generated using SHA256 algorithm and data transferred in Base64 encoding. Each block also maintains it's timestamp of creation for backtracking and validation of chronological sequence of block creation in the chain. This helps in checking if a block created later in time has not been inserted in between in fraudulent manipulation. The received results from the worker node are displayed on the webpage and a cumulative graph shows all the data that has been recorded since the user was registered.

5.2 | Aneka

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

5.2.1 | Aneka Interface

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.

5.2.2 | Aneka software

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 has been implemented to ensure data integrity and secure the data and results from hackers. Whenever a new thread or task is spun, the proof of work is calculated by the master node and is sent to other nodes for verification. If verification passes then the block of data is added to the chain, otherwise the data is discarded. Each time new task is sent, the chain is validated by checking the correctness of the last block's hash value and matching the previous hash value of the last block with the hash value of second last block.
Digital Signature management has also been implemented so that data from unknown sources can not be added to the blockchain, thus ensuring no outside and unwanted manipulation can be done. Each Aneka node is given a public and private key. The signatures are formed using private key and verified using public keys.

5.3 | Worker Node

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

5.3.1 | 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.

5.3.2 | Network Manager

The worker's 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.

5.4 | 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.

5.4.1 | 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.

5.4.2 | 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. Thus, a hacker can not add new data or tamper with existing data due to the following argument: the hacker can manipulate with data only in the following ways:

1. Add new data (and different from last) For adding different data from the last one the hacker would need to develop a tuple of data, public key and signature that pass the signature verification test and match with master public key. Two sub cases arise:
   - He/She can not use the same public key as the last sent data to develop a signature, he/she would require the private key of the master which is hidden. To reverse engineer the private key of the master using brute-force requires an unfeasible amount of computation power. Thus, the signature verification step ensures that this case cancels out.
   - He/She can not use some random private key as then the public key won't match with that of master. The step that matches the public key with that of the master node ensures that this case also cancels out.
2. Add new data (same as last)
   - For adding new data which is same as last the hacker can easily keep using the same tuple of public key, data and signature. This is very much probable for causing deadlocks and DDoS attacks, but due to the check performed by *Worker Interface* which rejects repeated blocks, this is not possible (the *Master Interface* ensures that every new block has at least one of public key, signature or data different from previous block)
   - Hacker cannot use different public key and signature for the same data as then the Master public key check would reject it.

   3. Tamper existing data
   When the validation of blockchain is performed the following checks at the *Worker Interface* report data tampering:
   - 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 data is tampered with, the one of these checks would report the manipulation as the hash of some block would not match pattern if not mined. Even if that particular block is mined, then the previous hash value of the next block won't match the hash of this block, so each and every block of the chain after this block needs to be mined. This is a very computation intensive task. Moreover, if the hacker is able to mine all blocks in the chain, in the next transaction the last hash value of this chain would be different from others and the chain would be rejected. So, the hacker would have to take control and mine simultaneously more than 50% of the network blockchains which is not possible.

6 | 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

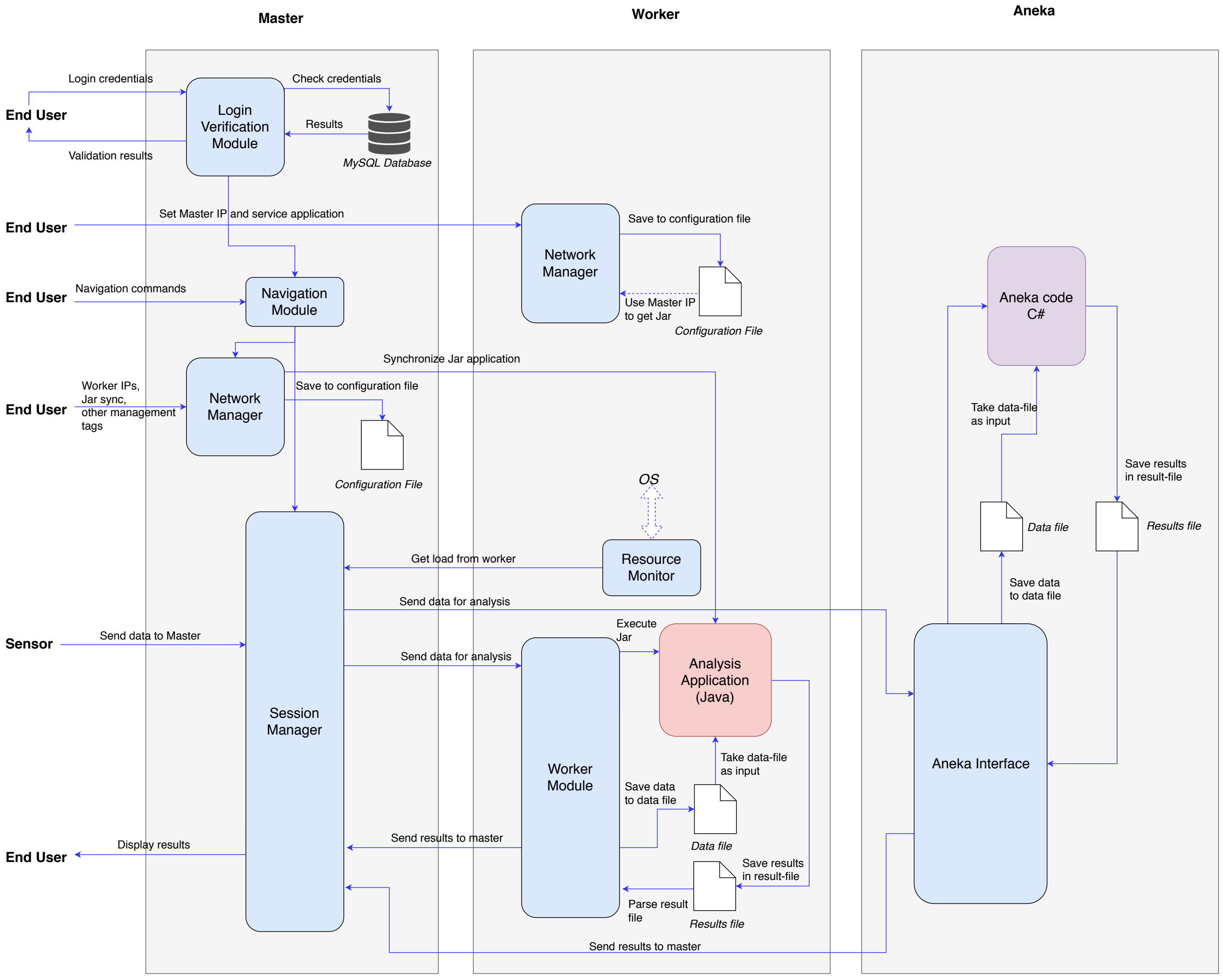


FIGURE 6 Complete working explanation of the analysis tool

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.

6.1 | 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.

6.2 | 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:

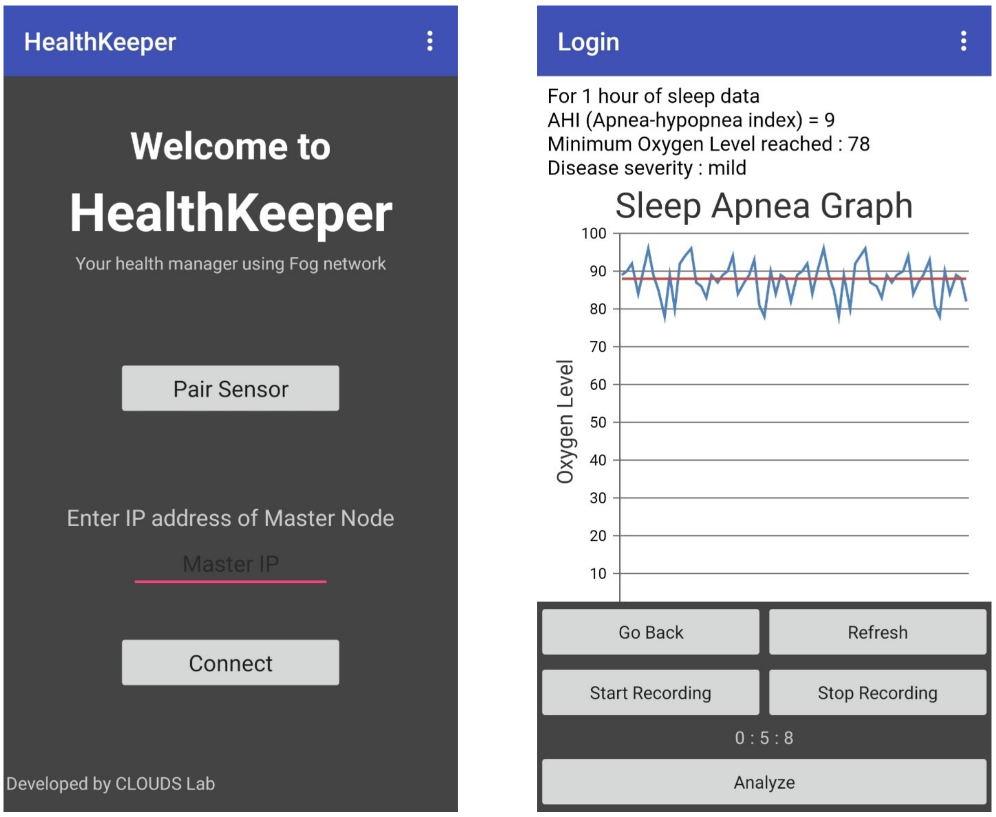
- The Minimum and Maximum heart rates are determined in the input data
- The average heart rate and average rise or fall of the heart rates is determined
- Those close to the dips in oxygen level are filtered
- Based on some thresholds the ECG diagnosis is determined

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

6.3 | Android Interface

References
MIT App Inventor: http://appinventor.mit.edu/explore/front.html
App Inventor JavaBridge: http://www.appinventor.org/jBridgeIntro
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 applicatoin 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 server's IP address.
The *masterip* is the global variable which stores the IP address entered by the user. The list global variable stores a list of *masterip* and Bluetooth device address. The *PairSensor* is a list picker to select from the available bluetooth devices. After picking, the BluetoothClient connects to the selected address and displays it on the screen. When the "Connect" button is clicked, the list objects are appended and sent to the Session screen. Based on the make of different oximeters, the corresponding service and characteristic UUIDs are used.
The Session screen is the main screen that handles all interaction with the master server. The blocks below show the variable initializations. The data variable stores the data received from sensor in list form, *loop* variable is a boolean for timer, and *url*

variable stores the web address of the master node's index page.
The "Refresh" button connects the "WebViewer" to the url. When the "Start Recording" button is clicked, the timer is reset, the BluetoothClient connects to the address and starts collecting data from the sensor till the "Stop Recording" button is clicked. When the Stop Recording button is clicked, the timer is stopped. When Analyze button is clicked, the data is submitted for analysis and results displayed on the WebViewer. The analysis function is shown in Figure 6, which Connects the WebViewer to the Session Manager with GET request: analyze value set, which ensure that data is submitted for analysis.



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

FIGURE 7 Welcome and session screens of the Android Application

7 | 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:

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.

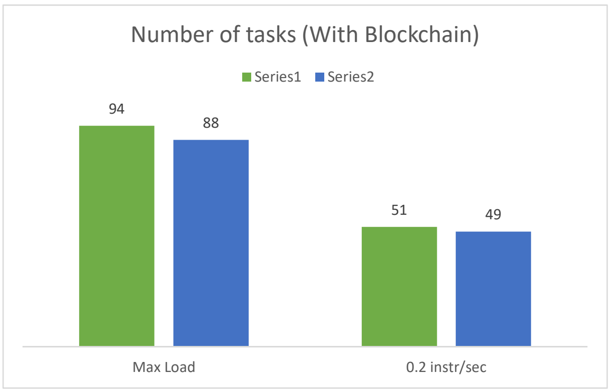
As the test configurations were built as a combination of these test case options, the total number of configurations were $2 \times 2 \times 3 = 12$. Each configuration was tested for 5 minutes as the following parameters were observed and recorded:

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.

7.1 | 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. As the results show us, it is apparent that the network propagation difference in fog and cloud cases (lesser in the former case) is more than the difference in computation time (lesser in the latter case). 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.



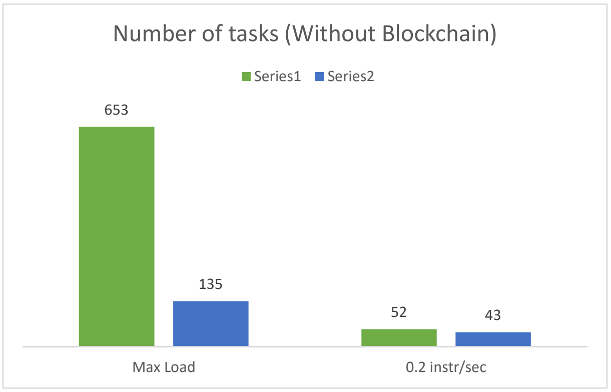


FIGURE 8 Number of tasks performed in 5 minutes

7.2 | 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.

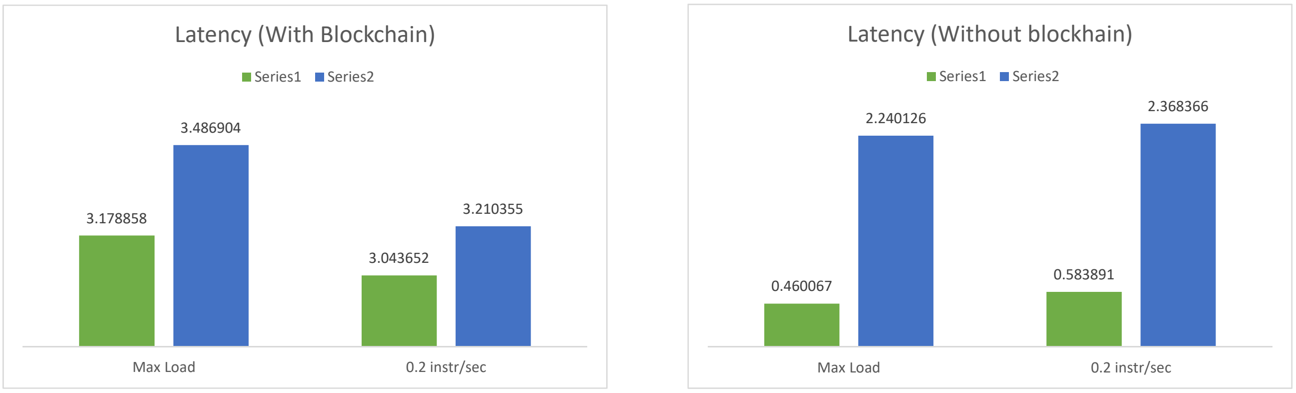


FIGURE 9 Latency

7.3 | 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 case. 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.

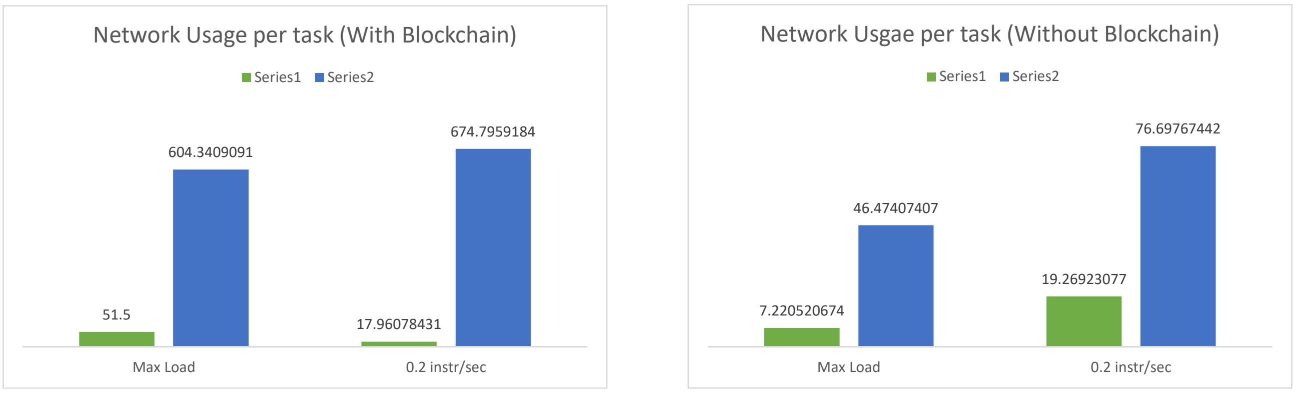


FIGURE 10 Network Usage

7.4 | 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.

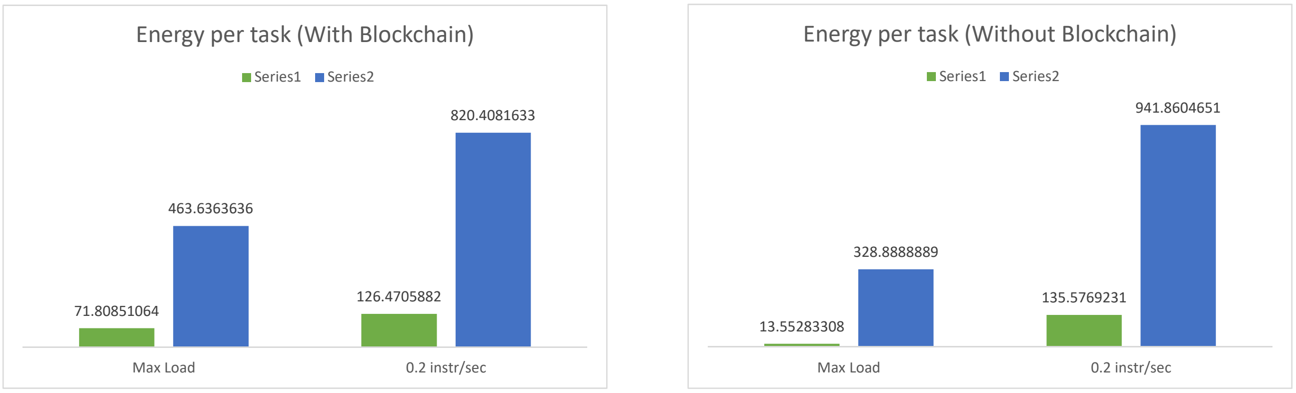


FIGURE 11 Energy Used per task

7.5 | Master Node CPU, RAM, Cache Usage

The Figure 12 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.

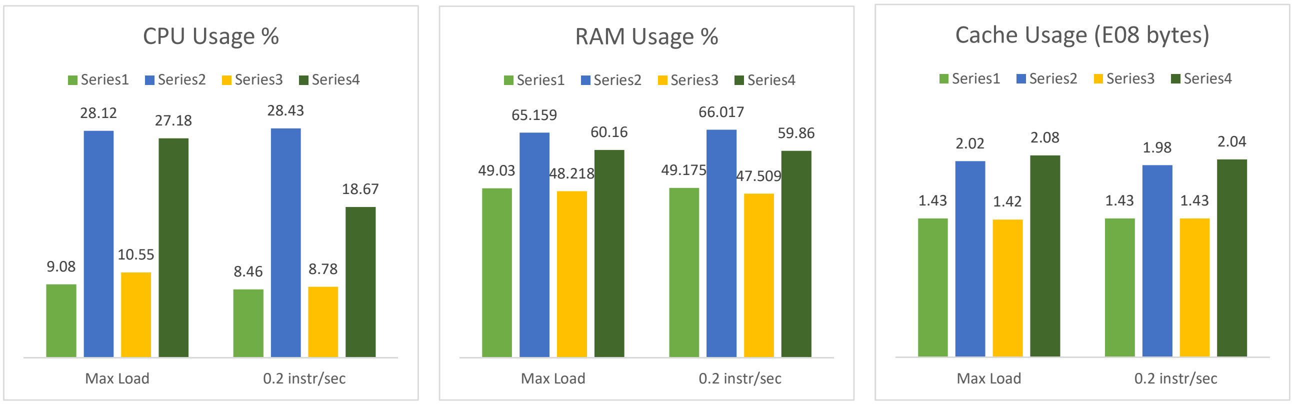


FIGURE 12 Master CPU Usage %

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

8 | 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:

8.1 | 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.

8.2 | 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.

8.3 | 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.

8.4 | 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.

9 | CONCLUSIONS

10 | SOFTWARE AVAILABILITY

References

How to cite this article: Shreshth Tuli, Shikhar Tuli, Redowan Mahmud, Rajkumar Buyya (2018), FogBus: A simplified framework for implementing Fog, Edge and Cloud computing applications/research secured using Blockchain Technology, *Softw Pract Exper.*, *2018;00:1–6*.