FOG BASED SERVICE ORIENTED IOT INFRASTRUCTURE

SUBMISSION DATE: DECEMBER 14, 2016

Tasnia Ashrafi Heya (13101200)
Department of Computer Science and Engineering

Sayed Erfan Arefin (13101223)
Department of Computer Science and Engineering

Kowshik Dipta Das Joy (13101206)
Department of Computer Science and Engineering

Md. Arshad Hossain (13101183)
Department of Computer Science and Engineering

Supervisor:

Amitabha Chakrabarty, Ph.D
Assistant Professor
Department of Computer Science and Engineering

Declaration

We, hereby declare that this thesis is based on results we have found ourselves. Materials of work from researchers conducted by others are mentioned in references.

Signature of Supervisor

Amitabha Chakrabarty, Ph.D
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Signature of Authors

Tasnia Ashrafi Heya
(13101200)

Sayed Erfan Arefin
(13101223)

Kowshik Dipta Das Joy
(13101206)

Md. Arshad Hossain
(13101183)

ABSTRACT

The Internet of Things(IoT) can be defined as a network connectivity bridge between people, systems and physical world. With the increasing number of IoT devices and networks, dealing with enormous number of data efficiently is becoming more and more challenging for the present infrastructure which is a very big matter of concern. In this paper, we depicted the current infrastructure and proposed another model of IoT infrastructure to surpass the difficulties of the existing infrastructure, which will be a coordinated effort of Fog computing amalgamation with Machine-to-Machine(M2M) intelligent communication protocol followed by incorporation of Service Oriented Architecture(SOA) and finally integration of Agent based SOA. This model will have the capacity to exchange data by breaking down dependably and methodically with low latency, less bandwidth, heterogeneity in less measure of time maintaining the Quality of Service(QoS) precisely.

Acknowledgement

We would like to express our gratitude to the Almighty who gave us the opportunity, determination, strength and intelligence to complete our work.

A very big and humble thank you goes to our supervisor Dr. Amitabha Chakrabarty who has constantly believed in us and has been there for us through thick and thins of the thesis and continuously pushed us to complete our work in time. We are fortunate and grateful to be able to work under his supervision.

Lastly, our gratitude goes to the faculty members of the Department of Computer Science and Engineering, BRAC University from whom we gained the knowledge, appreciation and help for the completion of our thesis work.

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

In introduction we will describe about the Internet of Things and the problems it may face in the near future. We also talked about the infrastructures available and gave an overview of what features we have in our infrastructure. In later parts we have described about our infrastructure.

1.1 Introduction

As per measurements in 2016 number of devices associated with Internet achieved 22.9 billion and it is evaluated that this sum will in any event twofold by 2020[1]. In view of this development rate, this number will cross trillion sooner rather than later. These devices will be in charge of creating more than quintillions of data which will be transmitted through the network. Because of discrete development and imprecise structure, taking care of such measure of data will involve challenge for present infrastructure.

IoT does not take after a particular infrastructure yet as Internet of Things is a developing field and numerous compositional models have been proposed by analysts which are very nearly getting actualized. These delivered effective results within specific segments of IoT. In spite of that we still lack a complete functional model by which we can effectuate in real world. M2M communication protocol, SOA composition model, Agent based SOA, Fog computing these are some individual design for various contextual connections of IoT. Yet, each of this architecture independently lack behind on a few prospects on which other architecture can

[1] Statistica, http://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide

perform better. Therefore, we can't think of them as a complete and impeccably workable infrastructure which IoT requires for such an enormous number of data.

In this paper, we are proposing an infrastructure which will be a combination of Fog computing merging with modified Machine-to-Machine(M2M) intelligent communication protocol emanating integration of the service composition model and Agent based composition which are built upon Service Oriented Architecture(SOA). This model will be able to transfer data by analyzing reliably and systematically with low latency, less bandwidth, heterogeneity in less amount of time maintaining the Quality of Service(QoS) appropriately.

1.2 Objective

- Optimizing the concept of cloud and create a more appropriate way to deliver data through a geographical distributed infrastructure.
- Have a better integration and contribution in the 'Internet of Things' (IoT).
- Enhance the cloud model with Fog services for better data process.
- To improve on latency and data access for end-devices by bringing the data computation on a middleware network instead of an endpoint.
- Creating easy geographical distribution for faster data analytics and real-time data processing on a large scale.
- Improving user and administration performance by creating adaptive services for benefitting the end-users.
- To have a firm grasp to see what users are using and what services they use to provide them with relevant and instant service in the future.

- Implement the solution in such a way that every other business can integrate and adopt with the system.

1.3 Motivation

Our main motivation throughout this thesis was to have a contribution in the emerging sector of Internet of Things (IoT). We wanted to implement a solution by thinking a few years ahead of us about the rising of technology and the increasingly amount of data that is to be processed in order to make the Internet of Things a reality. The topic of Internet of Things (IoT) rose in our minds after we saw the recent updates and innovations for making objects to sense and reply with the help of sensors and various IoT suites. We then realized the integration of these everyday objects into the Internet is a huge step and this will surely have a big impact on the network as there will soon be millions of these 'Things' everywhere. Therefore, then and there we were convinced to do research and work for the distribution of data for the Internet of Things so that we could lessen the huge volumes of data traffic for the cloud to make a faster and reliable infrastructure.

1.4 Thesis Outline

Chapter 1 is the introduction of thesis. We have discussed our motivation and objectives.

Chapter 2 is the background study that covers the literature review and all the research work we have done and projected the basic real life applications of Fog computing.

Chapter 3 is where we have proposed a new and better infrastructure for Internet of Things which complements almost all the drawbacks of the traditional infrastructure.

Chapter 4 is the implementation section where we described all the algorithms and flowcharts we have built to prove the validity of our proposed infrastructure and compared our algorithms with the traditional infrastructure.

Chapter 5 is the results of our algorithms projected through graphs and result comparison with the present cloud computing model.

Chapter 6 contains conclusion and discussion about the future aspects of our thesis and research.

Chapter 2
LITERATURE REVIEW

Considering the ad-hoc network, increasing usage of network is making people habituated of social communication applications with required privacy controlling opportunities. These applications are restricting users in a fixed networking area combined with fixed components by communicating through nodes inside a particular ad-hoc wireless network instead of communicating through central server [2]. This wireless ad-hoc network can be designated as an appropriate solution for the data traffic problem of today's fiber optic based networks [6, 19]. The communication between different nodes in a particular network and resolving their next destination network are confined within a specific group where the source may not consist of any information about those groups. Dividing all nodes in two categories, (i) small size with less popularity and (ii) with many social contacts and more popularity and to divide bandwidth in equal parts to utilize network resources for better performance the traffic through each cell can be routed assuming three different scenarios (i) Nodes in transmission mode, (ii) Nodes in relay mode and (iii) Node is in receive mode where each nodes transmit just one flow at a time and carries traffic within maximum supportable traffic [2]. But processing these data and application processing in cloud is very time consuming for large data, sending every bit of data over cloud channels causes problem of bandwidth at remote places, depending servers are located which causes slow response time and scalability. Whereas, location awareness with less bandwidth, low latency and geo-distribution is one of the core requirement of IoT which is not entirely possible to handle through traditional cloud computing by following this structure.

Cloud computing having a significant ramification, is a riotous technology. Despite everything, it has a few issues in regards to service-level agreements (SLA) with security, protection and energy efficiency. Cloud uses three conveyance models

Software as a Service(SaaS), Platform as a Service(PaaS) and Infrastructure as a Service(IaaS) with various level of security conditions [25]. In the event of SaaS, it can't be guaranteed about the availability of utilization in need [20]. PaaS is not legitimately arranged for responses of harmful actors on new cloud framework which prompts unverifiable reviewed application parts [17]. IaaS is conveyed from organization model which includes serious security issues. These security issues of service models of cloud computing can be decreased noteworthy through applying trust administration principle in the agent based SOA level (third level) of Fog computing model of our proposed infrastructure [10].

Moving all information from IoT to the cloud for analysis would require unfathomable measures of data transfer capacity. Today's cloud models are not intended for the volume, assortment, and velocity of data that the IoT generates [9] as specified in the past passage. Fog computing is a model that empowers extensive variety of uses and services to the end clients by amplifying cloud computing model towards the edge of network. Exchanging information over the network through internet without human-to-human cooperation’s or human-to-machine associations, is the supremacy of IoT which incorporates elements, for example, versatility support, extensive variety of geo-distribution, availability of wireless accesses and expansive number of nodes make Fog computing, a superior stage for a particular number of IoT services [18].

From the view of IoT, devices are being able to communicate with each other with or without any human inference [9]. A wireless sensor network contains large number of wireless devices considered as the endpoints of the network. Success of IoT is strongly linked with the collaboration of the end points. Therefore, computation will need to go beyond traditional mobile computing scenarios that use smart phones, portables and evolve into connecting existing objects and embedding systems into our environment capable of collaborating among them and should be identified

having a well-defined functionality and connected to a network [21]. All these must have their own identities, physical attributes and interfaces where they will seamlessly be connected into the wireless network as active participants, sharing information whenever and wherever it is needed [7]. Envisioning the practicality of IoT Machine-to-Machine (M2M) communications is an emerging communication standard that provides pervasive connectivity between devices able to interact autonomously.

The service oriented architecture is one of the most widely used architectures for heterogeneous devices. In the other hand, a light-weighted distributed service composition model can be used for data acquisition which will convert basic existing heterogeneous devices into better software units along with complex functionality added with corresponding QoS features following the soft-real time restrictions by the most appropriate sampling time of specific services. [21] Since this is a lightweight model it can be used in the lower levels of the fog computing nodes as they may have lower resources.

For the upper levels of the fog computing nodes we can use agent based Service Oriented Architecture. Agent technology suites complex systems based on distributed computational and information systems. For implementation, we can use Hydra as it targeted the development of a service oriented Architecture based middleware for intelligent networked embedded system which can be deployed on both new and existing networks of distributed wireless and wired devices [2].

2.1 First Layer Of Fog Computing (M2m Communication)

Breaking down the FOG model in the lowest part, M2M devices become both producer and consumer of data and from these devices will be able to learn and gain information and knowledge directly with the data fed from things. All these devices will create data and this huge number of data needs to be send, received and processed by our current infrastructure. As the number of users (in our case it is also Machine) and network increases the software system that runs on small scale mockup may lose their properties.

2.1.1 ***Data Streams***

All the connected devices will transmit data throughout the network possibly continuously. Some major characteristics of data streams [23],

- Data objects may come continuously.
- Stream size may be unbounded and
- Disordered Distributed systems can change the route and therefore unknown data generation process.

In our study of IoT from a data perspective, from the beginning we have to keep in mind that we have to work differently than normal Internet protocols as in the Internet of Things, the main actors become the things. The ultimate goal is for these machines to sense and react to the real world for humans. As of 2012 about 2.5 quintillion (2.5 × 1018) bytes of data are created daily [14]. Now, connecting all the things that are connected would create much more data and this vast volume of data processing become much more critical for existing technologies. Multiple data streams can be generated at anywhere around the world and can be accessed globally via the Internet if being made public. Therefore, a large number of data streams have to be processed efficiently to provide real-time monitoring. For each device to be identified devices stores their configuration in a local database. In case of a smart M2M devices it

locally saves a name, model number, hardware type, unit, version, type and timestamp to the sensor values which creates metadata for each device [4]. The management of the M2M devices are done using gateway. For a non-smart or legacy device, the same is done using another gateway called intermediate gateway (IG) which is configured using a predefined model. This gateway makes the connection between the devices. For better apprehension gateway is diverged into two parts North and South [11]. The North interface of the gateway which implements an API to provide push notification containing sensor measurements and assists in dynamic device discovery where the South interface employs proxy-in and proxy-out.

2.1.2 ***Configuration of Resource Description API***

An initial configuration of the device and its endpoints can be done by XML or JSON file containing the static description [12]. This API reads the configuration file using GET request or the file can be pushed to it. The configuration of the device for the API to be recognized has the attributes,

- Location - It signifies the type of device's location which can be described using GPS co-ordinates, X and Y value.
- Id - Unique identification of the device.
- Name - Name of the device
- Value - Gives the reading or value of the hardware.
- Protocol - It provides information on the type of request.
- Proxy-in - URI to which a device with sensor is connected.
- Proxy-out - URI to which a device with actuator is connected.

Then the configuration of the endpoint for the API to be recognized has the attributes,

- Name - Name of the endpoint.
- Password – Unique password of the endpoint.

- Token – non-cryptographic token for unique identification.

After this the initial configuration files are pushed to the gateway and are examined by the configuration resource API. Then the device and endpoint descriptions are extracted from those files by the API and stored in the local database. Then, when the device sends a GET request to receive the details of the devices connected to the gateway, the API responds with the full list of devices and their descriptions. Therefore, from this the devices are forwarded the data they require by the gateway. This generates the data stream which needs to be transferred which is described in the communication segment.

2.1.3 ***Communication Between M2M***

IoT Promises to build the globe where all the Objects around us will be connected to the Internet and will communicate each other with bare minimum human intervention. Standardization of communication has been already done

In this paper, we have conducted our work on smart objects both stationary and non-stationary.

2.1.3.1 ***Smart Objects:***

In general, Smart Objects are those who can efficiently communicate with Human or other Objects by following some specified protocols. Using smart object oriented IoT, generally means to use smart communication orientation objects being reachable and exploited [14]. But such huge heterogeneous network makes distributed network and management very complex. Intelligence as in 'smart' should be provided with service and actions not embedded inside objects.

As described the four major parts of object oriented IoT are [14], the Application layer encompasses applications based not only on SO's but also on other IT

infrastructures, the Middleware layer provides as set of mechanisms for the naming, discovery, high-level interaction and state management of SOs, the Internet layer includes application, transport, and network protocols for supporting the communication with SOs and among SOs, the Smart Object layer offers programming frameworks and tools enabling the design and implementation of SOs.

Calling this “architecture of Smart Object oriented IoT” which is at higher level of abstraction and promotes an ecosystem of smart objects based on the Internet. We find an architecture consisting of sensing layer, application layer and network layer [28], which was later extended by cloud assistance [14].

We have a successful almost generic paradigm for smart devices [11]. These devices store their configuration in the local database system. This paradigm also transmits metadata. These metadata will be particularly helpful for analyzing data.

2.2 Second Layer Of Fog Computing (Service Oriented Architecture(SOA))

The nodes that are not at the end of the fog computing architecture will have a middleware in order to fulfill the distributed architecture. The way “Things” are going to communicate is a challenging matter. We found some models and before choosing any of them we would like to discuss the models and their development.

Here the main modeling concept is ‘resource’ with all sensors, actuators and processors which are modeled as resources [8].

2.2.1 ***IoT Information Model***

“Things” of Internet of Things can be anything such as human, car, watch, household things, vehicles etc. Here the “entity” is the main focus of interactions by humans or

agents and involves a device that can monitor the “thing” and the portion of software that gives information on the entity or controls that device possible which is called a ‘resource’. A “service” provides a well-defined and standardized interface which offers all required functionalities which will interact with the entities and related process since the ‘resource’ is highly dependent on the device. The service can expose the functionality of a device by accessing its resources. Other low-level services can access these services in order to provide high level functionalities. “Association” is the relation between services and entities, which can be static or dynamic. The concepts need to be presented such a way that will provide interoperable and automated human and machine readable representations.

OWL-DL (Ontology Language Description Logic) provides a platform that is formal and machine processable structure in order to present data collected from different sources.

2.2.2 ***Entity Model***

An entity can have some properties such as, domain value, location and temporal values. An entity can have several values for each of these properties. Location can be Global location or Local location. For global location ontology uses a URI and for local location it can be detailed.

2.2.3 ***Resource Model***

Resource model is the main part that represents an entity digitally. Resource model has some properties of its own like, name, resource id and time zone. Resource also has a functional location property and another attribute known as the resource type. Which can be an instance of any kind of sensors, actuators or tag etc. The resource interface is specified by Access Interface that is also interfaced by an Interface Type

which is a set of instances used in distributed technologies, for example, REST, SOAP, RPC.

2.2.4 ***IoT Service Model***

In IoT service model resources are accessed by services where services provide functionality. Functionality includes collecting information from entities they are connected with or manipulate their physical properties. As we can see the service based approach is so far the best for IoT context, we would like to use the Service Oriented Architecture (SOA).

A service-oriented architecture (SOA) is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology [24].

For the nodes that are not at the end but is in a lower level of the fog architecture we will use the composite model based on the Service Oriented Architecture.

Classic old distributed software architecture doesn't support a network of heterogeneous devices. So, the solution is to propose a middle-ware layer application, that can handle heterogeneous devices running different services on different platforms, which provides a dynamic distributed system assuring flexibility and interoperability along with improving robustness, reliability, availability and scalability if existing SOA lack proper settings of non-functional requirements.

A high-level light-weighted distributed service composition model for improvised data acquisition which will convert basic existing heterogeneous devices into better software units along with complex functionality. This functionality is added with

corresponding QoS features following the soft-real time restrictions by the most appropriate sampling time of specific services.

The scalability of the system depends on the combination of SOA middleware and the service composition model which maintain the efficiency by ensuring the rectification of this combination between services by formation instead of providing the identification of the requesting service, prioritizing the required function ensures a lightweight composition system.

2.2.5 ***IoT Services***

Services of IoT can be represented by five-tuple. We can show them with this definition:

IoT $services_i = < Id_i, Ps_i, Ip_i, Ir_i, At_i >$ (1)

Here,

Id_i= Identification,

Ps_i= Purpose,

Ip_i= Provided Interface,

Ir_i= Required Interface,

At_i= Set of Attributes.

This equation characterizes IoT services from rest of the services on the network [15].

Each IoT service needs to be identified uniquely with an Id or name and we can use URN (Uniform Resource Name) a kind of URI (Uniform Resource Identifier) for that. Services are going to be publicly available and accessible for any other IoT service that requests them with along with a particular purpose.

The operation of an IoT service can be either simple or composite where simple operation defines a service which does not depend on other services for transactions because of having full resources. On the contrary, composite services can be depended on other services. Moreover, IoT services can act as both provider and consumer using different interfaces creating a controlled as well as synchronized mechanism. Here operations are assigned to different ports. Services may use parameters based on its configuration when it acts as a provider.

2.2.6 ***Service Composition Model***

Composite operations which are the core of this model, can be defined in the required interface (Ir) of the service definition. A service composition map is defined by a set of predefined services which is more likely to be a static approach. In a dynamic approach, the service and the called operations are selected in runtime using semantic information. There is a misinterpretation between dynamic selection and dynamic composition. Dynamic composition is very powerful that can determine dynamically, which service can handle the request and increases the complexity in runtime.

When an operation is invoked, the requester knows its maximum execution time and hence, the maximum time it has to wait to receive a response. This mechanism ensures executing operations with soft real-time quality properties.

The service composition model was developed using the Graph Theory. Here the composite operations form the composite map. The relation between these operations is basically the relation between invoker & requested. This composition map can be seen as a composite graph. Each composite operation op of a service S can be viewed as a directed graph:

$$G_{op\ S} = (\ O_{op\ S}, \text{V(G)}, \text{L(G)}, \text{E(G)}) \qquad (2)$$

Where,

$O_{op\,S}$ = the main vertex of the graph and it indicates to the origin service S of the composite operation op_S.

V (G) = a set of vertices from the graph that requires a service on which an operation is invoked from the composite operation.

L(G) = A set of labels where each label carries a requested operation in a required service of V(G).

E(G) = A set of edges between the origin vertex $O_{op\,S}$ and a destination vertex in V(G) which is labeled with an element from L(G). These edges are directed. Each element here is defined by,

$$edge_i(\, O_{op\,S}, op_i, v_j\,) \qquad (3)$$

Here, $O_{op\,S}$ is the origin service and op_i is the requested operation in the required service v_j, verifying E(G) ⊆ o x L(G)x V(G). [15]

2.3 Third Layer Of Fog Computing (Agent Based Soa)

For the nodes that are in the upper level we can use the agent based compositions to make complex compositions and since these nodes have very low chances of having low resources, the model does not need to be lightweight.

There has been a rise of interest in ontologies as artefacts to represent human knowledge. Which leads to a concept titled “marriage” between agents. Here agents work as a glue and the backbone of the system. To make agent based composition effective the required three actors are, service provider, business process manager and users.

This will make the autonomous agents work together to make their goal fulfill. The agents should be able to do some activities which can be listed as: build workflows, compose the external web services and monitor execution.

An agent based framework (Multi Agent Service Environment) which overcomes the limitations of JADE, allows dynamically composing Web services. This architecture is based on Society of Agents and mostly made up with two components:

2.3.1 ***Component Manager***

Each component manager here is in charge of interacting with one or more web services. With the use of WSIG JADE add-on [22]. These can communicate with web services by converting WSDL messages into ACL messages and vice versa. This helps to provision flexible services which is based on some business rules maintained by a rule engine and editable by some operator through an interface. This is titled "On the fly".

2.3.2 ***Workflow Manager***

Goals:

- Supporting users to build the workflows
- Composing external web services
- Monitoring their execution

This is a complex activity to accomplish and workflow manager does by two alternative processes:

2.3.2.1 Predefined workflow

This helps users to select the most related or accurate workflow from a standard and common template used in previous communications. Here the workflow manager works by matching services, which is possible because of common background knowledge of the agents based on shared ontology.

2.3.2.2 Dynamic workflow

This creates a new workflow based on user requirements and compose available atomic services, with the help of a planner. After its creation, it replaces the failed or deprecated or unavailable web services. This also allows users to manually build workflows.

2.3.3 Trust Management

Local names and the certificates are the main building blocks for the Trust Management Principles. These systems avoid completely centralized authority and works as a distributed system which opens up the way to build large peer-peer networks where each node held responsible for its own security and also is in charge of its own security. They will provide proper credentials to access other node's resources.

The authorization is very critical and important point for the trust management because it helps building up the trusted peer to peer network without a centralized control. Every system should ideally follow "least privilege". The RBAC model is a good abstraction of managing complex systems, large systems and systems like corporate environments. RBAC follows three things: principle, permissions and roles.

A many to many relationships incorporates with principals and roles which they are associated with and also incorporates permissions with roles. This enables privilege inheritance schemes among superior and subordinate roles towards other principals. To express properties of authenticable principals, a language has come to being known as SAML which can associate public keys to local names and certify the relation or links between different namespaces, as it happens in SDSI or SPKI certificates. Delegation is particularly important as it activates intermediate agents while acting between the human user and the pure service provider.

2.4 Fourth Layer Of Fog Computing (Generalized Cloud)

2.4.1 ***Fog Nodes in the Cloud***

IoT-enabled applications run for real-time control and analytics. Data transmission between fog nodes and IoT devices can be done using any protocol in real time. This ensures a very small response time. Fog nodes will have transient storage where data can be saved locally and periodically data summaries are sent to the cloud.

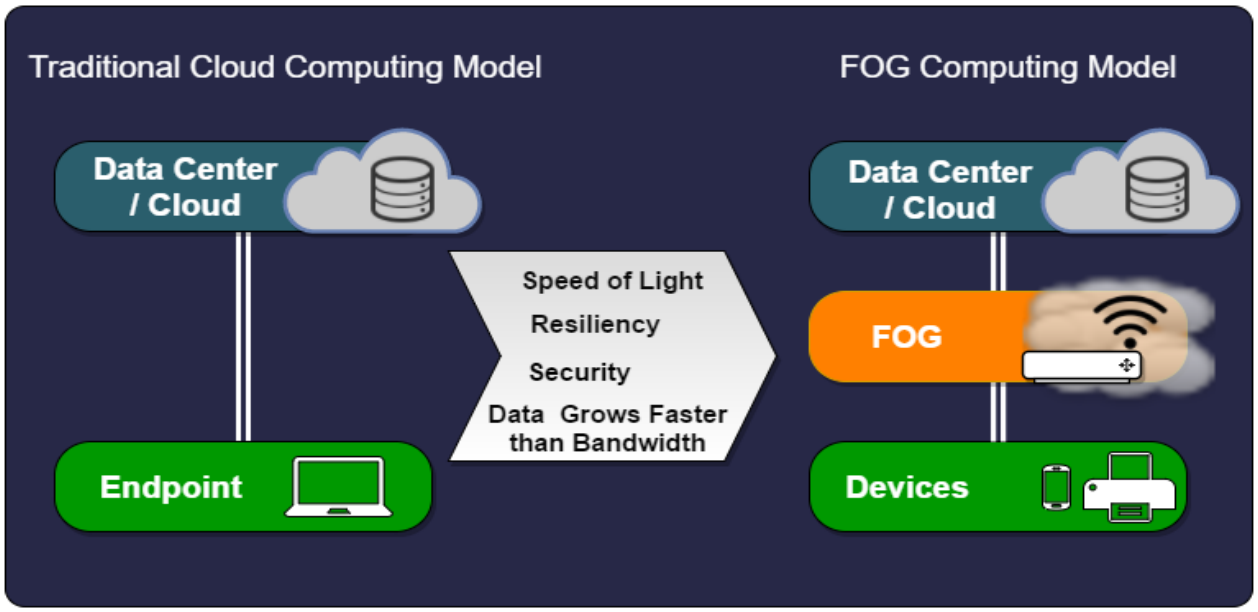

Figure 1. Traditional cloud computing model vs fog computing model

2.4.2 ***The Cloud Platform***

After receiving data from Fog nodes, summaries are collected, analyzed on IoT data and data from other sources to generate business insights and depending on these insights new application rules can be conveyed to the fog nodes [9].

2.5 FOG COMPUTING APPLICATION FOR IOT

Billions of IoT devices adds up the number of new types of IoT devices including machines connected to a controller using industrial protocols instead of IP. Continuous data generation regarding the IoT devices should be analyzed rapidly [9]. This is the major reason for introducing Fog computing which is a significant extension of cloud computing. Instead of utilizing the whole cloud computing platform, Fog computing reproduces new applications and services that enhances data management and analytics.

Fog Computing interrelates building blocks of cloud such as compute, storage, and networking services with end devices virtually and traditional Cloud Computing Data Centers, not necessarily located at the edge of network solely. Fog operates on network edge instead of processing from a centralized cloud which is less time consuming, every bit of data combining at particular access points rather than sending over cloud channels results in less demand for bandwidth and small servers known as edge servers in visibility of users are established which establish faster response time and scalability [18].

It should also be mentioned that the very lower end of the fog computing architecture, data is only transmitted by the connected objects via M2M communication is filtered which eliminates the locally analyzable data and remaining data are transmitted to

the next layers. Handling the two Exabytes of data generated from the Internet of Things regularly becomes easier with Fog computing. Exploding data volume, variety, and velocity ends up with some challenges which are solved by processing data closer to requirement and adjacent to edge where produced. Fog computing avoids necessity of costly bandwidth additions by offloading gigabytes of network traffic from the core network and also evades recursive visits to the cloud for data analysis which results in reduced response time with awareness speed up based on policy and send selected data to the cloud for historical analysis and longer-term storage inside company walls along with ensuring the privacy of sensitive IoT data.

For example, With the semi-permanent storage at the highest level and momentary storage at the lowest level FOG can be used to collect and utilize smart grid data locally and make real-time reports, transactional analytics and data visualization to the higher level to make proper decisions and send commands to the device actuators [18]. Moreover, Software Defined Networks (SDN) concept in FOG will reveal and improve vehicular network problems with connectivity, collusions and high packet loss by increasing vehicle and infrastructure communication and control [21].

Fog enables low latency and context awareness as its nodes provide localization, on the other hand Cloud provides global centralization. Both Fog localization, and Cloud globalization are required for many applications, particularly for analytics and Big Data. Fog collectors consumes the data generated by grid sensors and devices at the edge where some of this data are related with protection and control loops that require real-time processing [7].

In short, characteristics of Fog computing which make it surpass cloud computing are edge location, location awareness, low latency to support endpoints with affluent services at network terminals, geographical distribution with very large number of nodes in demand of widely distributed deployments as sensor networks in general,

large-scale sensor networks to monitor the inherently distributed systems, requiring distributed computing and storage resources, support for mobility, real-time interactions rather than batch processing, supremacy of wireless access, heterogeneity, fog components must interoperate as well as services must be federated across domains, focuses on the ingestion and processing of the data closer to source.

Chapter 3
PROPOSED INFRASTRUCTURE AND IMPLEMENTATION

3.1 Proposed Infrastructure

Our proposed infrastructure is allotted into several levels of nodes, at the very lower end, there are the devices or ‘Things’ which utilizes the Machine to machine (M2M) communication protocol. This protocol is particularly beneficial to a very high level of communication messaging among the ‘Objects’ or ‘Things’, intelligently. This layer then communicates with the next two following layers which are addressed as the ‘Middle Ware’. This is based on their universal and local locations, which we designated as the ‘region’s. The second layer of Fog or the lower one of the Middle Layer utilizes the Service Composition model. It is based on Service Oriented Architecture which is a novel solution in this context and is a very light weight model that suites devices with lower resources. The third layer of Fog or upper portion of the Middle Layer utilizes the Agent based composition. Which can compose complex compositions depending on the available resources. The whole middle layer also opens up a peer to peer communication network without any centralized control but secured. This provides a better interoperability for the ‘Objects’. These layer follows the Fog Computing Architecture. The M2M portion only sends the data which is required to be sent in the higher levels or it just saves it locally. The local data in the Higher and Lower Layers of the Middle Layer are also saved in the Fog Computing Context.

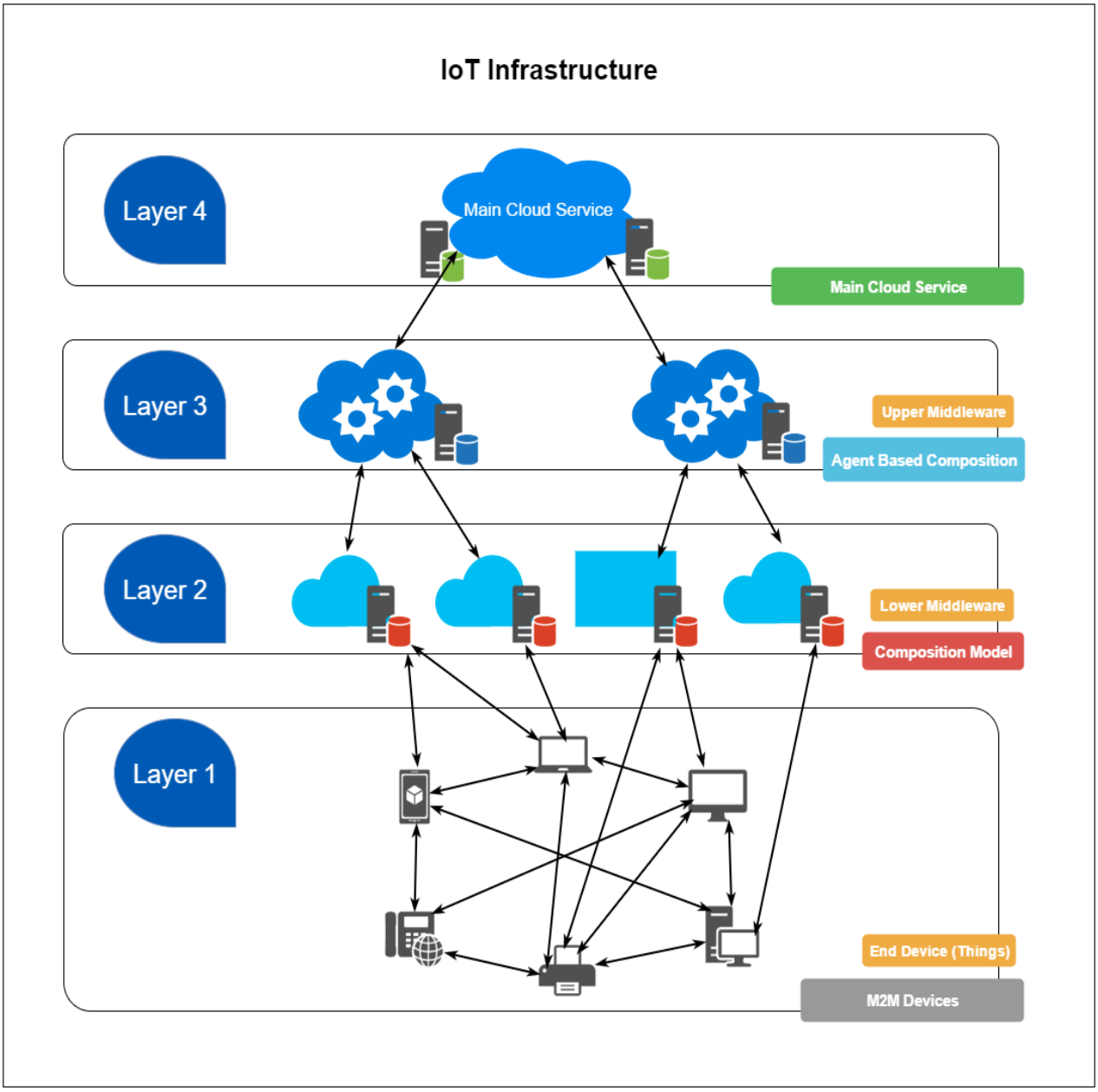

Figure 2. IoT infrastructure model with FOG implementation

But in our infrastructure instead of doing that, the Upper Layer of the Middle Layer learns which service to invoke in order to get the local data. Which is possible by using the proper agent from the society of agents. This opens up the support of devices to be executed as the middle layer. The last or the Highest Layer of the infrastructure is the main cloud service where the data is ultimately sent or processed.

Our infrastructure tries to utilize every layer properly, which leads to the support for a wide range of devices. For the security portion of the lower end, we have utilized the ‘Trust management’.

The existing infrastructure that came into being from the discrete development of IoT needs to be specifically modeled in order to be properly utilized, characterized and also make commercially available so that everyone can cope up with that. We believe that our proposed model will definitely be able to fulfill these requirements.

In brief, in our proposed model, the first layer of the Fog is designed for Machine-to-Machine (M2M) interaction which generally collects data from end devices. The second layer works based on the service composition model and third layer works with the agent based composition. At all part of the Fog, the time scales of these interactions range from seconds to minutes (real-time analytics), and even days (transactional analytics). It results in, the Fog supporting several types of storage, from short-lived at the lowest layer to semi-permanent at the highest layer. Wider geographical coverage, and longer time scale can be obtained in higher layers. The ultimate, global coverage is provided by the Cloud, which is used as repository for data that has a permanence of months and years, and which is the bases for business intelligence analytics [7].

3.2 Implementation

In our Experimental setup, end devices, SOA Architecture and Agent based Architecture have been represented using Virtual Machines(VM). For this purpose, we have chosen Microsoft Azure as an implementation structure. Azure datacenters were situated in different geographical positions, this is really efficient and convenient to perform some test runs. Initially we planned to use two different geographical positions: North Central US, South Central US and Central US. The VMs represented the SOA, Agent based SOA and machines which were in the same geographically available data centers. The main cloud service could be deployed in any region.

Figure 3 below shows the deployed infrastructure in Azure using VMs in different layer. In the above figure, SCUSL1M1, SCUSL1M2 and SCUSL1M3 are VMs which represent the layer 1(M2M) in the South-Central US region and NCUSL1M1 belongs to layer 1 in the North Central US region. Next, in the second layer(SOA), SCUSL2M1 and SCUSL2M2 are in the South-Central US region and NCUSL2M1 belongs to the North Central US region. In the third layer (Agent based SOA), SCUSL3M1 is in the South-Central US region and NCUSL3M1 is in the North Central US region. Finally, CUSMAIN is the main cloud server.

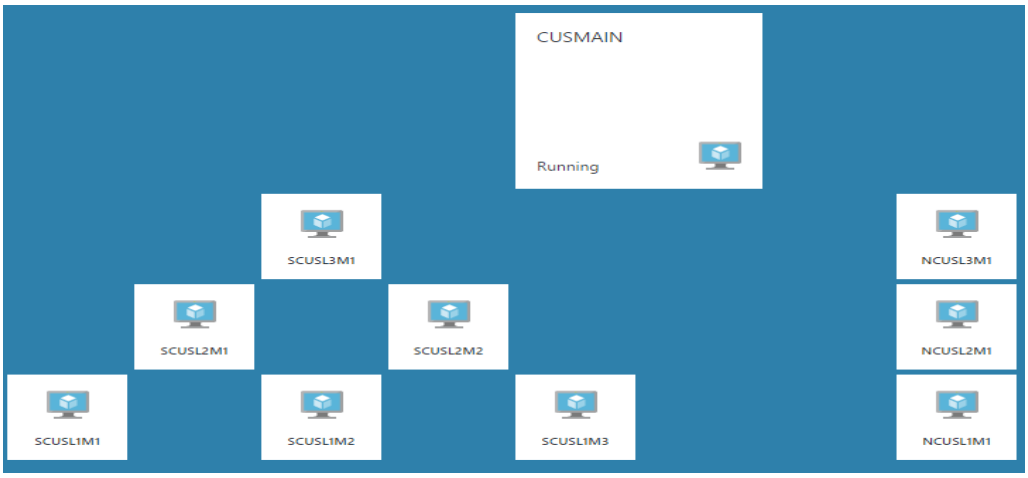

Figure 3. Deployed virtual machines for test run of our infrastructure

As mentioned before in this paper, our infrastructure has four layers from M2m to main cloud server. In case of implementation, communication between these layers was established with different algorithm and pseudocodes as the structure and mechanism of each layer follows different approaches. But the request of data requested by any of the VM and response of that requested data replied by any server are represented through a json format. Request format is uniform for any request from any layer and response format is also uniform for every reply in any layer.

Request JSON format:

```
{
  "REQ":{
    "authentication":{
      "USERID":"user.name",
      "password":"password123"
    },
    "token":"2b2c5f9e6655ce42740584f4c25c85b6",
    "service":{
      "name":"environment",
      "components":"temperature,humidity"
    }
  }
}
```

Response JSON format:


```
{
  "RES":{
    "Token":"2b2c5f9e6655ce42740584f4c25c85b6",
    "C.Service":{
      "ServiceName":"environment",
      "provider":"metro"
    },
    "B.Service":{
      "ServiceName":[
        "temperature","humidity"
      ],
      "value":[
        "32","55"
      ],
      "optionalParameters":{
        "protocol":[
          "http","http"
        ],
        "url":[
          "http://www.example.com/temperature",
          "http://www.example.com/humidity"
        ],
        "ttl":[
          "500", "500"
        ],
        "timestamp":[
          "2016-10-08 08:26:27","2016-10-08 08:26:27"
        ]
      }
    }
  }
}
```

In our implementation process, we have defined services in two phases: Simple and Complex. Complex services are combined with several basic or simple services. For example, from our json format we can observe that “environment” is a complex service which consists simple services such as “temperature” and “humidity”. There can be many independent simple services as well which are not generalized in a particular complex service. The main advantage of this generalization of simple services as a specific complex service is, there can be plenty of simple services with same name but different category or mechanism for instance “temperature” can be of many categories such as environment, food, room, water and many more. So, if a machine wants to request for environment temperature, it is easier to fetch the data value from the server as a complex service called “environment” which consists “temperature”.

To Implement our proposed infrastructure, at first, we built some algorithms for layer to layer communications for each layer and applied them in datadog in order to generate graphs to compare the results. Datadog provides monitoring as a service and to use that we need to integrate datadog agents in azure VMs which sends metric of the azure VMs to the datadog dashboard. But datadog agents can have delay upto 2 minutes to send the data to datadog dashboard which may cause a bit delay in the generated graphs. Therefore, we implemented our results in the second phase by VM monitoring in Microsoft Azure with improved algorithms. So, two phases can be observed in case of our implemented pseudocodes and results.

Though our proposed infrastructure worked for both of phase1 and phase2 pseudocodes and generated proper results but we implemented our infrastructure twice to build more efficient algorithms and to get more appropriate results which will help us to reserve the Quality of Service more precisely.

3.3 Pseudocode

3.3.1 ***Pseudocode for Implementation Phase 1***

3.3.1.1 ***Communication from M2M to SOA:***

In our communication from M2M to SOA algorithm, SOA is always listening for incoming requests from M2M devices. When a request for service is received its saved as a String serviceName. Then getComplexServiceValues(serviceName) method is called where firstly, a query runs in the "complex_service" table in the database and return value which is saved in variable named result (mysql type). If result is not null, then a set of basic service names are received. Next, within a loop, for each basic services the method getSimpleServiceValues(String basicServiceName) is called to retrieve all the components or parameters of that service from the "simple_services" table. Otherwise, if the query result was null, that means the serviceName does not belongs to any complex service but it can be an independent simple service and therefore getSimpleServiceValues(String basicServiceName) is called to retrieve data. Now, in the getSimpleServiceValues(String basicServiceName) method, a query runs in "simple_services" table in database and stores all the parameters in the result variable and if the result is null, that means SOA layer does not consist this data, so, it will send an http request to an Agent in next layer which is the third layer(Agent based SOA) of our infrastructure. On the other hand, if result is not null, then within a loop, all the parameters are retrieved with values and then the difference between requested timestamp and response timestamp to observe if the ttl has expired or not and it exceeds the ttl then the request is sent again and an agent will update the table and response is retrieved again after update in a different thread. The algorithm is given below:

Algorithm1. ServiceBootstrap(Sevice s): getComplexServiceValues(String serviceName), getSimpleServiceValues(), ttlCount(String serviceName, String timestamp)

```
/* SOA data fetch, wait for request from M2M, Received request */
String serviceName
Call getComplexServiceValues(serviceName)
Intialize Object serviceResult [ Map<String, String> valueresult.size()]
If (valueresult is not null) then
k=0
for (Object i: valueresult.keyset()) {
serviceResult[k] = new M2MReply(i.toString(), valueresult.get(i).toString())
k++
}

getComplexServiceValues(String serviceName):
result = query for searching all basic services of requested complex service
if (result is null) then
call getSimpleServiceValues(String serviceName)
else
for (every basic service)
call getSimpleServiceValues(String basicServiceName)

getSimpleServiceValues(String serviceName):
result = query for searching requested service and values from simple service table
if(result is null) then
send http request to agent based on that region
```

```
else
for(Object i: result.res.keySet()){
Map<String, String> timestampresult.put(serviceName,timestamp)
Map<String, String> ttlresult.put(serviceName,ttl)
Map<String, String> valueresult.put(serviceName,value)
Map<String, String> urlresult.put(serviceName,url)
serttl = Call ttlCount(serviceName, timestamp)
If (ttl - serttl < 0) then
http request to url for latest value
return valueresult.put(serviceName, value)


ttlCount(String serviceName, String timestamp):
st = MiliSeconds(timestamp)
ct = currentTimeInMilis()
diff = ct – st
return diff
```

3.3.1.2 ***Communication from SOA to Agent based SOA:***

We have divided our Agent based SOA in three parts, Reply Agent, Update Agent and Fellow Agent.

Prioritizing the services based on how frequently they are requested, success rate and up time, the agents decides as an “Artificial Intelligent”, which services should be served at the first place. A Reply Agent always keeps listening requests sent from layer two(SOA) and an Update Agent also keeps updating the simple services through HTTP request based on priorities.

In our communication from layer two (SOA) to layer three (agent based SOA) algorithm, a reply Agent of layer three is always listening for incoming requests from layer two and an update Agent also continuously updates simple service values via HTTP request based on priority where priority is fixed depending on request or update counts which increased every time a service is requested. For the response sent by the agent based SOA executes getComplexServiceValues(String serviceName) and getSimpleServiceValues(String basicServiceName) which works the same way described in SOA to fetch the values from "agent_lookup_table" database. But here if it receives a null value it requests fellow agents situated in its own region for the service. If the service is still not found it requests the main server for the service. After that increaseUpdateCount(serviceName) is called to increase the "update_count" of that service by one and then the increasePriority(serviceName) from the update agent is called to update the priority of services in "simple_service" table based on the "update_count". The priority of the services is determined observing the request rate of the services and if several services have the same update_count, then it observes the most recent update timestamp.

The update agent on the other hand continuously updates the simple service values. For updating, it calls the updateTable() method in which all the complex services are retrieved in priority based order and runs a loop to call updateComponents(String serviceName) to retrieve simple service URL and make an http request to the main server through that URL and get "value" and "ttl". Finally, the updateValues (String serviceName, String serviceValue, String ttl) is called to save the new "value" and "ttl" of that service in the "agent_lookup_table" database.

The fellow agent is called when an agent based SOA receives null after querying in database. This agent also executes getComplexServiceValues(String serviceName) and getSimpleServiceValues(String basicServiceName) and responds with the "value" to its requested fellow agent. The used algorithm is described below:

Algorithm 2. ReplyAgent(Sevice s): getComplexServiceValues(String serviceName), getSimpleServiceValues(String serviceName), increaseUpdateCount(String serviceName)

/* Agent based SOA data fetch, wait for request from SOA, Received request */
String serviceName
updateAgent ua = new updateAgent();
Call getComplexServiceValues(serviceName)
Call increaseUpdateCount(serviceName);
Call ua.increasePriority(serviceName);
Intialize Object serviceResult [Map<String, String> valueresult.size()]
If(valueresult is not null) then
k=0
For(Object i: valueresult.keyset()) {
serviceResult[k] = new M2MReply(i.toString(), valueresult.get(i).toString())
k++
}
increaseUpdateCount(String serviceName):
Increase and update "update_count" of "serviceName" by 1 in Database

getComplexServiceValues(String serviceName) and getSimpleServiceValues(String serviceName) is same as explained in SOA

Algorithm 3. UpdateAgent(Sevice s): updateTable(), updateComponents(String serviceName), updateValues(String serviceName,String serviceValue), increasePriority(String serviceName), changePriority(String serviceName, int i)

/* Update Agent always runs updateTable() in the background */

updateTable():

result = query for all available complex service by priority

csname[number of received services]

foreach (i : for all values of csname[])

CallUpdateComponents(csname[i])

UpdateComponents(serviceName):

result = query for simple service “URL”

Make threads and send http request and get new value from that url

value = http response

Call updateValues(serviceName, value)

updateValues(String serviceName, String serviceValue):

Update simple service value to “serviceValue” of the “serviceName”

increasePriority(String serviceName):

Get complex services with decreasing update_count

foreach(I : complex_services)

Call changePriority(complex_serviceName, i+1)

changePriority(String serviceName, int i):

Set and update priority to “i” of the “serviceName”

Algorithm 4. fellowAgent(Sevice s): getComplexServiceValues(String serviceName), getSimpleServiceValues(String serviceName), increaseUpdateCount(String serviceName)

/* Agent based SOA data fetch, wait for request from SOA, Received request */

Ip = ip addresses of another agent based SOAs

getComplexServiceValues(String serviceName) and getSimpleServiceValues(String serviceName)

is same as explained in SOA

increaseUpdateCount(String serviceName):

is same as explained in agent based SOA

3.3.1.3 ***Communication from third layer to main server:***

When second and third layer is unable to fetch the requested service or data, then agent sends an http request to the main server. After getting the request from third layer, the main server call the getComplexServiceValues(String serviceName) method which follows exactly similar algorithm described in section I (Communication from M2M to SOA) along with the getSimpleServiceValues(String serviceName) using the main cloud server(main_server database) except the method named ttlCount(String serviceName, String timestamp) and the main server won't need to send any http request.

3.3.2 ***Pseudocode for Implementation Phase2***

In the second phase of algorithm, only "Communication from End Devices to SOA" and "Communication from SOA to Agent based SOA" have been improved and the rest of the algorithms are the same as phase1.

3.3.2.1 Communication from End Device to SOA

For communication within the end devices and SOA, a server is always running to process any request that comes from the end devices. Any request that has been received is saved as a String “serviceName” and get_complex() meathod is called upon which runs a query in “complex_service” table within the SOA database and returns mysql type variable definite as result. If the return value is not null then required set of basic services are recieved. If result is not null, then a set of basic service names are received. Next, within a loop, for each basic services the method get_Simple (String cs_id) is called to retrieve all the components or parameters of that service from the “simple_services” table. Otherwise, if the query result was null, that means the serviceName does not belongs to any complex service but it can be an independent simple service and therefore get_Simple(String cs_id) is called to retrieve data. Now, in the get_Simple () method, a query runs in “simple_services” table in database and stores all the parameters in the result variable and if the result is null, that means SOA layer does not consist this data, so, it will send an TCP Socket request to an Agent in next layer which is the third layer (Agent based SOA) of our infrastructure. On the other hand, if result is not null, then within a loop, all the parameters are retrieved with values and then the difference between requested timestamp and response timestamp to observe if the TTL has expired or not and it exceeds the ttl then the request is sent again and an agent will update the table and response is retrieved again after update in a different thread.

The algorithm is given below:

Algorithm1. SOA: M2M_Response search(), get_complex(),get_simple(String csid), get_simple(), ttlCount(String serviceName, String timestamp)

```
Class SOA:
M2M_Response search():
        if (cs_id != null) then  get_simple(cs_id)
        else then
              get_simple()
            if (response.B_Service.isEmpty()) then
                return null
    return response

String get_complex():
        HashMap res = get_complex_services_from_db()
        if ((res.get("csid")).isEmpty()) then return null
        return res.get("csid")).get(0)

void get_simple(String csid):
          HashMap res = select_from_simple_with_relation(csid)
          rowLength = res.get("ss_name").size()
          for (int i = 0; i < rowLength; i++) {
            Simple_Service ss = (new M2M_Response()).new   Simple_Service()
            ss.Ss_name = res.get("ss_name").get(i)
            ss.ss_value = res.get("ss_value")).get(i)
            response.B_Service.add(ss) }

void get_simple():
```

```
        HashMap res = select_from_simple(SERVICE_NAME)
        if (!(res.get("ss_name")).isEmpty()) then
            int rowLength = res.get("ss_name")).size()
            for (int i = 0; i < rowLength; i++) {
                Simple_Service ss = (new M2M_Response()).new    Simple_Service();
                ss.Ss_name = res.get("ss_name").get(i)
                ss.ss_value = res.get("ss_value").get(i)
                response.B_Service.add(ss) }
ttlCount(String serviceName, String timestamp):
                st = MiliSeconds(timestamp)
                ct = currentTimeInMilis()
                diff = ct – st
 return diff
```

3.3.2.2 ***Communication from SOA to Agent based SOA***

We have already mentioned about the three types of agents which are, Reply Agent, Update Agent and Fellow Agent. In the second phase of our implementation only the reply agent is improved as Checker agent and rest of the agent follows the phase1 algorithms.

For the response sent from agent to the second layer, always executes “checkerAgent” inner class under the “AgentSociety” class. “AgentSociety” determines where to go and how to get the result. There are two methods inside “CheckerAgent” which are get_complex() and get_simple(). These methods query throughout its own database which works the same way described in SOA to fetch the values from “agent_lookup_table” database. But if it receives a null value it

requests its fellow agents for the service. If the service is still not found it requests the main server for the service.

One additional thing from SOA server is that, these two methods also increases the "update_count" of that service by one and increase priority from the update agent is called to update the priority of services in "simple_service" table based on the update_count.

Algorithm 2. AgentSociety: SOA_server compile(),CheckerAgent(String serviceName), CheckerAgent(String serviceName, List a), get_simple(),get_simple(String csid), increaseUpdateCount(String serviceName)

```
Class AgentSociety:
    final M2M_Request req;
AgentSociety(M2M_Request req):
        this.req = req;

SOA_server compile():
    CheckerAgent ca
    if (req.COMPONENTS.isEmpty()) then ca = new
CheckerAgent(req.SERVICE_NAME)
    else then ca = new CheckerAgent(req.SERVICE_NAME,
req.COMPONENTS)
     if (ca.result != null) then
     call increaseUpdateCount( String SERVICE_NAME)
     return ca.result
return null
```

```
Class CheckerAgent:
    SOA_server result = null
    final mysql DB = mysql()
    final String SERVICE_NAME
CheckerAgent(String serviceName):
    this.SERVICE_NAME = serviceName
     call get_complex()
    if (getResult() == null) then call get_simple()
        else then call get_simple(result.C_Service.csid)
  CheckerAgent(String serviceName, List a):
     this.SERVICE_NAME = serviceName
     call get_complex()
          if (getResult() != null) then call  get_simple(result.C_Service.csid, a);

void get_complex():
     HashMap res = select_from_complex_db(SERVICE_NAME)
     if (!(res.get("csid")).isEmpty()) then
        result = SOA_server()
     Complex_Service cs = (new SOA_server()).new Complex_Service()
     add res.get(all cs values).get(0) to all cs column
       result.C_Service = cs

void get_simple():
     HashMap res = select_from_simple_db(SERVICE_NAME)
     if (!(res.get("ss_name")).isEmpty()) then
     result = SOA_server()
```

```
        Simple_Service ss = (SOA_server()).new Simple_Service()
        add res.get(all ss values).get(0) to all ss collumn
         result.B_Service.add(ss)

void get_simple(String csid):
       HashMap res = select_from_simple_with_relation_db(csid))
       rowLength = res.get("ss_name").size()
       for (int i = 0; i < rowLength; i++) {
        Simple_Service ss = (new SOA_server()).new Simple_Service()
        add res.get(all ss values).get(i) to all ss collumn
         result.B_Service.add(ss)

void get_simple(String csid, ArrayList optionalParam):
        for (int ii = 0; ii < optionalParam.size(); ii++) {
            HashMap res =
select_from_simple_with_optional_param_db(csid,optionalParam.get(ii)
         if (res.containsKey("ssid") && !(res.get("ssid")).isEmpty()) then
        Simple_Service ss = (new SOA_server()).new Simple_Service()
        add res.get(all ss values).get(0) to all ss column }
          result.B_Service.add(ss)

increaseUpdateCount(String serviceName):
       Increase and update “update_count” of “serviceName”  by 1 in Database
```

3.3.3 ***Traditional Cloud computing:***

Figure 4 shows the deployed present infrastructure in Azure using VM. Among the VMs, in the first layer end devices “TSCUSMACHINE1”, “TSCUSMACHINE2”

and “TSCUSMACHINE3” are situated in the South-Central US region and “TNCUSMACHINE1” is in the North Central US region. Finally, “TCUSMAIN” is the main cloud server.

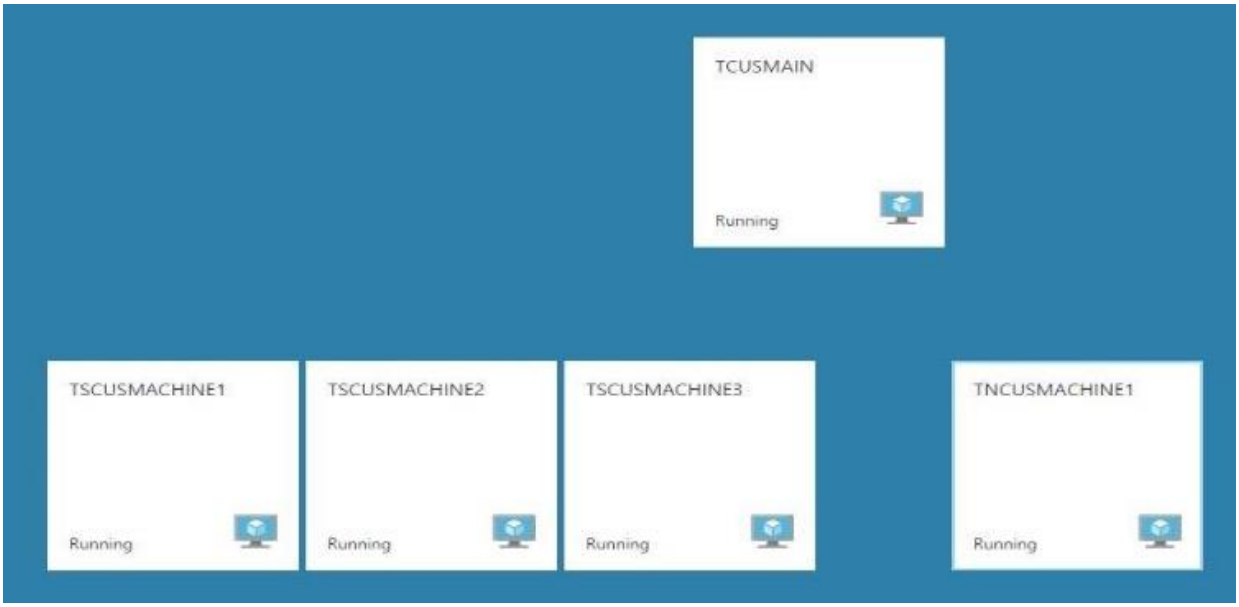

Figure 4. Deployed VMs in Azure for traditional cloud computing infrastructure

In case of implementing present computing infrastructure, the main server directly gets the request from end devices in same json format described before. Afterwards, the main server fetch service from the “traditional_main_server” database and send reply to the end devices through json response format. The used algorithm is described below:

Algorithm 5. serverCommunication(Service s):
Mysql result = query for searching service data from main server TCUSMAIN Object [] serviceresult of result size for(Object i: result.res.keySet()){ Map<String, String> valueresult.put(serviceName,value serviceresult[k] = new endReply(i.toString(), valueresult.get(i).toString()) k++ }

3.3.4 ***Flowcharts***

3.3.4.1 ***Request and Response of SOA***

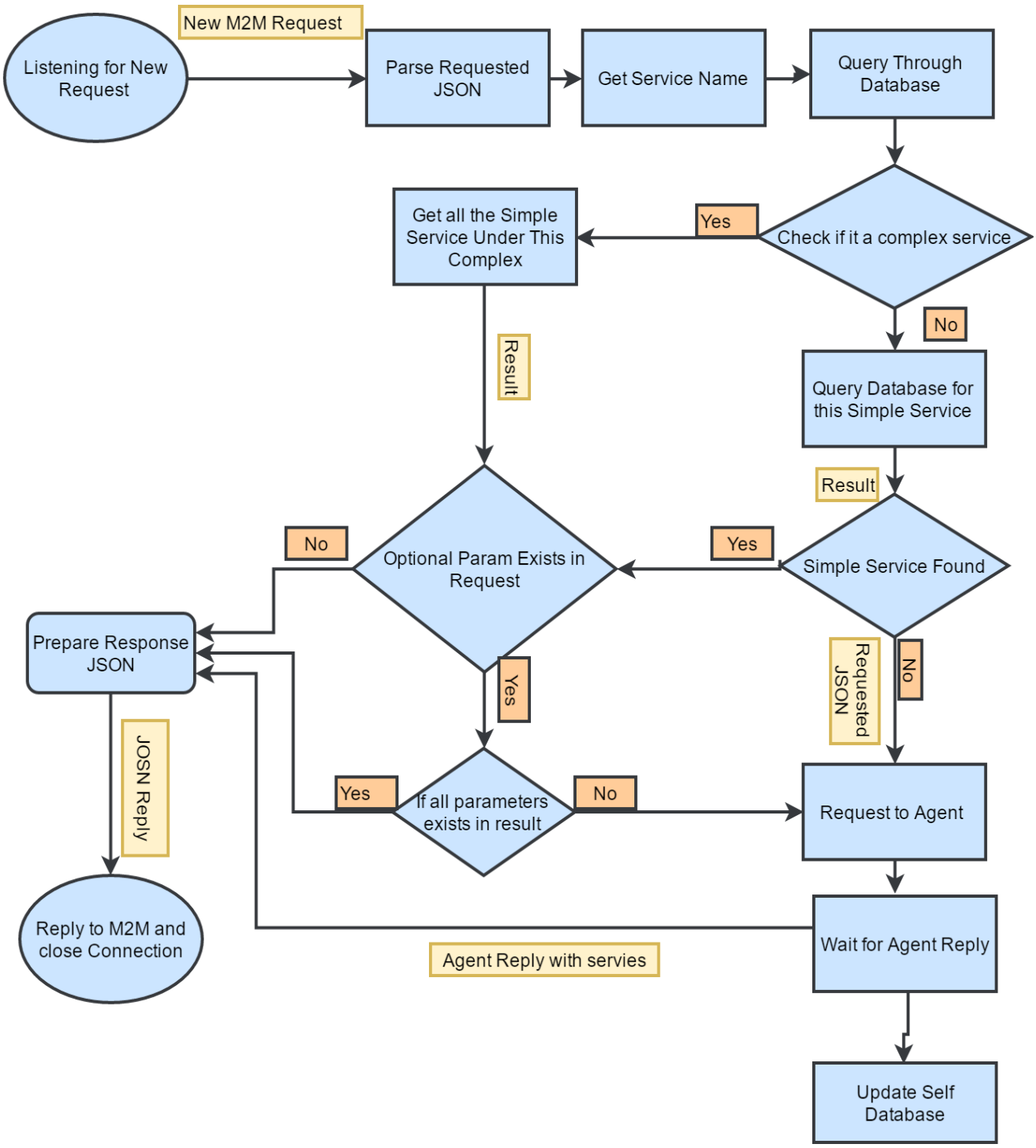

Figure 5. SOA request and response

3.3.4.2 *Checker Agent*

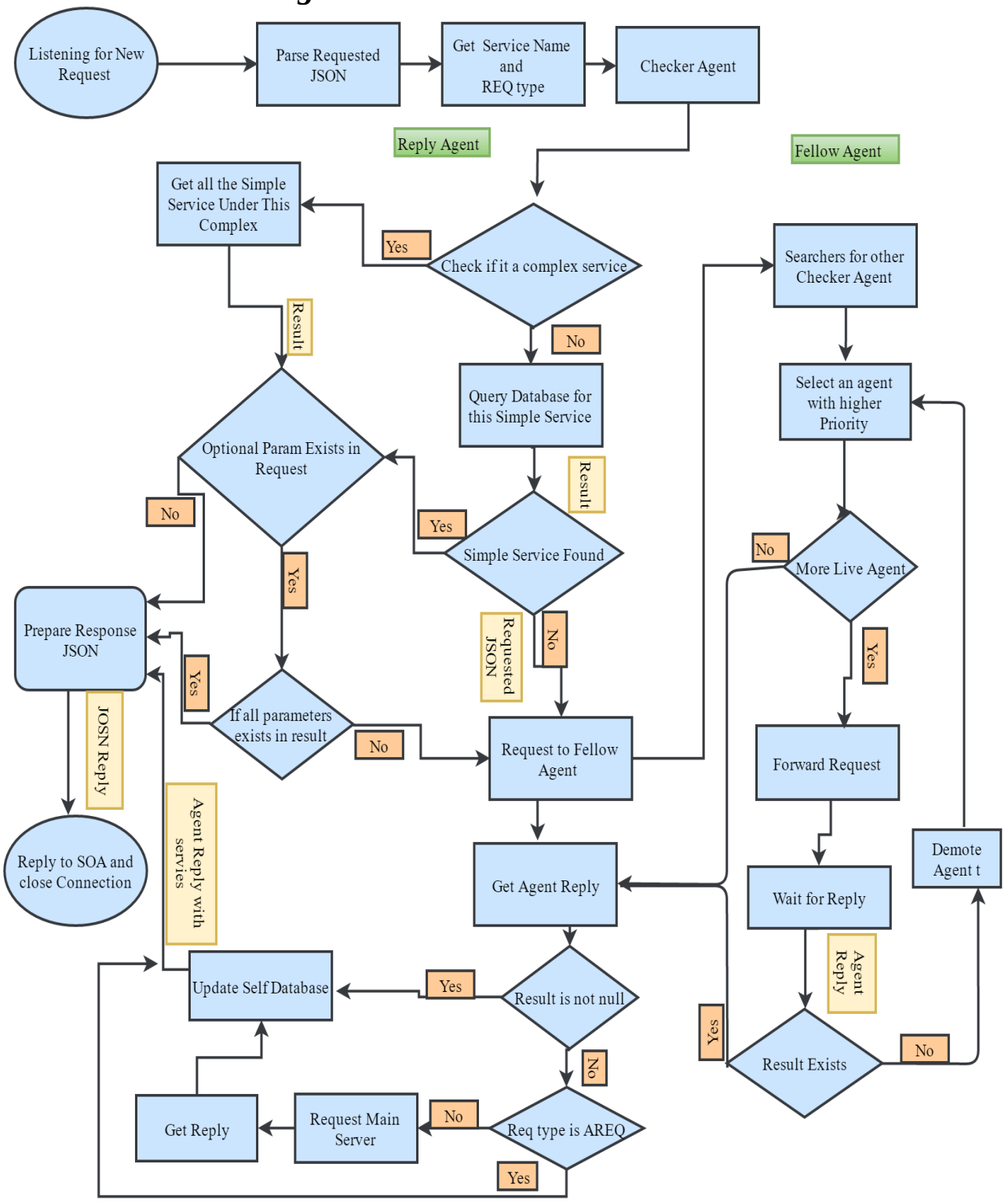

Figure 6. Checker Agent

*3.3.4.3 **Update Agent***

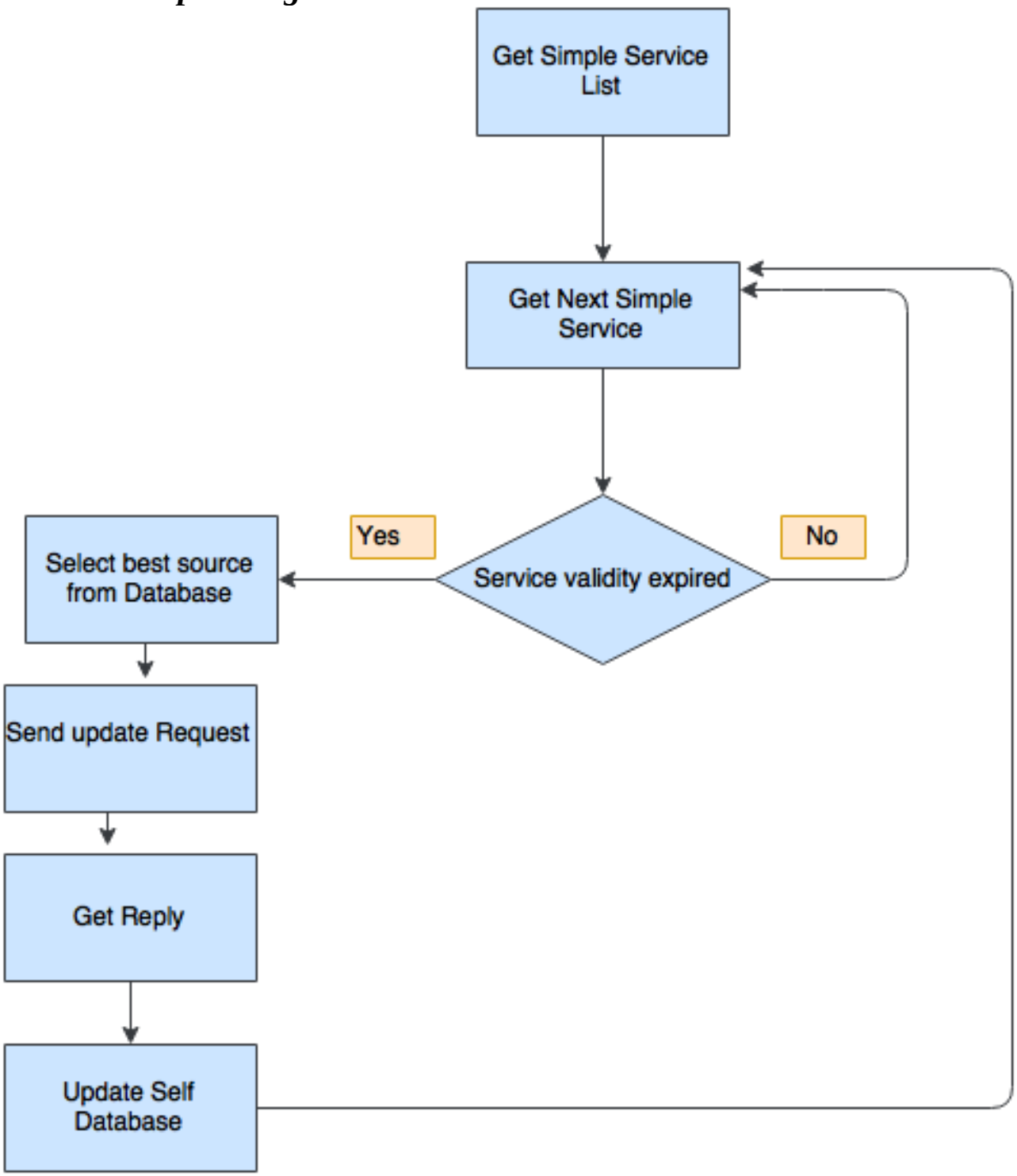

Figure 7. Update Agent

3.3.4.4 Agent to Main Server Communication

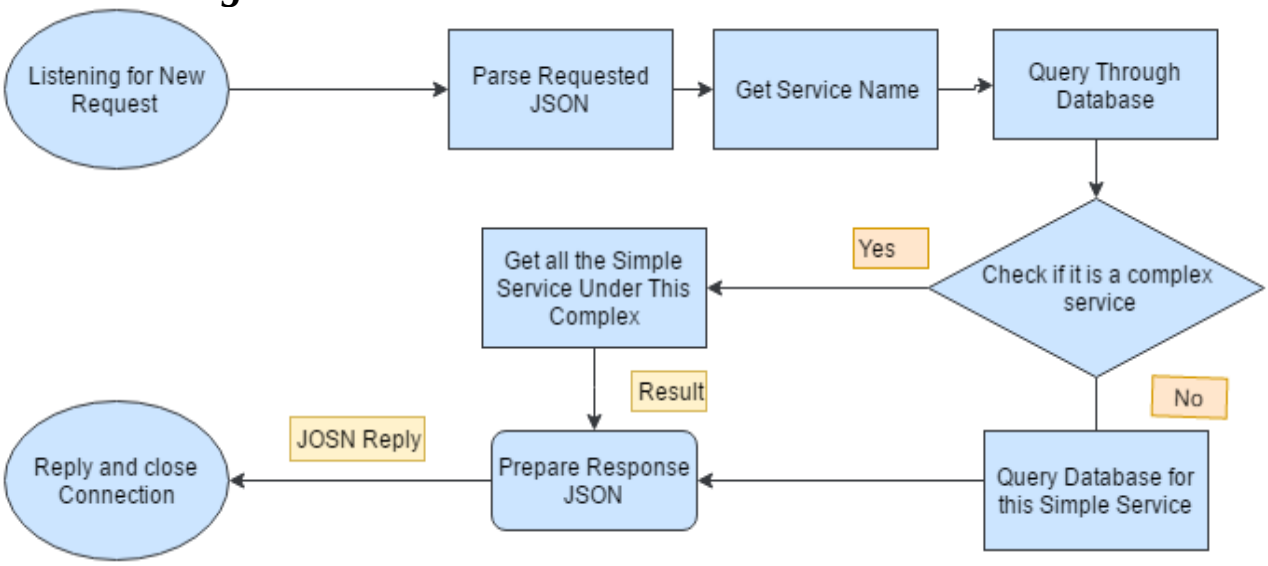

Figure 8. Main server request and response

3.4 Comparison between Traditional and Proposed Infrastructure algorithms:

3.4.1 Less Latency:

If we consider the worst case in our proposed infrastructure model, three end devices in the south-central US “SCUSL1M1”, “SCUSL1M2” and “SCUSL1M3” request a service. Both “SCUSL1M2” and “SCUSL1M3” can get response from the updated second layer without requesting for the same service to the main server which reduces latency as it was saved in second and third layer while fetching for SCUSL1M1. On the other hand, in traditional algorithm the main server will get request three times for the same service by “TSCUSMACHINE1”, “TSCUSMACHINE2” and “TSCUSMACHINE3”.

3.4.2 ***Local Backup:***

If for any reason the main server is not available or get, the end devices can get response from the second and third layer as they have stored the values in their own databases. But for the traditional infrastructure if the main server is unavailable the whole communication is halted.

3.4.3 ***Less Bandwidth and Traffic:***

As in our proposed infrastructure can result in less latency than the present infrastructure, it will help to decrease the amount of bandwidth and data traffic because all the requests are not necessarily going to the main server which is the result of dividing the infrastructure in different layers with locally updated backup data.

Chapter 4

RESULT ANALYSIS

4.1 Result Graph in Individual VMs

A small-scale experiment was performed to monitor the network usage of each of the VMs for our proposed infrastructure by requesting and responding with JSON amounting to a few hundred kilobytes. It is to be considered that for a large-scale deployment the request and response will exceed by millions and network will be adjusted to cope up with delivering terabytes of data.

Table 1. VM Information

VM Name	Layer	Location
SCUSL1M1	Layer 1	South Central US
SCUSL1M2		
SCUSL1M3		
SCUSL2M1	Layer 2	
SCUSL2M2		
SCUSL3M1	Layer 3	
NCUSL1M1	Layer 1	North Central US
NCUSL2M1	Layer 2	
NCUSL3M1	Layer 3	

4.1.1 ***Result Graph Using Datadog:***

4.1.1.1 ***Result Graph of South Central US:***

The graphs below show the network usage of VMs of South Central US which were involved with the test environment while the experiment was conducted.

4.1.1.2 ***South Central US Layer 1:***

For a trial within the first layer, request for the same service was sent from every device, “SCUSL1M1” at 2:31:40am (Figure 9), “SCUSL1M2” at 2:36:00am, “SCUSL1M3” at 2:36:00am.

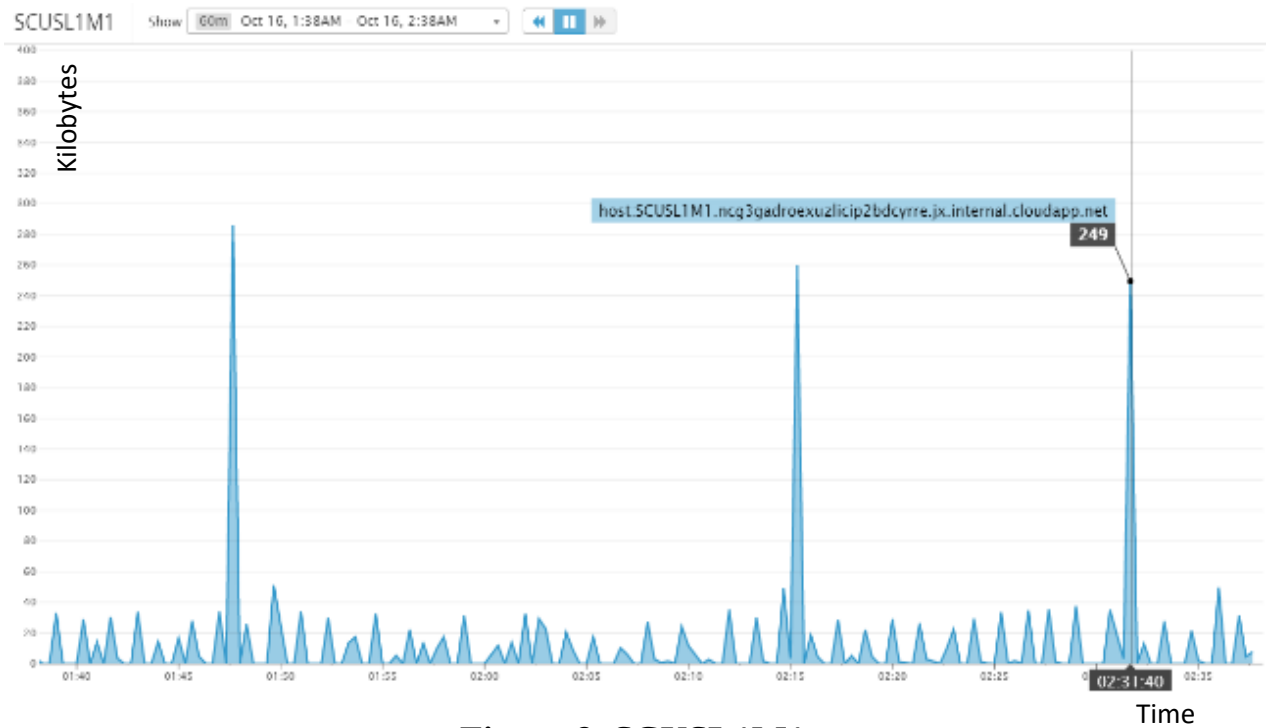

Figure 9. SCUSL1M1

4.1.1.3 ***South Central US Layer 2:***

Initially, VM of the second layer “SCUSL2M1” received the request but it did not have the service. So, request of the service was sent to “SCUSL3M1” at 2:31:00am (Figure 10) in the third layer.

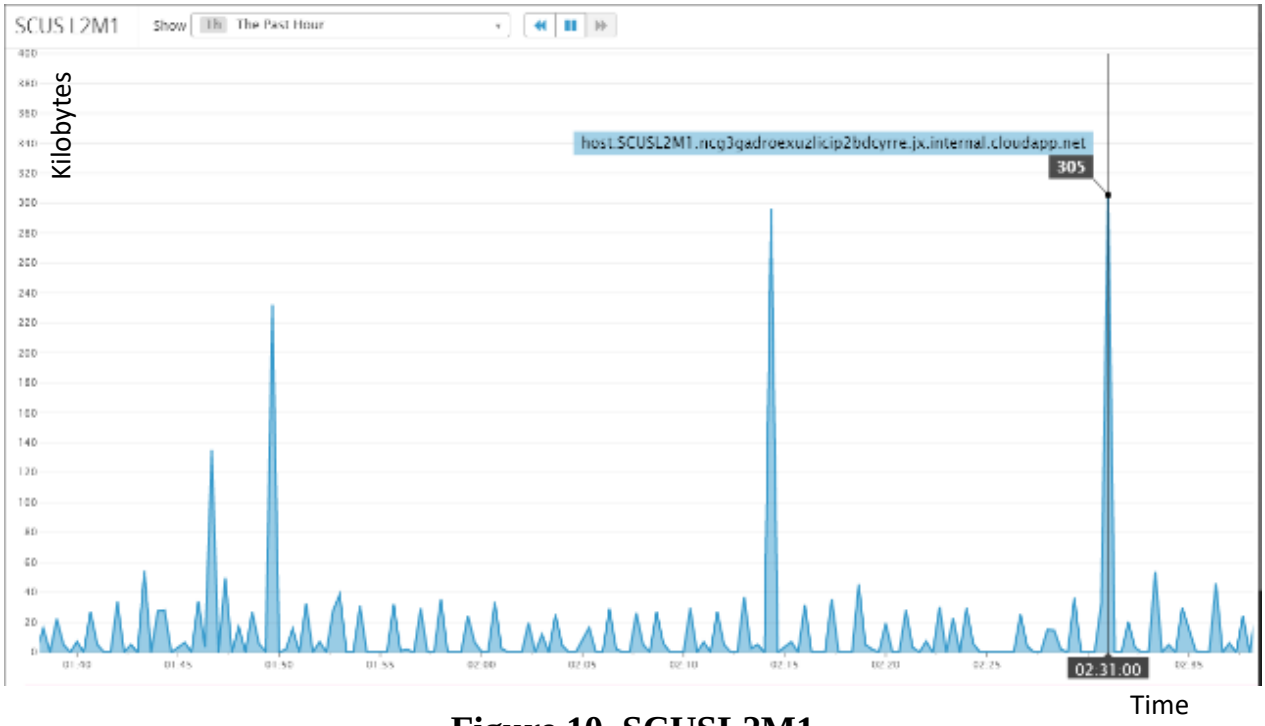

Figure 10. SCUSL2M1

4.1.1.4 ***South Central US Layer 3:***

When the service was not even found in the third layer it was sent to the main server “CUSMAIN” at 2:30:40am (Figure 11). Later, from the main server the result was saved and sent back to the third layer and after that in the second layer.

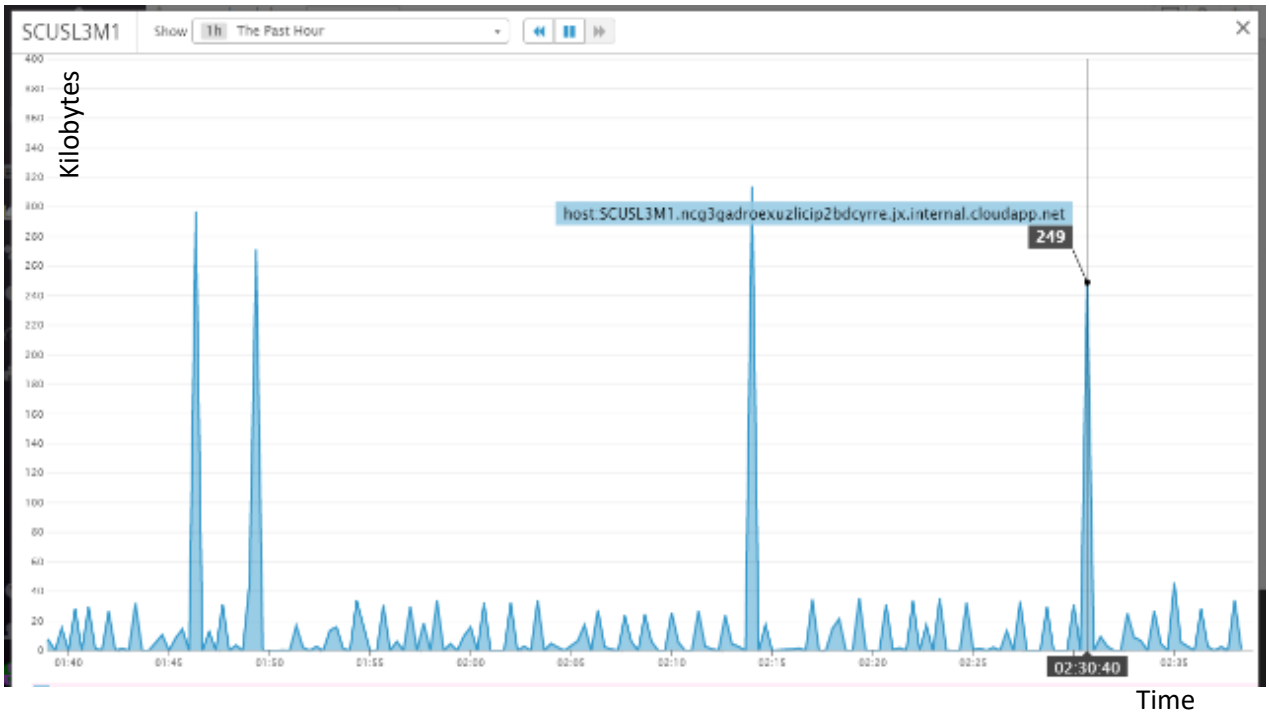

Figure 11. SCUSL3M1

We can observe from "SCUSL2M1" (Figure 10) and "SCUSL3M1" (Figure 11) where only request from "SCUSL1M1" was sent but not from the other two devices as the result was already saved in the second layer while processing for "SCUSL2M1". In the main server, request received and requested at 2:36:00am was not sent from "SCUSL1M2" and "SCUSL1M3" but from North Central US which is described below. So, "SCUSL1M2" and "SCUSL1M3" got the service directly from the second & third layer.

4.1.1.5 ***Result Graph of North Central US***

For "NCUSL1M1", the service needed to be requested 2:36:00am and received through all the layers "NCUSL2M1", "NCUSL3M1" and "CUSMAIN" at 2:36:00am (Figure 14,15), since that was not previously requested by devices within that region. It follows exactly the same procedure as South Central US.

Figure 12. NCUSL1M1

The figures show network usage in bytes (y-axis) vs time (x-axis) graphs where its clearly seen that in South Central US layer two VM has network usage of a size of 305bytes and layer one and layer three VMs with a usage of 249bytes.

4.1.1.6 Comparison between Traditional and Proposed Infrastructure algorithms (Datadog)

As mentioned before, two experiments were conducted in two different test environments. Among them, one represented our proposed infrastructure and the other one represented the conventional infrastructure. For the sake of computing the data transactions between the VMs and main cloud server and comparing our proposed and present infrastructure, same service was requested from four end-devices of different regions as shown in the table 1.

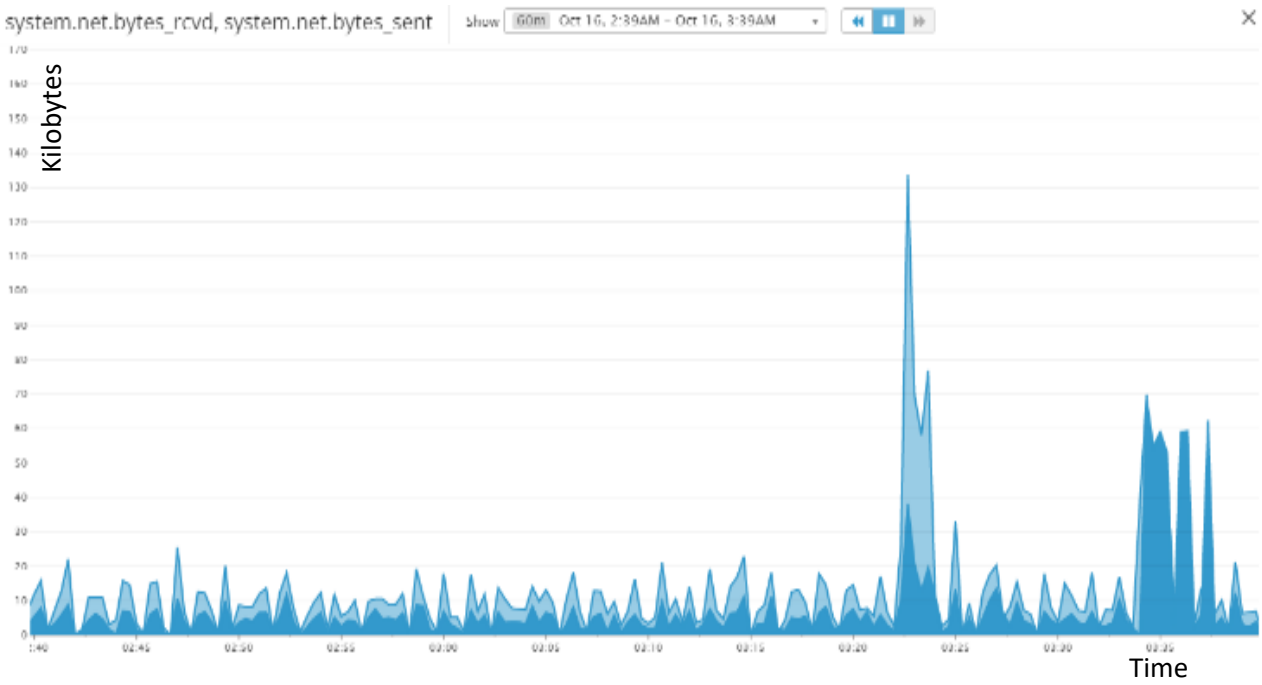

Figure 13. Conventional Infrastructure Result

After finishing the data processing, our result was projected through different graphs. Figure-13 shows the results of the conventional infrastructure where four devices have requested for the same service in between 3:33:00am to 3:38:00am. If we notice on the graph, we can observe that the total amount of data both received and sent, shows a constant data consumption.

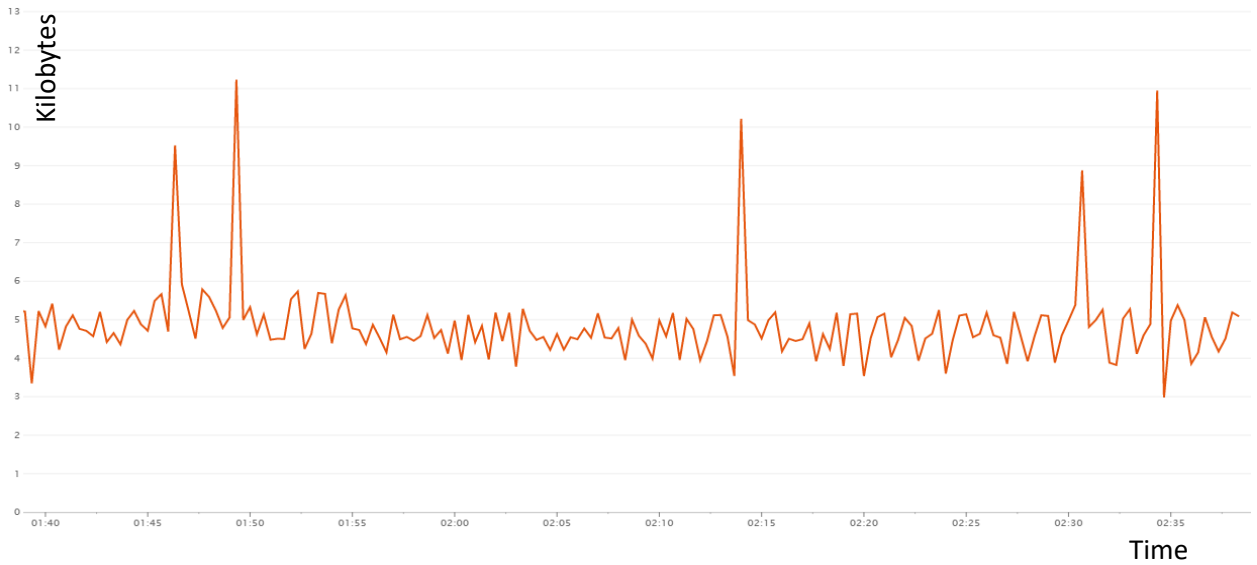

Figure 14. Fog Model's Total Received Data

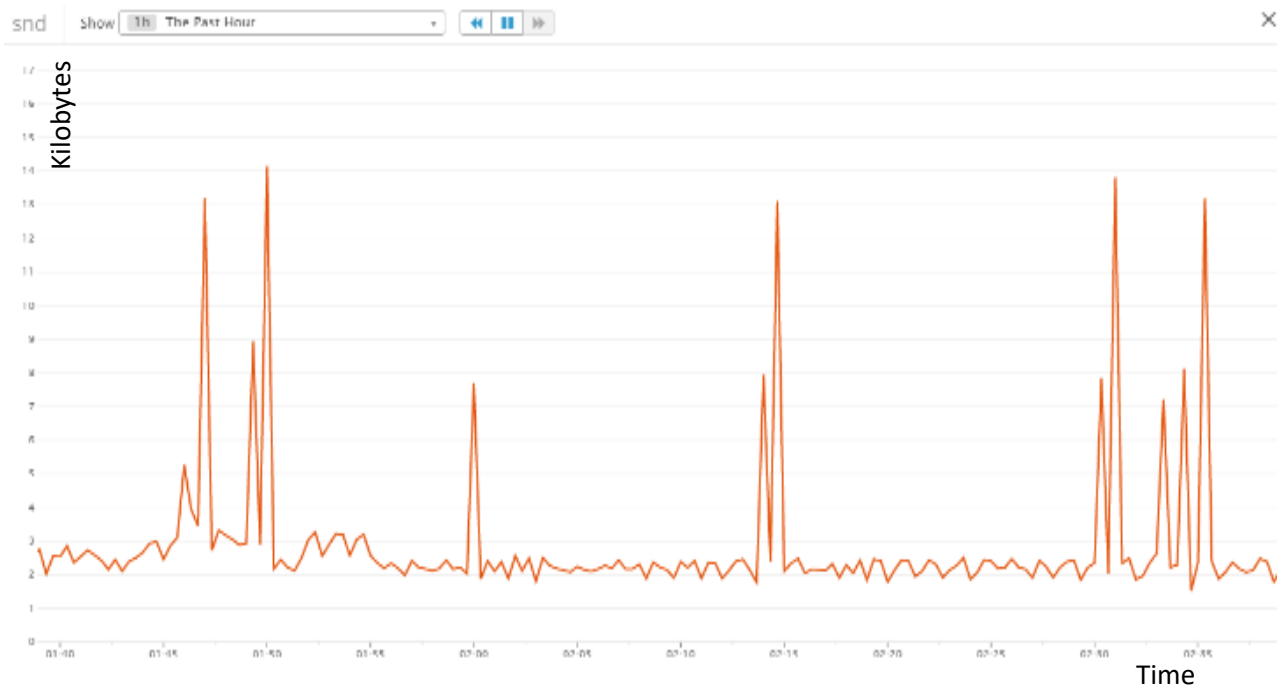

Figure 15. Fog Model's Total Sent Data

On the contrary, Figure-14 (representing total received data) and Figure-15 (representing total sent data) symbolizes our infrastructure where the same scenario was imposed, in between 2:31:00am to 2:36:00am. As indicated before, from the kilobyte/time graph we can see that there is a big drop of data consumption in the middle both while receiving and sending data. This data was recorded in at most 15 second interval, which gives this inconsistent growth of the graph. Now, comparing the graphs we can distinctly comprehend that our proposed infrastructure has a very low amount of network usage as it has a highest usage of 14kb to 11kb where as in the conventional infrastructure it reaches 70kb to 60kb within that time limit.

From these results of the described algorithms along with the comparison with present infrastructure, it can be ensured that our proposed infrastructure surpasses the

traditional one in less traffic along with less bandwidth, reliability through trust management by providing token authentication and heterogeneity maintaining the Quality of Service(QoS).

4.1.2 ***Result Graph Using Azure VM Monitoring:***

4.1.2.1 ***Graphs of Proposed Infrastructure***

At the very beginning one of our end device SCUSL1M1 requested for a ‘service x’ at around 9:30 am and we can observe that as a rise in NETWORK OUT graph of SCUSL1M1 in figure 16 which was received by SCUSL2M1 of layer two(SOA) at the same time as a rise in NETWORK IN graph of SCUSL2M1 in figure 17. Next, because of not having the service SCUSL2M1, it forwarded the service request (rise in NETWORK OUT graph of SCUSL2M1 in figure 17) to SCUSL3M1 which is located in layer three (Agent Based SOA) as a checker agent (rise in NETWORK IN graph of SCUSL3M1 in figure 17). Later, when the service was not even found by SCUSL3M1, it sent a request for the service to the fourth layer, the main server (rise in NETWORK OUT graph of SCUSL3M1 in figure 17 and rise in NETWORK IN graph of CUSMAIN in figure 19). Finally, the response of ‘service x’ is forwarded back to the end device SCUSL1M1 via third and second layer which can be observed through a rise in NETWORK IN graph of SCUSL1M1 in figure 16.

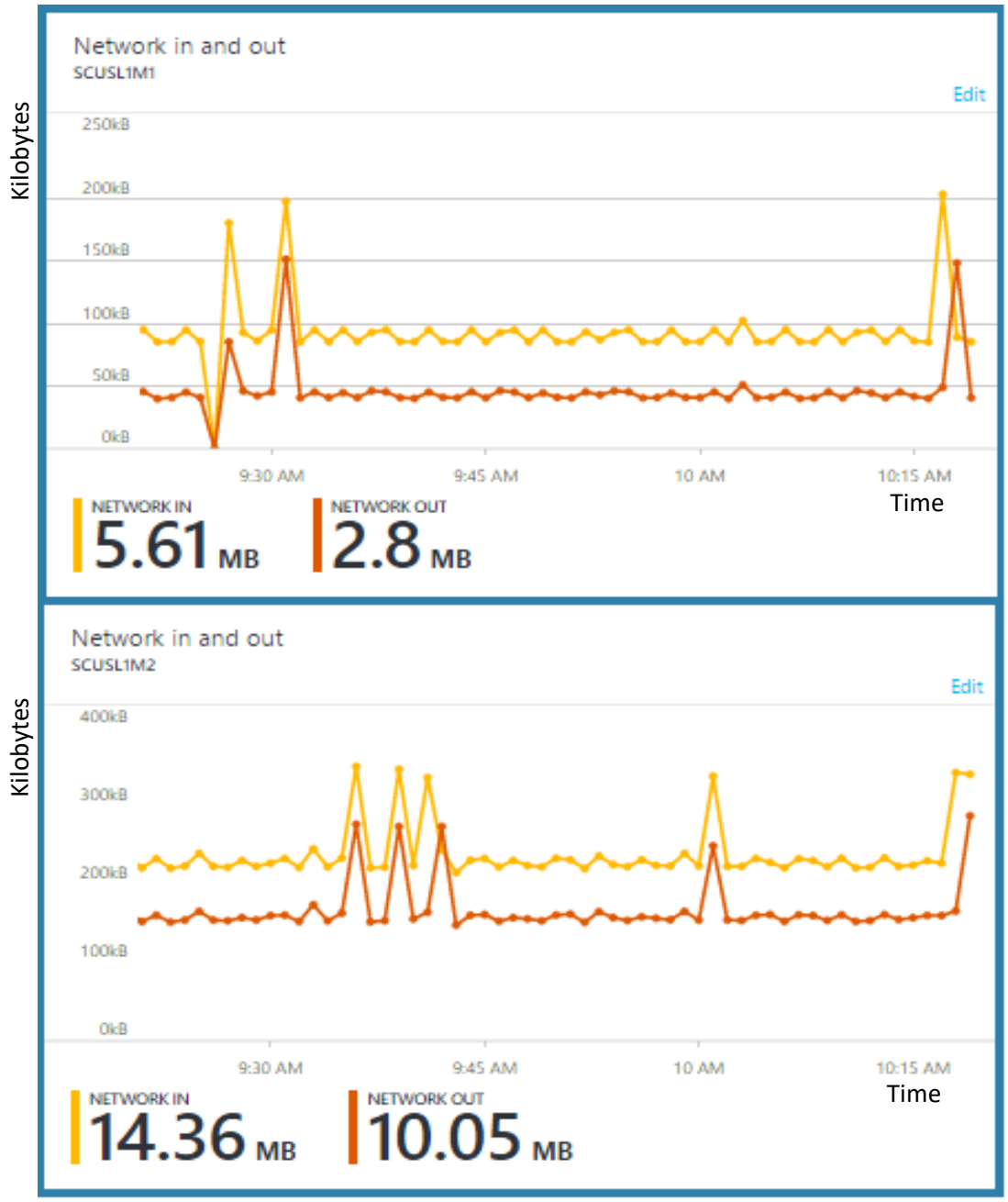

Figure 16. Graphs of SCUSL1M1 and SCUSL1M2

From another perspective, we can observe that three service requests were sent by SCUSL1M2 between 9:35am to 9:45am where first two requests were for 'service x' and the last one was 'service y' (rise in NETWORK OUT graph of SCUSL1M2 in figure 16) which were received by SCUSL2M1 in second layer (rise in NETWORK IN graph of SCUSL2M1 in figure 17) at the same time. Now we can notice from

SCUSL3M1 of Figure-17 that request for only ‘service y’ was received by the checker agent at 9:45am as it was not found in layer two but there is no edge for ‘service x’ in the NETWORK IN graph of SCUSL2M1 in Figure-17. The reason behind it is, ‘service x’ was already requested earlier by SCUSL1M1 which was saved in the second layer while fetching from the main server. Therefore, this lowers the data traffic towards the main server and causes lower latency for QoS as well.

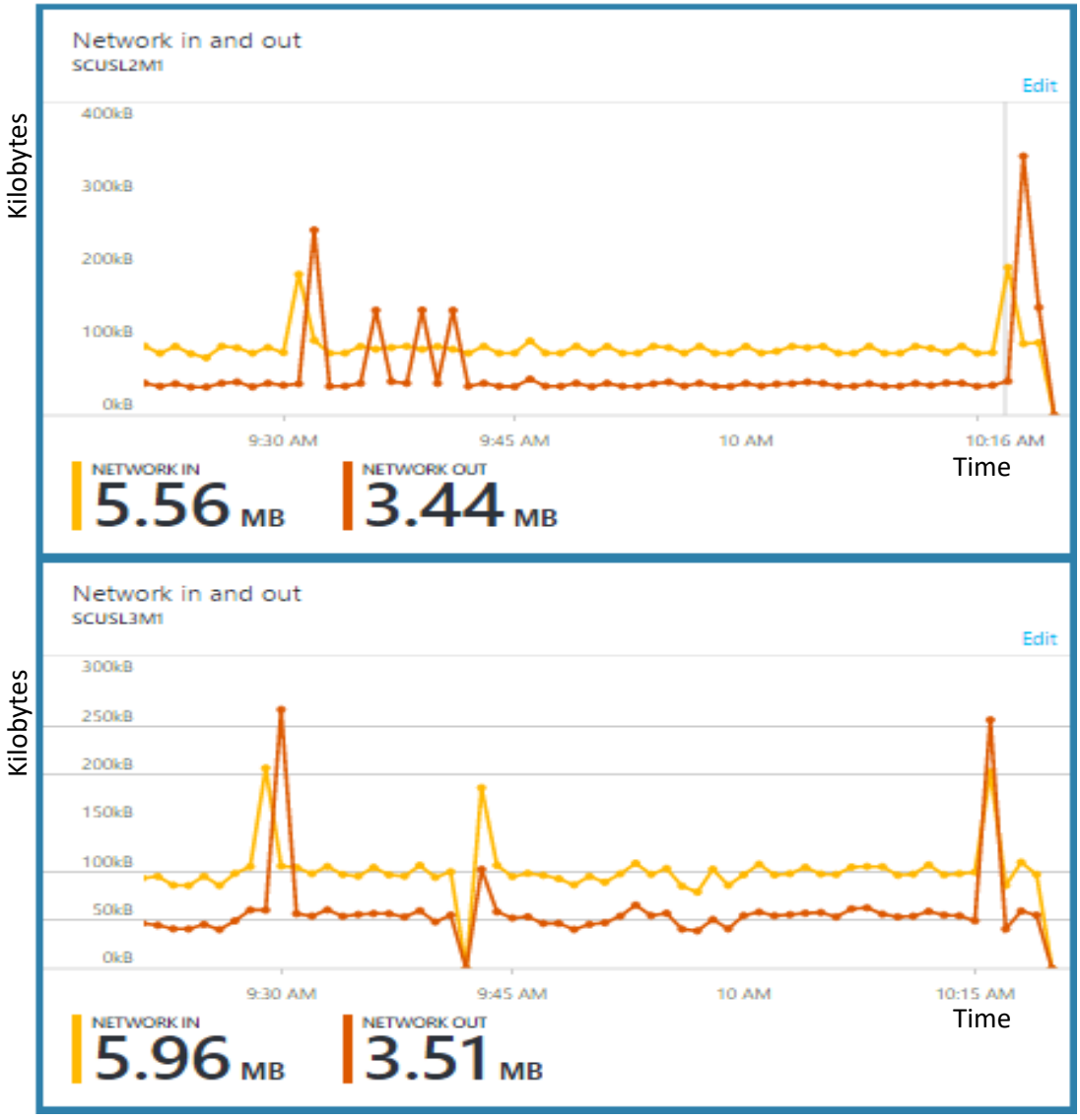

Figure 17. Graphs of SCUSL2M1 and SCUSL3M1

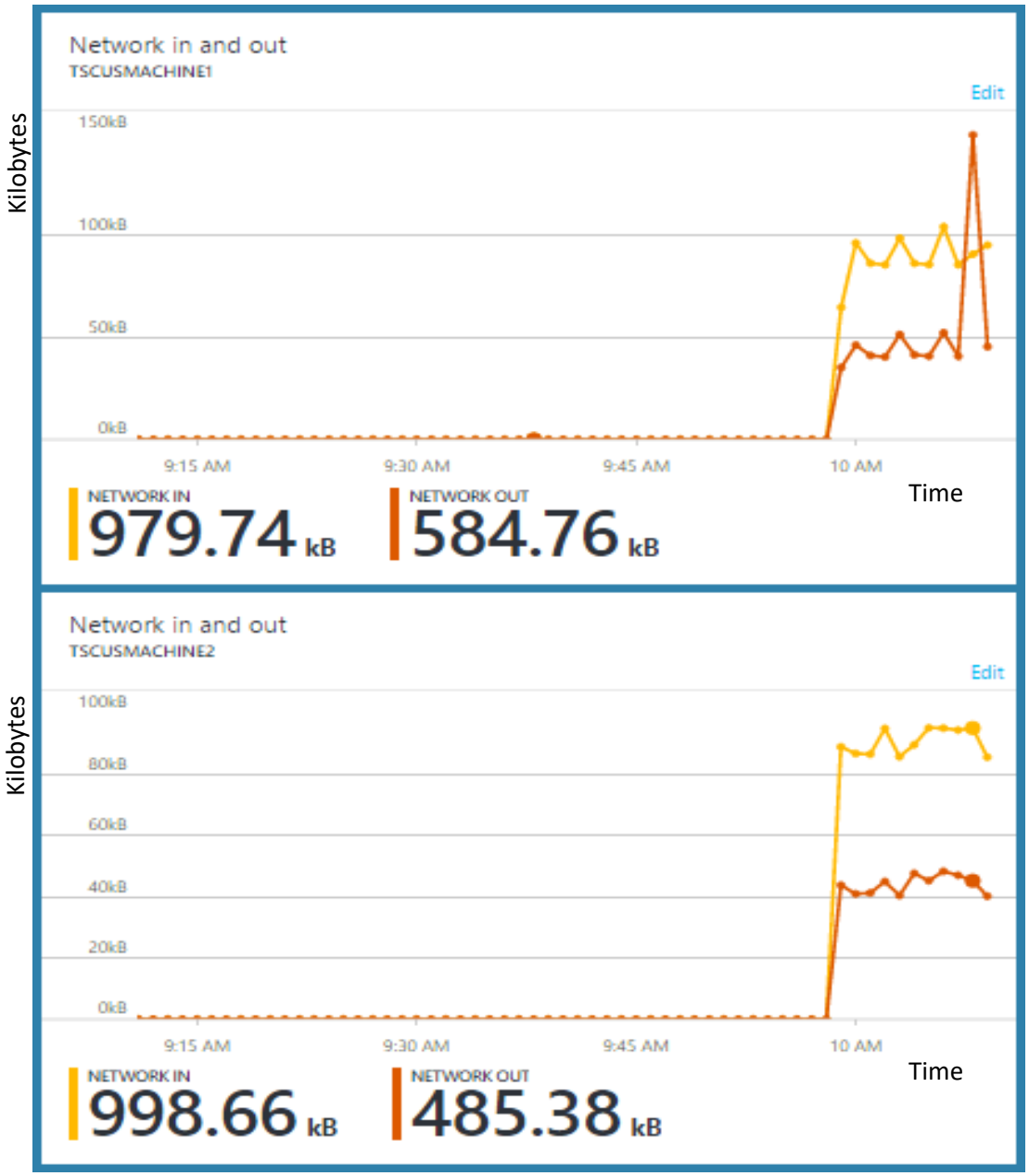

Figure 18. Graphs of TCUSMACHINE1 and TCUSMACHINE2

4.1.2.2 Graphs of Traditional Infrastructure

In the beginning the end devices in traditional infrastructure, TSCUSMACHINE1 and TSCUSMACHINE2 requested for ‘service x’ each three times. The requests were forwarded to the main server (rise in NETWORK OUT graph in figure 18) at around 9:57 am. After that, all of the responses were sent by the main server and we

can see rise in the NETWORK IN graph of both TCUSMACHINE1 and TCUSMACHINE2 at that time in Figure-18. So, the same ‘service x’ was fetched twice from the main server for the three requests at the same time.

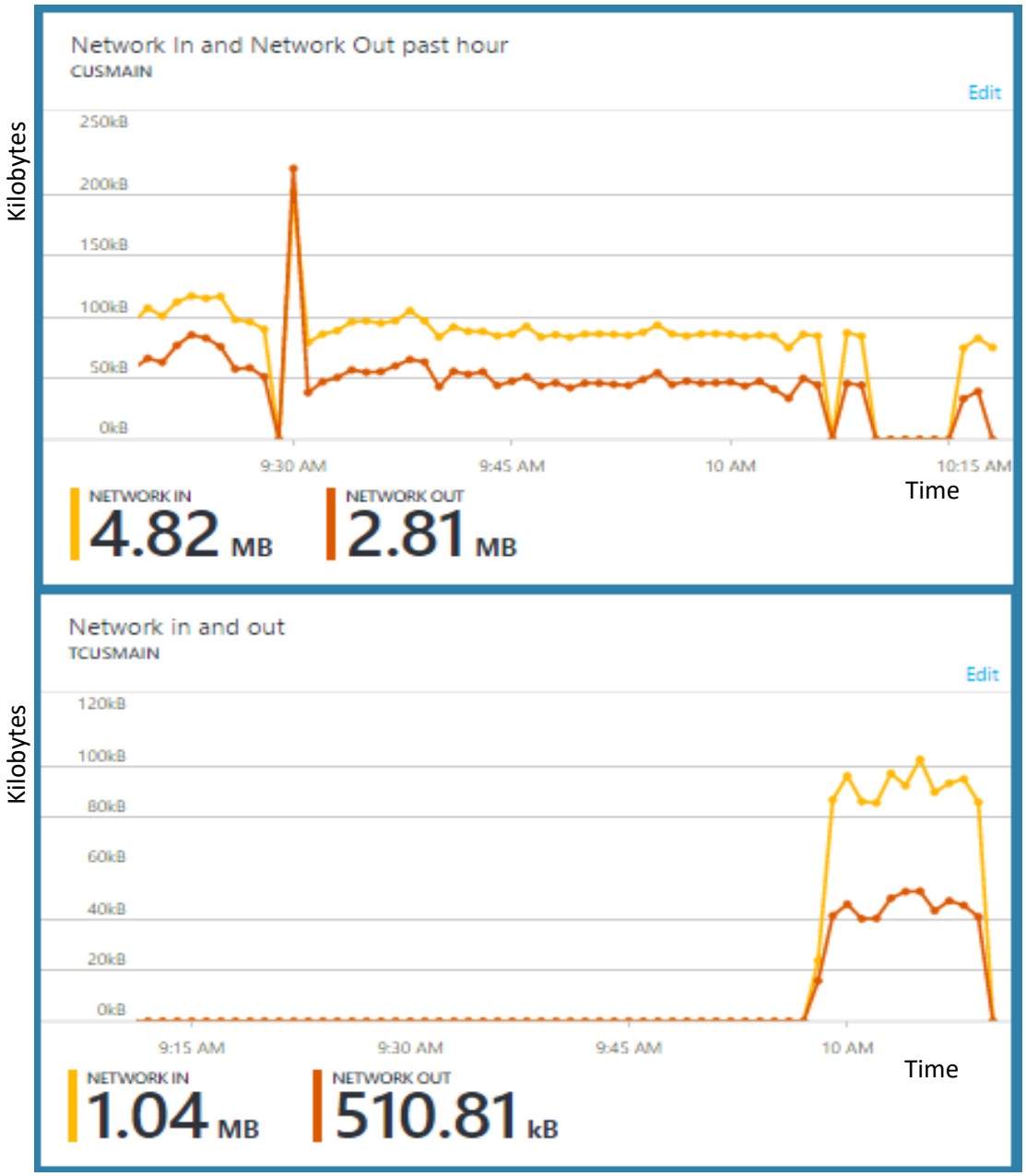

Figure 19. Graphs of SCUSMAIN vs TCUSMAIN

4.2 Efficiency of Our Infrastructure

In the graph given below, we plotted the amount of data consumed by each layer of the infrastructure for each requests. In our implementation chapter, we requested for two types of services (A and B) as 6 requests in total. We can observe the increase of requests leads to lower data consumption eventually with passing time. We have generated only 6 requests but in future when 20 billion devices will be involved with the IoT infrastructure, the number of data consumption will get lower because the Layer 3 Agents will get matured with supplementary of time.

Table 2. Data Consumption

Requests	SCUSL1M1		SCUSL1M2		SCUSL2M1		SCUSL3M1		CUSMAIN		Total
	in	out	in	out	in	out	in	out	in	out	
A1	192.28	153.31			256.6	208.29	122.94	115.41	202.34	221.4	1472.57
A2			452.01	370.7	249.22	95.9	97.34	54.03	0	10	1329.2
B3	148.3	203.53			356.48	98.53	202.83	256.38	82.66	39.24	1387.95
B4			251.33	277.32	144.56	91	96.61	56.63	12	11	940.45
A5			247.65	274.5	144.84	90.79	10.1	0.56	0	15	783.44
B6			232.05	274.45	144.41	90.94	12.33	11.2	19	0	784.38

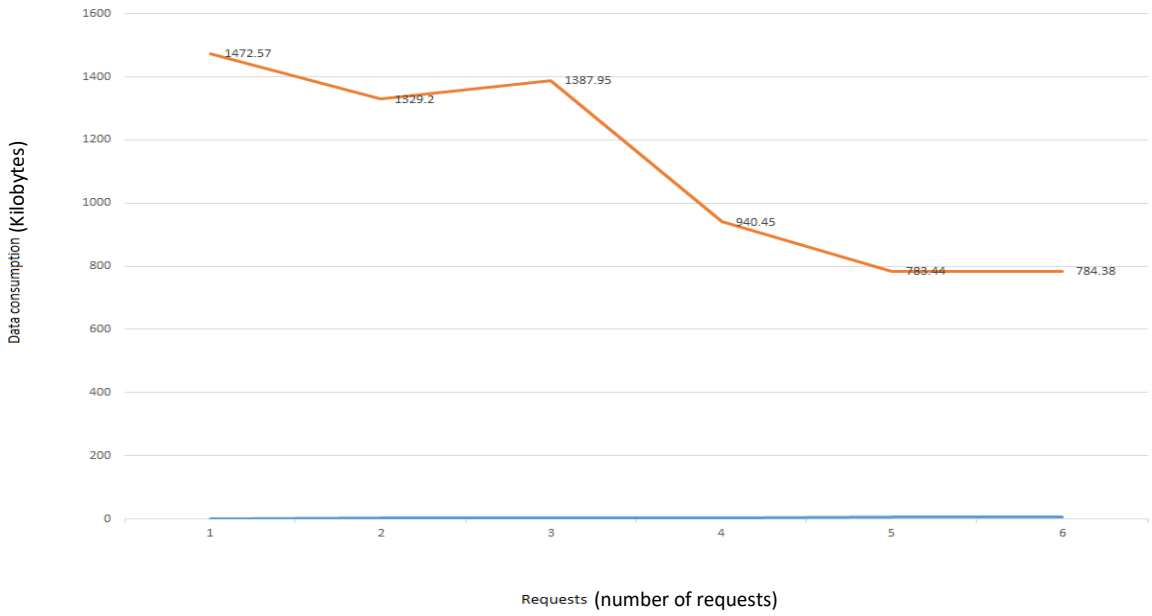

Figure 20. Data Consumption vs. Requests Graph

Chapter 5

CONCLUSION

Traditional approach deals with direct communication to the cloud in order to work with IoT devices which inefficient as we have already discussed. Our infrastructure relies on Fog computing model where nodes neighboring to the IoT devices are used for calculating and providing data to the local nodes. Our infrastructure supports a light weighted SOA which will be running in the fog nodes, that will provide services to the IoT devices locally as the second layer of our distributed model. For complex service compositions, the third layer will be providing agent based services as artificial intelligence where the more devices will be involved with our infrastructure, the more mature the agents of the third layer will become over the time. In addition, monitoring the traffic of the cloud channels, data is periodically synced with the cloud where the period is determined by the agent. Moreover, the third layer can implement miniature versions of services like stream analytics, bus services, machine learning and many more as services. That makes our infrastructure more flexible, proficient and versatile from the ongoing researches regarding fog computing and IoT.

Some other benefits of using this model will be, firstly, working as local backup of second layer can maintain the data transactions even if the main server gets down which results in no data loss or jeopardized data communication. Secondly, businesses can grow based on the infrastructure. For example, big companies which needs to serve IoT services to their clients can deploy second, third layer devices and they can provide other companies to use their nodes for their services based on some business model. This way users can get their services easily and this will also open up new opportunities for companies to grow. It can be deployed on the current growing infrastructure without changing the way devices work at present with

addition of trust management, low latency, less bandwidth preserving the Quality of Service(QoS).

5.1 FUTURE CHALLENGES

As IoT is relatively new in the growing world of technology, various obstacles are yet to be confronted by the researchers. In our proposed infrastructure, we attempted to cover all the current obstacles by giving it a novel solution. Theoretically, this proposed model will be sufficiently effective in robust and secured data transmission across the Internet through different geographical location. Yet some sectors are untouched for IoT. For example, load balancing of fellow agents.

REFERENCES

[1] Amol Dhumane, Rajesh Prasad, Jayashree Prasad. 2016. *Routing Issues in Internet of Things: A Survey*. In Proceedings of the International MultiConference of Engineers and Computer Scientists 2016, Hong Kong, vol. 1.

[2] B. Azimdoost, H. R. Sadjadpour and J. J. Garcia-Luna-Aceves. 2013. *Capacity of Wireless Networks with Social Behavior*. In IEEE Transactions on Wireless Communications, vol. 12, no. 1, pp. 60-69. DOI= http://dx.doi.org/10.1109/TWC.2012.121112.111581.

[3] O. Bello and S. Zeadally. 2016. *Intelligent Device-to-Device Communication in the Internet of Things*. In IEEE Systems Journal, vol. 10, no. 3, pp. 1172-1182. DOI= http://dx.doi.org/10.1109/JSYST.2014.2298837.

[4] F. H. Bijarbooneh, W. Du, E. C. H. Ngai, X. Fu and J. Liu. 2016. *Cloud-Assisted Data Fusion and Sensor Selection for Internet of Things*. In IEEE Internet of Things Journal, vol. 3, no. 3, pp. 257-268, June 2016.DOI= http://dx.doi.org/10.1109/JIOT.2015.2502182.

[5] K. P. Birman, R. van Renesse and W. Vogels. 2001. *Spinglass: secure and scalable communication tools for mission-critical computing*. In DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings, Anaheim, CA, vol. 2, pp. 85-99. DOI= http://dx.doi.org/10.1109/DISCEX.2001.932161.

[6] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta and P. Popovski. 2014. *Five disruptive technology directions for 5G*. In IEEE Communications Magazine, vol. 52, no. 2, pp. 74-80. DOI= http://dx.doi.org/10.1109/MCOM.2014.6736746.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. *Fog computing and its role in the internet of things*. In Proceedings of the first edition of the MCC

workshop on Mobile cloud computing (MCC '12). ACM, New York, NY, USA, 13-16. DOI= http://dx.doi.org/10.1145/2342509.2342513.

[8] 2014. Chapter 1: Service Oriented Architecture (SOA). Msdn.microsoft.com. Retrieved on June 30, 2016.

[9] Cisco. 2015. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Retrieved on July 10, 2016.

[10] Devi T., Ganesan R. 2015. *Platform-as-a-Service (PaaS): Model and Security Issues*. In TELKOMNIKA Indonesian Journal of Electrical Engineering, vol. 15, no. 1, pp. 151-161.

[11] S. K. Datta and C. Bonnet. 2014. *Smart M2M Gateway Based Architecture for M2M Device and Endpoint Management*. In Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing(CPSCom), IEEE, Taipei, 2014, pp. 61-68. DOI= http://dx.doi.org/10.1109/iThings.2014.18.

[12] S. K. Datta, C. Bonnet and N. Nikaein. 2014. *An IoT gateway centric architecture to provide novel M2M services*. In Internet of Things (WF-IoT), 2014 IEEE World Forum on, Seoul, 2014, pp. 514-519. DOI= http://dx.doi.org/10.1109/WF-IoT.2014.6803221.

[13] M. Eisenhauer, P. Rosengren and P. Antolin. 2009. *A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems*. In 2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, Rome, 2009, pp. 1-3. DOI= http://dx.doi.org/10.1109/SAHCNW.2009.5172913.

[14] Fortino G, Russo W. 2013. *Towards a Cloud-assisted and Agent-oriented Architecture for the Internet of Things*. InWOA@ AI* IA 2013 Nov 25, pp. 60-65.

[15] R. Garcia-Castro, C. Hill, and O. Corcho. 2011. *SemserGrid4Env Deliverable D4.3 v2 Sensor network ontology suite*.

[16] Hansen K. M., Zhang W. and Soares G. 2008. *Ontology-Enabled Generation of Embedded Web Services*. In Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering, Redwood City, San Francisco Bay, USA, pp. 345--350.

[17] P.R. Jaiswal. 2014. *Platform-as-a-Service (PaaS): Model and Security Issues*. In International Journal of Computer Science and Mobile Computing, vol. 3, no. 3, pp. 707-711. DOI= http://dx.doi.org/10.11591/telkomnika.v15i1.8073.

[18] Khan NN. 2015. *Fog Computing: A Better Solution for IoT*. In International Journal of Engineering and Technical Research, vol. 3, no. 2, pp. 298-300.

[19] M. K. Kiskani, H. Sadjadpour and M. Guizani. *Social interaction increases capacity of wireless networks*. In 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), Sardinia, 2013, pp. 467-472. DOI= http://dx.doi.org/10.1109/IWCMC.2013.6583603.

[20] N. S. Patel, Prof. Rekha B.S. 2014. *Software as a Service (SaaS): Security issues and Solutions*. In International Journal of Computational Engineering Research, vol. 4, no. 6, pp. 68-71.

[21] N. Peter. 2015. *FOG Computing and Its Real Time Applications*. In International Journal of Emerging Technology and Advanced Engineering.

[22] 2012. Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. ACM, New York, NY, USA.

[23] Y. Qin, N. Falkner, S. Dustdar, H. Wang. 2014. *When Things Matter: A Data-Centric View of the Internet of Things*. In CoRR. DOI= http://arxiv.org/abs/1407.2704.

[24] Rodríguez-Valenzuela S., Holgado-Terriza J.A., Gutiérrez-Guerrero J.M. and Muros-Cobos J.L. *Distributed Service-Based Approach for Sensor Data Fusion in IoT Environments*. In Sensors 2014, vol. 14, no. 10, pp. 19200-19228. DOI= http://dx.doi.org/10.3390/s141019200.

[25] S. Subashini, V. Kavitha. 2011. *A survey on security issues in service delivery models of cloud computing*. In Journal of Network and Computer Applications, vol. 34, no. 1, pp. 1-11. DOI= http://dx.doi.org/10.1016/j.jnca.2010.07.006.

[26] The OSGi Service Platform, OSGi Alliance, http://ww.osgi.org. Retrieved on July 25, 2016.

[27] C. Villalonga, Ed. 2010. D2.5 *Adaptive and Scalable Context Composition and Processing, Public SENSEI Deliverable*.

[28] H. Zhang and L. Zhu. 2011. *Internet of Things: Key technology, architecture and challenging problems*. In Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on, Shanghai, pp. 507-512. DOI= http://dx.doi.org/10.1109/CSAE.2011.5952899.