

Advanced Task Scheduling Algorithm for IoT Based FOG Communication Model

by

Zaber Mohammed

16301166

Riham Chowdhury

16301167

Stanley Dip Rozario

16301203

Sayed Bin Amirul Sakin

16301016

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
April 2020

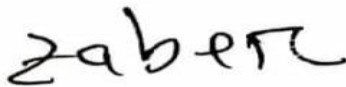
© 2020. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing the degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material that has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all the main sources of help.

Student's Full Name & Signature:



Zaber Mohammed
16301166



Riham Chowdhury
16301167



Stanley Dip Rozario
16301203



Sayed Bin Amirul Sakin
16301016

Approval

The thesis titled “Advanced Task Scheduling Algorithm for IoT Based FOG Communication Model” submitted by

1. Zaber Mohammed (16301166)
2. Riham Chowdhury (16301167)
3. Stanley Dip Rozario (16301203)
4. Sayed Bin Amirul Sakin (16301016)

Of Spring, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on April 7, 2020.

Examining Committee:

Supervisor:
(Member)



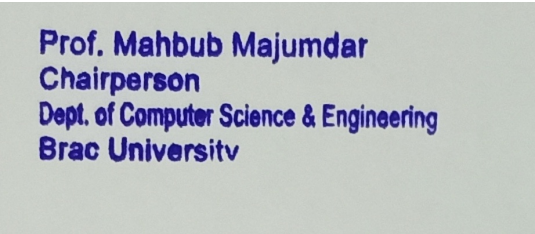
Amitabha Chakrabarty, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)



Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)



Prof. Mahbub Majumdar
Chairperson
Dept. of Computer Science & Engineering
Brac University

Mahbubul Alam Majumdar, PhD
Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement (Optional)

We, hereby declare that this thesis is based on the results we obtained from our work. Due acknowledgement has been made in the text to all other material used. This thesis, neither in whole nor in part, has been previously submitted by anyone to any other university or institute for the award of any degree.

Abstract

Internet of Things (IoT) is hugely dependent on Cloud Computing. Cloud computing uses a high degree of polymerization calculation mode and so it cannot ensure effective use of resources like computing, storage, etc. FOG computing is a developing paradigm that broadens computation, communication and storage facilities towards the edge of a network. It is used to improve efficiency along with the reduction of transmitted data for processing to the cloud. Although the primary aim of FOG computing is to improve the processing speed of cloud computation, it has many challenges such as task scheduling, resource allocation, security, etc. Among these challenges handling incoming requests to improve latency and throughput is one of the crucial factors. As a solution, the proposed model uses a multi-layered FOG model in which tasks are scheduled on the basis of priority based on the request type to increase the efficiency of the current FOG model. Firstly, the proposed model creates a rule list based on the user's request priority. While creating the rule list the model will use advance caching mechanism based on the request type in the different layers to improve latency and throughput. When a user sends data to the FOG, it finds its configured layer of the FOG cloud on which the data will be processed. Stored data will be loaded in the corresponding layer based on packets' priority to make the computation faster. The proposed model has been simulated using Microsoft Azure. In the simulation, the inbound data after caching was more than 70 MB per 30 seconds wherein the traditional cloud, the inbound data rate was around 30 MB per 30 seconds. Therefore, after caching the data, the model performed twice faster than the traditional cloud.

Keywords: FOG computation, Task Scheduling, Caching mechanism.

Acknowledgement

Firstly, and foremost, we would like to thank our Almighty for enabling us to conduct our research, give our best efforts and complete it. Secondly, we would like to thank our supervisor Amitabha Chakrabarty sir for his feedback, support, guidance and contribution in conducting the research and preparation of the report. He encouraged us to conduct the research, give guidance to us and always were present to offer any help we could ask for. We are grateful to him for his excellent supervision to successfully conduct our research. We are also grateful to Sayed Erfan Arefin for his valuable guidance. We are thankful to Ahnaf Atef Choudhury to help us getting the Microsoft Azure account. In addition, we like to extend our gratitude to our family and friends, who guided us with kindness and gave inspiration and with their suggestions. Last but not the least, we thank Brac University for providing us the opportunity of conducting this research and for giving us the chance to complete our Bachelor's degree.

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iii
Abstract	iv
Dedication	v
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
1 Introduction	1
1.1 Overview of cloud computing and Fog computing on IoT	1
1.2 Thought Behind Working on Fog Computing	2
1.3 Problem Statement	2
1.4 Research Contribution	2
1.5 Research Objectives	3
2 Related Work	4
2.1 Literature Review:	4
2.2 Layered Architecture:	6
2.2.1 Layer 1 (End Device):	6
2.2.2 Layer 2 (Lower Middleware):	6
2.2.3 Layer 3 (Upper Middleware):	6
2.2.4 Layer 4 (Cloud):	7
2.3 Application of FOG for IoT:	7
3 Proposed Model and Experimental Setup	9
3.1 Proposed Model:	9
3.2 Caching Mechanism:	15
3.3 Experimental setup	18
3.3.1 General setup:	18
3.3.2 Layer Configuration:	23
3.3.3 Client-Side Programming:	24
3.3.4 Server-Side Programming:	24

3.3.5	Layer 1 configuration:	26
3.3.6	Layer 2 configuration:	27
3.3.7	Layer 3 configuration:	29
3.3.8	Layer 4 configuration:	31
4	Result and Analysis	32
4.1	Result	32
4.1.1	UDP Type Data Transfer:	32
4.1.2	TCP Type Data Transfer:	34
4.1.3	Cloud Server Data Transfer:	37
4.2	Analysis:	39
5	Conclusions and Future Work	42
5.1	Conclusions:	42
5.2	Future Work	43
	Bibliography	46

List of Figures

2.1	Comparison of Cloud and FOG model	7
3.1	Block Diagram of Proposed Model's Infrastructural Setup	10
3.2	Layered Architecture	11
3.3	Flow chart of Processing Sent Request from End Device	13
3.4	Flow Chart of Reply Sent from Cloud	14
3.5	Lazy Population of Caching Mechanism	16
3.6	Flowchart of Caching Mechanism	17
3.7	Four Layers combined as resource group	19
3.8	Resources of End Devices	20
3.9	Resources of Lower middleware	21
3.10	Resources of Upper middleware	22
3.11	Resources of Cloud	22
3.12	Working principle of socket API	25
4.1	Inbound Data graph for UDP type request (End Device)	32
4.2	Outbound Data Graph of Layer 2 server	33
4.3	Inbound Data graph for TCP type Request (End Device)	35
4.4	Outbound Data Graph of Layer 3 server	36
4.5	Inbound graph of End Device for the traditional cloud model	37
4.6	Outbound Data graph for traditional cloud server	38
4.7	Inbound Data Comparison Graph of End Device for Traditional Cloud vs Proposed Model	40
4.8	Inbound Comparison Graph for TCP vs UDP Request of Proposed Model	41

Chapter 1

Introduction

1.1 Overview of cloud computing and Fog computing on IoT

IoT devices have introduced a new paradigm in the field of technology. Billions of physical devices are collecting and sharing data via the connection of the Internet which is the main concept of IoT. The real success of IoT lies in the utilizing of gathered information from the environment provided to the cloud by the connected devices rather than connecting billions of physical devices. Hence, the IoT infrastructures must have the capability to deal with a large amounts of things that are distributed in different geographical position and the produced data may require real-time analytics and data aggregation at different levels with the lowest possible latency [1]. Connectivity provided by cloud services helps devices by reaching out and acting on the world to provide valuable information. The devices associated are not restricted to specific devices that any organizations possess, however, these devices can range from individual devices each individual uses to the enormous ones through Internet cloud services. Cloud computing uses remote servers and computers across the Internet in replace of local servers and computers to store and manage data along with performing data operation. It offers delivery services directly over the Internet by providing storage, databases, software, applications, network, servers, etc. to its clients. At present, the improvement and implementation of scalable Internet of Things applications and business model have created a revolution around the world that has been possible for the huge uses of IoT in the cloud. Cloud computing and IoT have emerged as very intently affiliated future Internet technologies with one providing the alternative a platform for success. Converging IoT and Cloud computing have created a platform where a huge number of benefits are found. Due to the convenience of IoT devices, usages and the number of IoT devices are increasing exponentially which results in the market worth of 19 trillion pro-t and 50 billion IoT devices by 2020 [2]. To support this huge market, it is quite difficult to rely on only cloud computing. In order to overcome the challenge, a new model called the FOG communication model has been incorporating with cloud computing. With the rising of FOG communication, the challenges of implementing FOG are increasing rapidly. Fog computing allows for the distribution of critical core functions like storage, communication, control, decision making and application services closer to the origination of data. FOG computing spans the continuum between the cloud and everything else. It makes fog computing, a

standard design and a necessary one for eventualities wherever latency, privacy, and alternative data-intensive issues a cause for concern.

1.2 Thought Behind Working on Fog Computing

The fog nodes can be deployed in any environment with a network connection. To process the services of the fog nodes, it has additional storage and processing system at the edges. In addition, the fog computing model is still a new one and does not have a proper structure. Hence, the maintenance process encounters some issues like privacy, security network management, placement of fog servers, delay in computing and energy consumption [3]. As a result, there is a huge scope for researchers to work on. Therefore, we set up our mind to work on the delay in computing to increase the throughput by reducing latency. Delays in computing generally occur due to data aggregation, resource over-usage reduces the effectiveness of services provided by the fog servers, causing delays in computing data [3]. Thus, task scheduling can provide an effective way of overcoming the delays to make the fog computing faster.

1.3 Problem Statement

As FOG computing is totally new to overcome the challenges of cloud computing is facing, different approaches and models of fog computing are being proposed. Several layers of FOG has been proposed to improve the communication model by efficient load model [4]. Moreover, IoT sensors are divided into different types based on the impact of data loss of different sensors on the total system [5]. FOG computation covers and supports the applications that require low and predictable latency which can not be achieved in cloud computation [6]. Furthermore, a huge data centre with cloud computing can be still used for deep analytics with the help of the FOG model [7]. In the report, the proposed model has utilized the hierarchy and packet classification based on request type to ensure load balancing in the communication model by scheduling tasks. In the model, the FOG communication model will balance the computation load by scheduling tasks in the different layers. The Fog computation model has divided into four different layers where layer 1 will be end-user, layer 2 will be lower middleware, layer 3 will be upper middleware and layer 4 will be the cloud. Figure 1 will illustrate the distribution of the layers in the system. The task scheduling approach will be applied on layer 2, layer 3 and layer 4. Packet classification has been used to identify the data type and based on that request type the tasks are scheduled on the mentioned layers. To do the packet classification, rules will be set on considering two parameters which are the impact of data loss in the system which means TCP type packets and the necessity of real-time data transfer for the best outcome of the system which denotes UDP type packets.

1.4 Research Contribution

We have proposed the model to design a futuristic system that will be able to handle the upcoming challenges in the huge IoT market in terms of efficient communication. To fulfill the motive, layers will be configured according to the algorithm which is

made of considering the two parameters which are UDP type request and TCP type request. Therefore, this proposed model will increase the efficiency of the traditional fog computing model by reducing the latency and increasing the throughput.

1.5 Research Objectives

The main objective of this research is to increase the efficiency of the IoT based FOG model by improving the FOG communication model. Thus, some of the proposed model objectives are given below:

- To develop a task scheduling algorithm for the FOG server.
- To decrease the latency of any request.
- To increase the throughput of the FOG server.
- To develop a more efficient caching mechanism for request handling.
- To decrease the IoT device's communication time and improve the response time for any request.
- Design the FOG layer to cache different types of data.
- To get a good idea about how the FOG communication model is going to be established in the future.
- Implement the proposed model in a simulated environment to compare the mechanism with the existing cloud computing model.

Chapter 2

Related Work

2.1 Literature Review:

Because of being emerging technology field, researchers are putting great effort in the FOG communication model to derive a suitable model. There is no standard architecture for the FOG computation model which creates a huge opportunity to explore different approaches [8]. Task scheduling on FOG servers is one of the great challenges among the challenges in implementing the FOG communication model. Different approaches considering different scenarios are being analyzed and simulated.

Authors focused on layered FOG communication models where FOG servers' are divided into two upper middleware and lower middleware between the end devices and cloud [9] [10] [11]. The main idea is that the lower middleware will be close to the end device and the upper middleware will be close to the cloud. The data will be searched starting from lower middleware and if not found it will search on upper middleware. If not found in upper middleware, it will search nearest upper middleware and then to the cloud. After that, the packet will be saved in lower middleware.

In another model, the authors focused on implementing a data mining approach for task scheduling algorithms [12]. The execution of the main algorithm consists of two consecutive algorithms which are the I-Apriori algorithm and the output of this algorithm works as the input of the TSFC (Task Scheduling in FOG Computing). Besides, another model has proposed a task scheduling algorithm where after receiving a task, the task is composed in different subtask in fog node and an algorithm is used to find out the optimal task sequence [13]. The main objective of finding this sequence is to ensure minimum use of resources and cost. Another model has adapted the approach of Greedy Knapsack-based Scheduling (GKS) [14]. The main idea is to dividing the task into equal sub-tasks and putting weight on the individual tasks. Then the author handles the problem as a knapsack problem's greedy approach. Besides, one model has divided the tasks of IoT into two categories which are near real-time and delay-tolerant [15]. Between these two types, near real-time is handled faster and real-time. This is done based on the activity different algorithms such as FCFS, Concurrent Strategy and Delay-priority are used. Another group of researchers' have proposed a model in which different clusters are used for performing any task [16]. The task is divided into multiple sub-tasks and then

distributed into appropriate clusters. This clustering and division of sub tasking is done considering parameters such as latency, power consumption, and cost. In another proposed model, the FOG servers are divided into three containers which are request evaluator, task scheduler and resource manager [17]. Request evaluator evaluates whether the request can be managed, task scheduler schedules the task and resource manager provides appropriate resources. In addition, for implementing optimal scheduling, another model has proposed heuristic-based algorithm for implementing task scheduling in FOG servers [18]. This method produces two Directed Acyclic Graphs which are Task Graph and Process Graph. Considering these two graphs, they determine appropriate task schedule. For determining the appropriate task schedule, they considered determining task priority and selecting the appropriate nodes for performing a task.

Moreover, FOG uses two types of communication models which are request-reply and publish subscriber [19]. Among these, request-reply model is based on client-server architecture which is one of the basic. On the other hand, the publish-subscribe model has a subscriber (client), broker (Middleware) and publisher (Cloud). We have followed the request-reply approach where there is the scope of publish-subscribe introduction in the future. Another approach for optimizing task scheduling is a joint optimization technique [20]. The authors focused on addressing 3 issues which are how to balance the workload on a client device and computation server, how to place the task to the storage device and how to balance I/O interrupt request. They used 2 types of servers which are storage server and computation server. Besides, interconnection among FOG nodes has also been proposed [21]. Here, the fog nodes are distributed based on different regions and a request can be handled either one or multiple regions based on the request. In another approach, the authors used a priority task scheduling algorithm [22]. This is an improved version of Efficient Resource Allocation (ERA). At first, the request is assigned to the nearest FOG server and then the FOG server sends the request to a priority queue. Moreover, a group of researchers has proposed MOSSO (Multi-Objective Simplified Swarm Optimization) for task scheduling [23]. Their main focus was on finding the relation between processing rate and cost per unit time of the processor.

Among these models, we have focused on enhancing the layered model architecture by utilizing packet classification. Our proposed model will utilize packet classification to search and store expected requests in a certain layer. As a result, it will increase the latency of any request and also decrease the throughput of requests by utilizing computational capabilities. In addition, our proposed model does not search for the specific requests among upper middleware. Therefore, our model requires less computational time for searching the packet in FOG layers before going to the cloud. For swapping stored values, our model has the used Least Time between Access algorithm with a lazy population caching mechanism.

2.2 Layered Architecture:

The proposed model follows request-reply model which is based on client-server architecture. Despite following request-reply model, there are scopes for upgrading this model to publish-subscribe model by modifying the FOG middlewares.

2.2.1 Layer 1 (End Device):

This layer consists of the end devices of various types. In IoT based model, most of the End Devices are different types of sensors and other smart IoT devices. Data packet will be requested from Layer 1 to Layer 2. Reply always will be received to Layer 1 from Layer 2 for all the packet types. For communication protocol, Layer 1 will use M2M (Machine to Machine) protocol. M2M protocol is suitable for devices to devices communication. Key applications for M2M protocols are connecting device to device and connecting device to service centers [8]. In our proposed architecture initially, communication is done using request-reply model but in Layer 2 and Layer 3 there is scope to develop publish-subscribe model. The M2M communication protocol is suitable for both cases. Besides, Layer 1 will deal with mostly SaaS and few PaaS services. The reason is that in Layer 1 most of the IoT Devices will use built-in software and some advance IoT devices will have the option to choose software and constrain on data sending properties. This layer will work as a client for request-reply model.

2.2.2 Layer 2 (Lower Middleware):

This layer is connected with Layer 1 and Layer 3. This layer is considered as Lower Middleware of FOG. The objective of Layer 2 is to receive a request from Layer 1 and check whether it can process the request or not. If it can process the request then it will send the reply to Layer 1 otherwise send the reply to Layer 3. This layer will consist of servers that will cache data and process data if it has the capability of processing the request. In this model, it will work as both client and server in client-server architecture. When Layer 2 receives any request from Layer 1 it works as a server in client-server architecture. At the same time, if it needs to extract the data from the upper layer, it sends a request to Layer 3 where it works as a client in client-server architecture.

2.2.3 Layer 3 (Upper Middleware):

This layer is situated between Layer 2 and Layer 4. This layer is considered as Upper Middleware of FOG. This layer receives a request from Layer 2 and sends a reply to Layer 2 if the request can be handled in this layer. If the request cannot be handled then this layer sends the request to Layer 4. Then it receives the reply from the cloud and sends it to Layer 2. Layer 3 also works as both client and server in a client-server architecture. To illustrate, when Layer 3 receives a request from Layer 2, it works as a server but when it sends a request to Layer 4 it works as a client.

2.2.4 Layer 4 (Cloud):

This layer is a traditional cloud service. The requests that have not been processed by Layer 2 and Layer 3 will be sent to the cloud. The request will come to cloud via Layer 3 and reply will also send to the Layer 3 from Cloud. Layer 4 is situated at the top of this hierarchical model. Layer 4 works as the only server in this architecture.

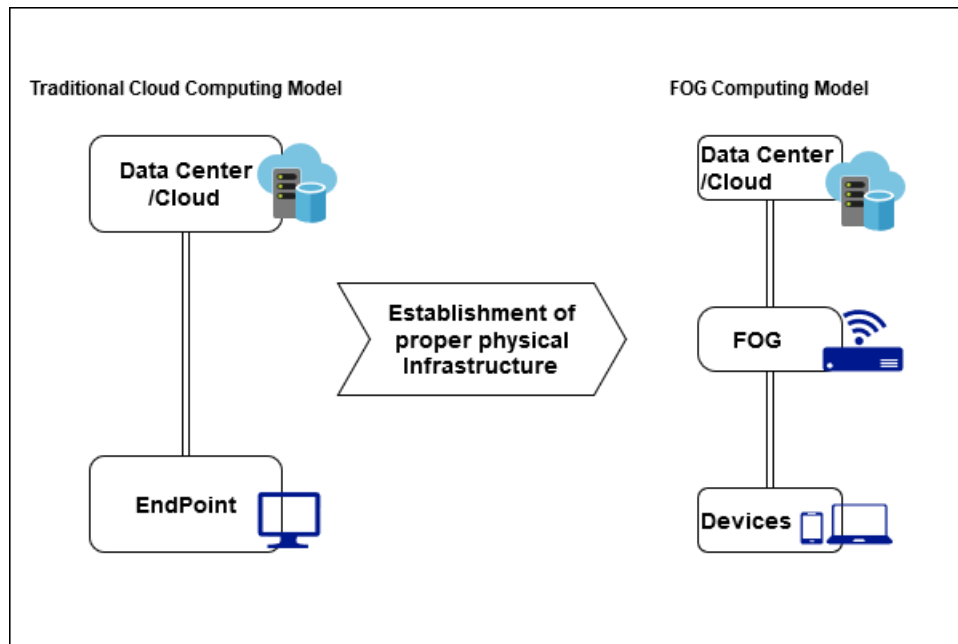


Figure 2.1: Comparison of Cloud and FOG model

2.3 Application of FOG for IoT:

Cloud is the upper layer and it stores all the information of a fog node and the cloud can store all the data in it. Thus the network congestion increases and thus the quality of the network service is also effected and latency is also increased. Fog computing provides an intermediary between these IoT devices and the cloud computing infrastructure that they connect to. The FOG is also able to analyze and process data closer to where the data is coming. The users can fetch the data from the fog nodes through the Internet Communication. Instead of hosting and working from a central cloud, the FOG system computes are each end device that is dealing with IoT [2]. The theory of FOG computation suggests the FOG provides more scalability and gives a clear concept of the network as multiple data points feed data into it [24]. There is too much data to handle in the cloud thus FOG reduces the need for bandwidth by not sending all the information to the cloud. Fog computing environment has broadly covered deployment to provide QoS for phones and motionless end devices [25]. The FOG computing can provide a better

quality of services in terms of delay, power consumption, and reduced data traffic over the Internet [2]. Moreover, FOG computing also provides better security by protecting fog nodes with the same policy and procedure areas of IT environments [25]. FOG computing can produce low-latency network connections between end devices. FOG computing is in the preliminary stages of being rolled out in formal deployments but various cases have been identified as potential scenarios for fog computing, for example, connected semi-autonomous cars, smart cities and smart grids [26]. Terabytes of data are created and sending all these to the cloud for storage and analysis thus, fog becomes a solution of inefficiency [27]. FOG computing also improves business activity since according to customers the FOG node supports flexibility as the nodes can join and leave the network at any time [26]. FOG computing is the perfect partner for IoT devices and cloud because it can help them in different functions of which they perform. This extends from an individual person to large firms only because of real-time analysis and monitoring [27].

Chapter 3

Proposed Model and Experimental Setup

3.1 Proposed Model:

To develop a task-scheduling algorithm for the FOG server, the main purpose is to decrease latency time and increase throughput. As mentioned earlier, the proposed model has focused on developing FOG architecture in 2 layers which are lower middleware (Layer 2) and upper middleware (Layer 3). In order to ensure effective communication and low latency data transfer with high throughput; we have incorporated packet classification with layered FOG communication model. On the Internet, while transferring the data, it has to be smashed into small pieces and each of the small pieces is sent solely. Those tiny portions of data are known as packets. Exchange of information among IP networks takes place in the form of packets. Packets are constructed by following a particular structure. In that structure, every protocol has a layer for a specific connection that is wrapped over the packets. On the other hand, packet classification is used for the pigeonholing packets. Therefore, we have used the concept to our layered architecture. The packets that belong to the same flow have to adhere to the rules that are defined earlier and actions are taken accordingly. There are multiple fields in the packets where packet classification can be done and also it can be done in a single field [28]. Here, the packet classification is needed in a single field. Based on the packet type, the packets are classified. we have mainly focused on two types of packets which are:

1. **UDP:** User Datagram Protocol is not a trustworthy transport system that is used to exchange data. The sent data is not guaranteed to be reached on the receiver's end but it only assures that the data is sent out on the medium. It is a connectionless protocol where the error-checking process along with recovery services are not necessary. Rather, it constantly sends datagrams to the receiver's end even if the receiver receives those or not. As a result, it is used to the information that is not sensitive but has to be transferred within the shortest possible time.

2. **TCP:** Transmission Control Protocol is a connection-oriented protocol that the networking nodes use while exchanging data. It is a reliable transport system. TCP provides an error-checking process and if the data corrupted it sends the particular portion of data again. Moreover, this protocol guarantees delivery of data by ensuring each packet is received on the receiver's end. In addition, the packets are sent using this protocol delivers maintaining the proper order. As a result, it is used to the information that is very sensitive and has to be transferred without any data loss where the time is not the main concern.

As lower middleware is placed closer to end devices (Layer 1), it will be able to communicate with lower middleware in less time. As a result, the latency will be less and throughput will be more between layer 1 and layer 2 comparing layer 1 with other layers. As the UDP type packet requires faster communication comparing TCP type packets, UDP type packets will have priority in Layer 1. After the Layer 1, Layer 2 comes and so TCP type packets will have priority in Layer 2. By packet classification, packet types will be determined and will assign to its corresponding layer. This will ensure less time for processing a response when a request is sent.

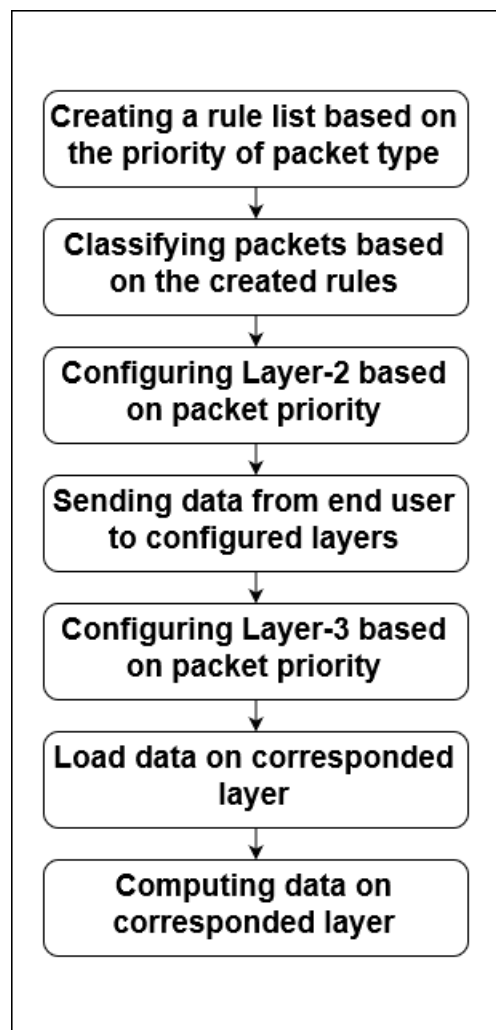


Figure 3.1: Block Diagram of Proposed Model's Infrastructural Setup

Figure 3.1 shows step by step process of configuring each layer of our proposed FOG server and how data will process to provide a faster response. Here, at first the different layers will be configured to handle or pass corresponded requests accordingly.

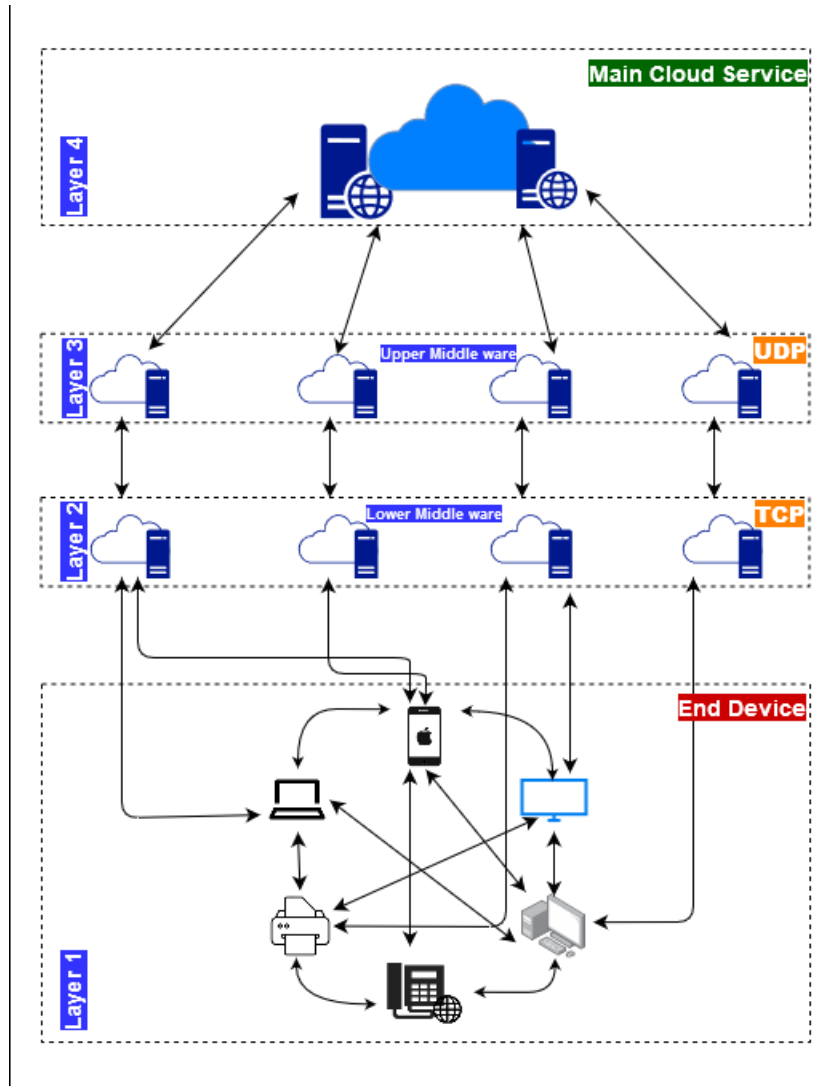


Figure 3.2: Layered Architecture

Figure 3.2 explains the model of Layered Architecture.

When a request is sent from Layer 1 to Layer 2, it will first check whether the packet is UDP type or TCP type. Therefore, when a request is sent, we can assume two cases:

1. Case I: The Requested packet is UDP Type

If the packet is UDP type then the request will send to Layer 2 and search for the related packet in that layer. Then it will take necessary actions and respond back to the end device or layer 1. If the requested packet is not found, the request will forward to layer 3. Then Layer 3 will check if it is TCP type or not and as the requested packet type is UDP, it will be forwarded to layer 4 which is the main cloud. Here, layer 3 will only work as a transport medium for requested packets. After that, the request will be extracted from the cloud and serve to the user through layer 3 and layer 2. As layer 1 prioritizes UDP type packets, it will store that the requested packets in the server which will be resulted in faster response for the next request.

2. Case II: The Requested packet is TCP Type

On the other hand, if the packet is TCP type then the request will send to Layer 2 and layer 2 will check whether it is UDP type or not. As it is not UDP type, the request will forward to layer 3. Then Layer 3 will check if it is TCP type or not and as the requested packet type is TCP, it will search for the related packet in that layer. Then it will take necessary actions and respond back to the end device or layer 1 through layer 2. Here, layer 2 will only work as a transport medium for requested packets. If the requested packet is not found, it will be forwarded to layer 4 which is the main cloud. After that, the request will be extracted from the cloud and serve to the user through layer 3 and layer 2. As layer 2 prioritizes TCP type packets, it will store that the requested packets in the server which will be resulted in faster response for the next request.

Figure 3.3 explains how a sent request from Layer 1 is handled at the different layers. When layer 1 sends request TCP type packets, layer 2 will not search for the packet and send it to layer 3 directly. This cut off the search time in layer 2. The packet will be searched in layer 3 and if the packet is not found then the request will be sent to the cloud for processing. After that, the response will be sent to layer 1 via layer 3 and layer 2. In the meantime, the packet will be saved in layer 3 which will result in faster response for the next request. Thus, for the next user of that packet will get the response in a faster way.

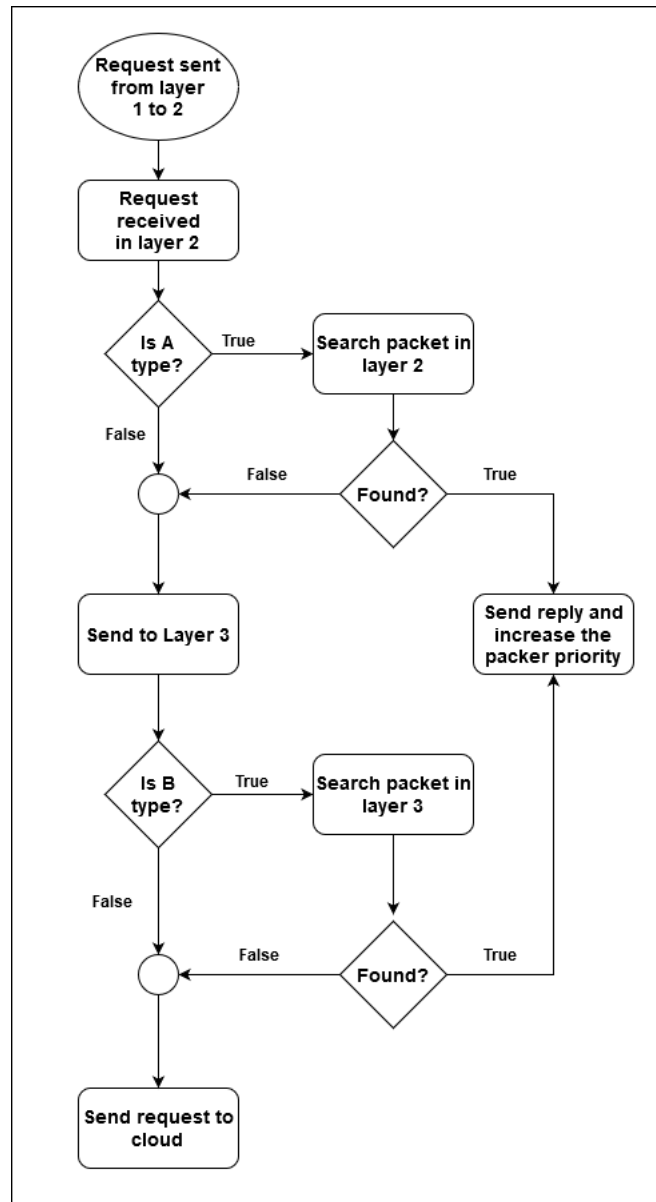


Figure 3.3: Flow chart of Processing Sent Request from End Device

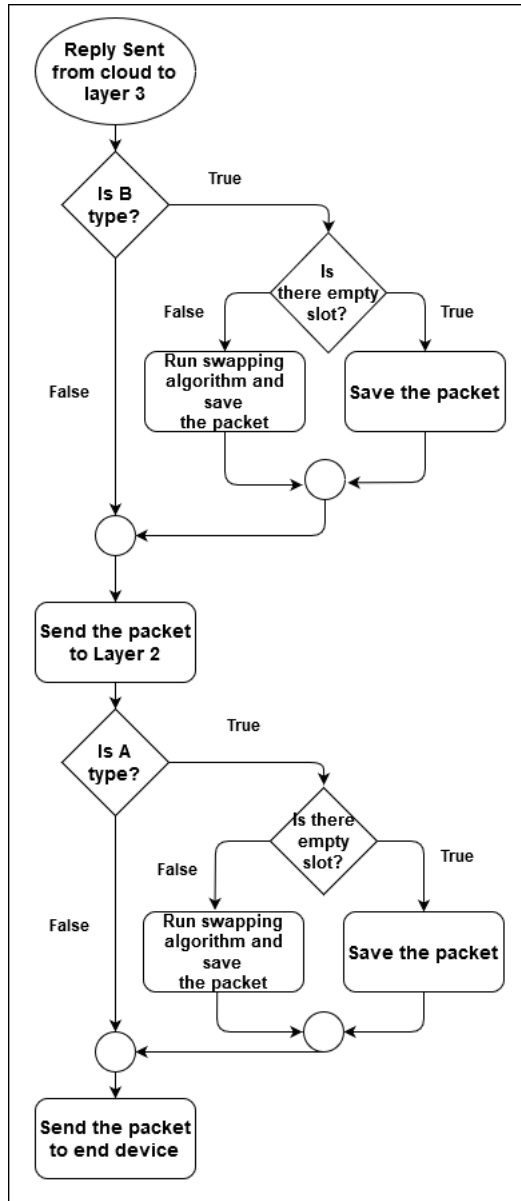


Figure 3.4: Flow Chart of Reply Sent from Cloud

Figure 3.4 explains the flow chart of handling reply sent from the cloud. When any packet will be stored in any layer, it will search for available space in that specific layer. If there is no space for the packet, it will run Least Time between Access algorithm and replace the packet which has been used least recently. As a result, the proposed infrastructure will ensure optimal use of the FOG server's storage. To implement Least Time between Access, every packet will be given a flag which will be increased automatically over time. The maximum value of flag will be stored so that swapping can be done efficiently. To find out the maximum valued flag, selection sorting mechanism is preferable. Figure 4 explains how reply is sent and handled from Layer 4.

In the worst case scenario, any request for UDP type packet will be searched in layer 2. The request will not be found in the layer and so the packet must be extracted from the cloud. Here one advantage is that the packet will not search in layer 3 which will reduce the time. On the other hand, if the request is a TCP type packet, the packet will not search in layer 2 and go to layer 3. If the packet is not found in layer 3, it will extract the packet from and save it to layer 3. After that, the next request will be processed quickly and which will ensure less latency and more throughput.

3.2 Caching Mechanism:

In the proposed infrastructure and mechanism, the UDP type packet will have a faster response than the TCP type packet. The mechanism of packet classification has done such a way so that to implement a secure system, packets that need faster processing falls under the category of UDP type and packet which need comparatively less fast processing, falls under the category of TCP type. In order to improve latency caching has many benefits [29]. Some benefits are:

- Predictable performance.
- Reduce the load on the back end.
- Increase read throughput.

In order to populate caching there are three main issues. These are:

- Populating cache.
- Keeping the cache and remote system sync.
- Managing cache size.

To populate cache, we are using the Lazy Population technique. This technique checks whether the data is in cache or not. If the data is not in the cache then it extracts data from the cloud. The benefits of the lazy population are given below [29] [30]:

1. The system only cached the requested data.
2. No upfront cache delay in this system.
3. The system does not face any fatal error if any node failure occurs.

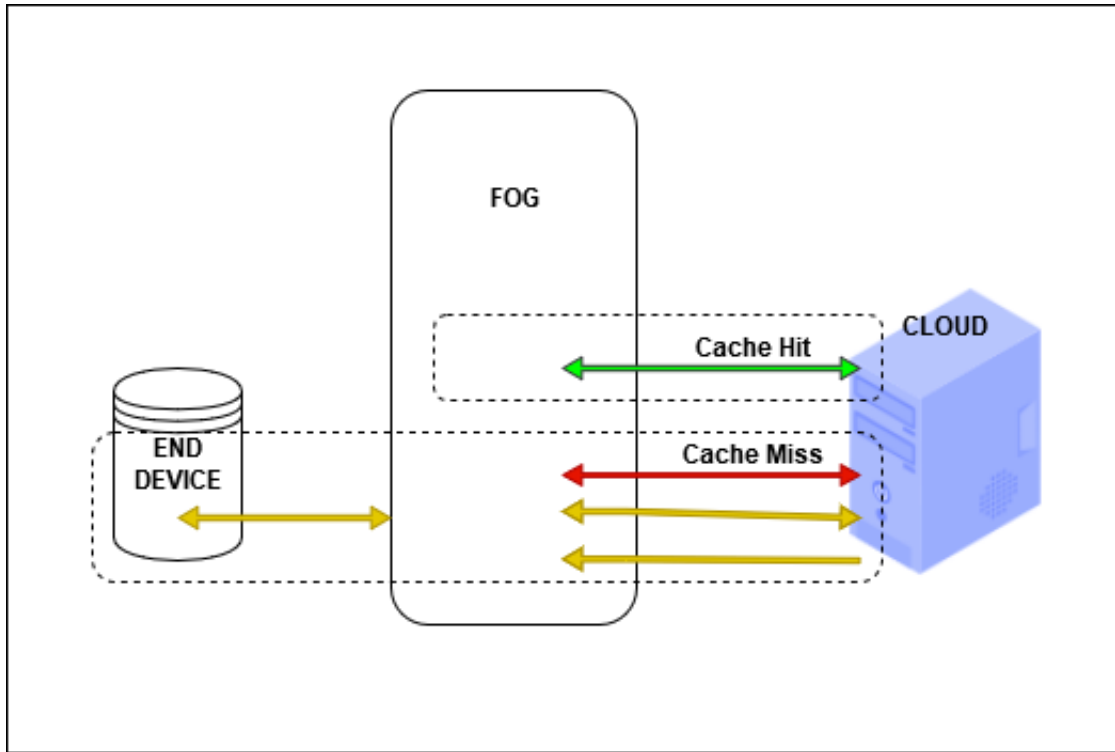


Figure 3.5: Lazy Population of Caching Mechanism

The following Figure 3.5 explains the cache population of the lazy population caching method.

For the second issue, which means keeping the cache and remote system sync, our proposed model follows the Active Expiry method. It means the remote system will update the cache directly if any change occurs in the cloud.

The advantages of Active Expiry Given below [29]:

- It ensures the fast change in caching.
- Unnecessary expiration of unchanged data does not occur in the cached data.

For managing cache size we have adapted the Least Time between Access method. It means when a cached data is accessed it counts both time and access time of that cached data. Then it takes the average time in between access time to calculate the average time for access. Whenever the data is accessed counter of that data is increased. During swapping any stored value, the layer first swap with the least accessed data by checking the counter. If there are multiple data with the same counter value, then it swaps with the highest average value of time accessed data. The following figure shows the mechanism of calculation of hit count and average time of access for any cached data stored in Layer 2 and Layer 3.

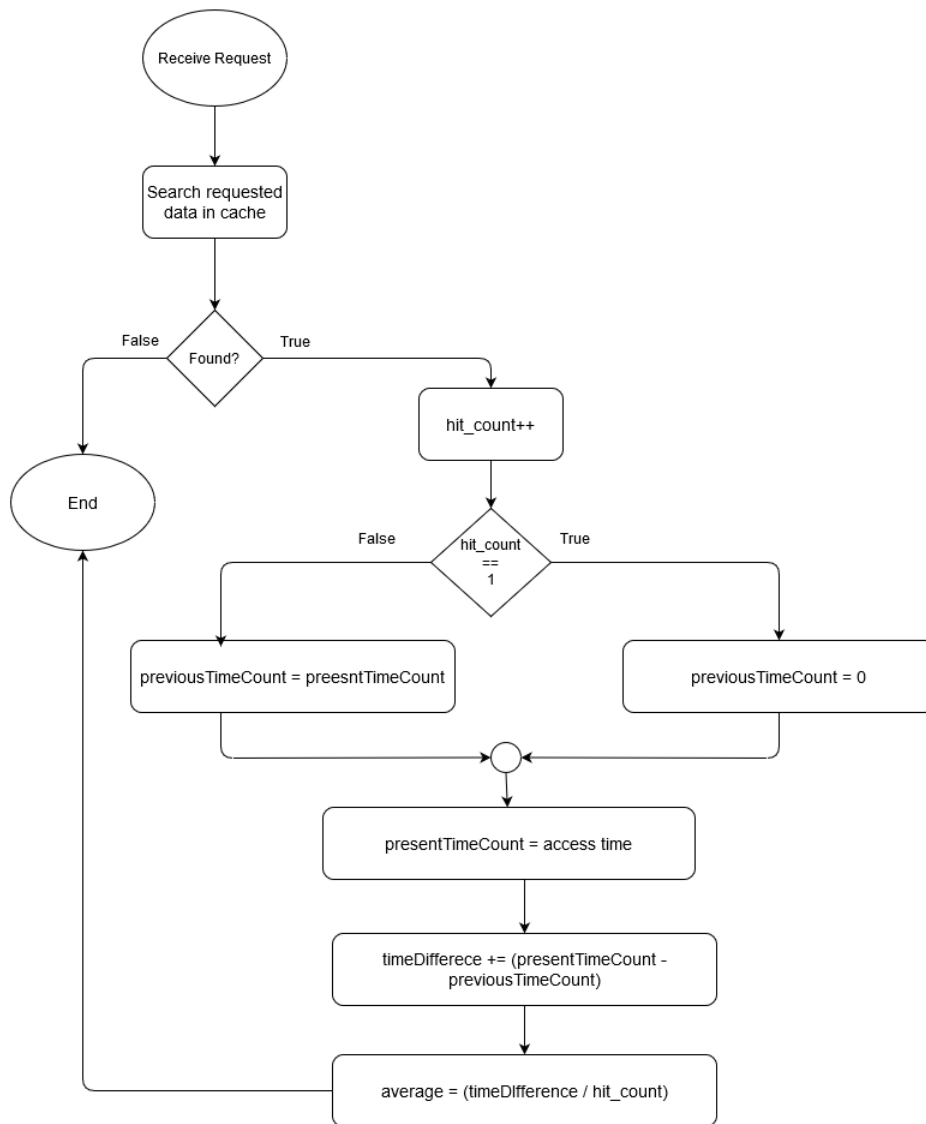


Figure 3.6: Flowchart of Caching Mechanism

Figure 3.6 explains how the caching mechanism calculates hit count and average access time. Initially, the hit count is set at 0. When hit count is 1 it means the data has been requested for the first time and so the average time between access calculation starts after this process. Each time the data is requested hit count increases and corresponded average time is calculated. The reason for calculating hit count and average access time to ensure effective swap of data when the cache is full. The storing technique along with swapping in the cache is given below:

Algorithm 1 Algorithm for swapping new data with stored data

```
begin
  if any slot is available then
    | Save the data;
  else
    Assume first data in minimum hit count;
    for every Data in cache do
      | if hit count < minimum hit count then
        | Update minimum hit count;
      end
    end
    Count the minimum hit count
    if Minimum hit count > 1 then
      | Find the maximum average time access among them;
      | Swap data with them;
    end
    else
      | Swap data with minimum hit count;
    end
  end
end
```

This algorithm explains how the FOG layer swaps and stores data. Here, if any slot is available, then the FOG layer will save the data without any checking. Otherwise, to swap data, it will first check the minimum hit count of any data. If there is a multiple numbers of minimum hit count occurs, then it will check the average access time and swap with the maximum average access time data among the minimum hit count. The reason is that if there is already a single minimum hit count data, then the system will not search for minimum average hit count and it will lessen the time complexity of the swapping mechanism.

3.3 Experimental setup

3.3.1 General setup:








The proposed model consists of four layers that include an end device layer, a lower middleware layer, an upper middleware layer and a cloud layer as Layer 1, Layer 2, Layer 3 and Layer 4 respectively. We called each resource group as a layer hence we created four individual resource groups for four layers. A resource group includes resources that we typically manage as a group. We can deploy resources to resource groups based on what makes the most sense for our organization. When creating a resource group, we are going to provide a location for that resource group. This is because of the resource group stores metadata about the resources that are hosted in it. When we are specifying a location for a resource group we are just specifying where the metadata for the resource is stored. In each resource group, we have deployed a virtual network, a network security group, and a virtual ma-

chine. The Azure Virtual Networks allow many types of Azure resources including Azure Virtual Machines to securely communicate with one another. It allows us to communicate with the Internet and communicate on prime networks. The address range for any subnet that is defined within a virtual network must fall within the address space of the virtual network itself so in this case, we have assigned a different range of IP address to each resource group. Besides, a network security group contains security rules that are used to allow or deny inbound network traffic to or outbound network traffic from many different types of Azure resources. This helps us to manage our virtual machines once it is deployed to a network security group. The Virtual Machine is the common resource that people deploy in Azure to simulate or run a machine in a remote location. All the end devices will be at Layer 1 which is called the end devices layer.

Resource groups	
All subscriptions	
Cloud	Central US
LowerMiddleWare	Central US
EndDevices	Central US
upperMiddleWare	Central US

Figure 3.7: Four Layers combined as resource group

Thus, in Layer 1, each end device consists of a virtual machine, a virtual network, a network security group and storage. In a virtual network, we have assigned an IP-address range of 172.17.0.0/16 and a subnet of 172.17.0.0/24 so that the VM can use this to connect with the server. We have located the resource group and as well each resource in the central US. In a virtual machine, we have associated the subnet created with the virtual network, also we have selected a platform of windows 10 pro version with 4GB of RAM and a default HDD storage provided with the Virtual Machine. In addition to that we have deployed a network security group to the end device layer, this resource group is very important because it provides security to the layer. It also allows inbound and outbound traffic for this layer. In the network security group, we can assign in which port do we want to send the data or to receive data thus we have used it in Layer 2 and Layer 3 which will be discussed in detail in the Layer 2 and Layer 3 description.

Resources		
EndDevices		
Refresh		
 EndDevicesVM	Virtual machine	Central US
 EndDevicesNSG	Network security group	Central US
 EndDevicesVM-nsg	Network security group	Central US
 enddevicesdiag586	Storage account	Central US
 EndDevicesVM_OsDisk_1_fb70836...	Disk	Central US
 enddevicesvm814	Network interface	Central US
 EndDevicesVM-ip	Public IP address	Central US

[See more...](#)

Figure 3.8: Resources of End Devices

Similarly, in the lower middleware, we have also deployed resources such as virtual network, a virtual machine and a network security group. We have assigned an IP address of 172.18.0.0/16 and a subnet of 172.18.0.0/24. All the resources are also located in the central US region just like the end devices layer. It ensures that the Layer 2 is closer to Layer 1 which can also be written as we have placed the lower middleware closer to the end device. Then we deployed a virtual machine assigning it with the subnet and a platform of windows 10 pro version with 4GB RAM and a default HDD storage. In this layer, the network security group is very important because it gives security and we have given a protocol in the particular port. In this case, we have fixed the Layer 2 port with UDP type packets. Therefore, the layer can accept only UDP type packets request and replies the packet by taking necessary actions and if any other packet type is requested then Layer 2 will fetch the data from the Layer 3 and then give it to the end device. Thus, we have fixed the Layer 2 with the UDP type protocol with the help of a network security group.








Resources		
LowerMiddleWare		
		Refresh
 ImwVM	Virtual machine	Central US
 ImwVM-nsg	Network security group	Central US
 ImwNSG	Network security group	Central US
 Imwvm617	Network interface	Central US
 ImwVM_OsDisk_1_0771d6ba51694...	Disk	Central US
 ImwVM-ip	Public IP address	Central US
 ImwVN	Virtual network	Central US
		See more...

Figure 3.9: Resources of Lower middleware

Apart from that, in upper middleware, we have deployed resources including virtual network, a virtual machine, and a network security group. We have given an IP address of 172.19.0.0/16 and a subnet of 172.19.0.0/24. Similar to Layer 1 and Layer 2 we have deployed all the resources in the central US region. Similarly, it also denotes the Layer 2 of the proposed FOG model is placed nearer to the end devices to make sure to minimize the latency. In this VM, we have given a windows platform of version 10 with 4GB RAM and with a default HDD storage. Since this layer only accepts TCP packet requests thus with the help of the network security group, we assigned the port which only accepts TCP type packets.








Resources			
upperMiddleWare			
			Refresh
	lwmVM	Virtual machine	Central US
	umwNSG	Network security group	Central US
	lwmVM-nsg	Network security group	Central US
	ipconfig1	Public IP address	Central US
	lwmvm863	Network interface	Central US
	lwmVM_OsDisk_1_a3a13f8c04a54...	Disk	Central US
	lwmVM-ip	Public IP address	Central US
			See more...

Figure 3.10: Resources of Upper middleware

Finally, we have deployed the same resources as we used in other layers. This layer-4 is considered as a cloud and we have assigned the port so that it can accept all types of packet requests. The cloud is placed to the Japan west considering a long distance from the end device. Generally, clouds are not situated closer to end devices as a single cloud gives services to various regions.








Resources			
Cloud			
			Refresh
	CloudVM	Virtual machine	Japan West
	CloudNSG	Network security group	Japan West
	cloudvm388	Network interface	Japan West
	CloudVM_OsDisk_1_276c115681e...	Disk	Japan West
	CloudVM-ip	Public IP address	Japan West
	CloudVM-nsg	Network security group	Japan West
	CloudVN	Virtual network	Japan West
			See more...

Figure 3.11: Resources of Cloud

3.3.2 Layer Configuration:

To make communication between virtual machines we have used java socket programming. Socket programming is a process of communication and transformation of information by connecting network nodes. This connection between the nodes occurs when one node starts listening to the other node. This connection can be within the same network or different network.

Different Java runtime environment is used to build up that communication process of java socket programming.

There are two types of java socket programming and they are:

- 1. Connection-oriented socket programming:**

The most important part of the connection-oriented socket programming is the Socket class and ServerSocket class. The Socket class denotes the socket that helps both server and client to start the communication and ServerSocket class helps the server to get a port and then it starts listening for the clients to connect. To establish a TCP type connection between the networking nodes, connection-oriented socket programming is used.

- 2. Connection-less socket programming:**

On the other hand, the prime component of connection-less socket programming is DatagramSocket and DatagramPacket. DatagramSocket class is responsible for sending and receiving data that is individually addressed and it also denotes the data it sends does not have any order to follow. Likewise, the DatagramPacket class denotes that the data can be routed from different devices. Besides, it indicates that the information it sends or receives might lose. To establish a UDP type connection between the networking nodes, connection-less socket programming is used.

Therefore, it is clear now that one of the network nodes acts as a server and another one acts as a client. To start the connection between the server and the client, the client must have the proper information about the server. That information includes:

- 1. Server's IP Address**
- 2. Server's port number**

Here, two-way client and server communication has been built. In this model, the client sends a request to the server. Then the server starts reading the request and responds immediately by sending the requested file. In this request sending and receiving process, Socket and ServerSocket classes are responsible. Here the Socket class makes the communication between the client and the server. In addition, this class helps to read and write requests or send and receive data. On the server-side, the ServerSocket class takes the action. The ServerSocket class has an accept() method that blocks the console as far as the client establishes the connection. At the server-side, an instance of Socket is returned when a client successfully makes a connection.

3.3.3 Client-Side Programming:

At first, a client application needs to be created by forging an instance of the Socket class. As a parameter, we passed the server's IP address and the allocated port number of the server which is 12345.

In order to initiate a client request, we have followed the below-mentioned steps:

1. **Establish a Connection**
2. **Communication**
3. **Closing the connection**

3.3.4 Server-Side Programming:

On the other hand, a server application needs to be created by forging an instance of the ServerSocket class. To make the connection, we have fixed the port to 12345. The accept() method blocks the console as far as the client establishes the connection. An instance of Socket is returned when a client successfully makes a connection on that fixed port number.

To initiate a server response, we have followed the below-mentioned steps:

1. **Communication**
2. **Send the output through the socket**
3. **Close the Connection**

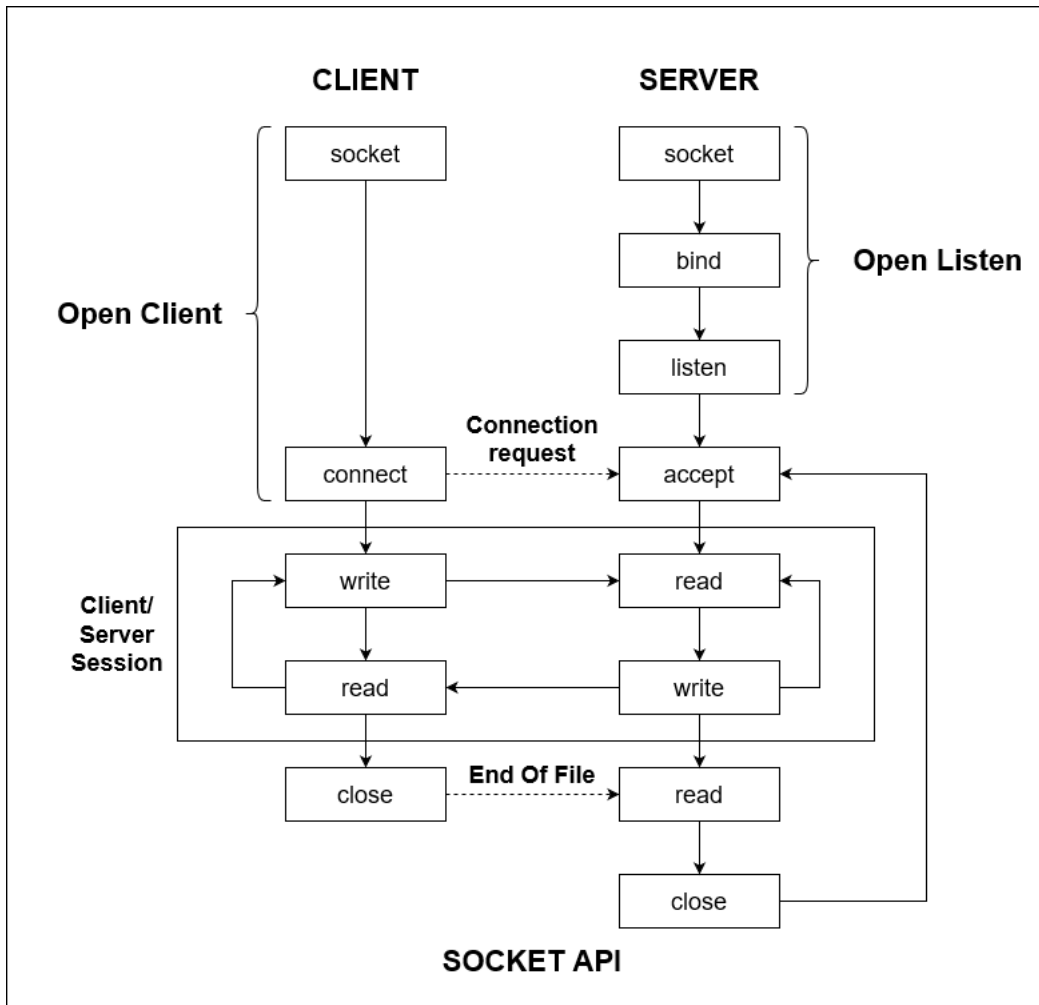


Figure 3.12: Working principle of socket API

In order to do the socket programming, the IP address and the port has to be fixed otherwise the connection cannot be established. For that reason, we have done port forwarding /port fixing first. In Microsoft Azure, we can do it using the network security group. In the network security group, we have used static IP for the VMs and also, we have fixed the inbound port for the VMs.

3.3.5 Layer 1 configuration:

In the network security group of layer 1, we have used static IP address which is 13.67.140.6 for the VM named End Devices and we have fixed the inbound port to 12345 for the VM.

Algorithm 2 Layer 1(UDP)

input : DATATORECEIVE the data name that will be requested

output: Receives response of that requested data

begin

for *each request* **do**

 Create a socket connection with Layer 2 on a fixed port for UDP type packets;

 Sends output to the socket;

 DATATORECEIVE stores requested name by input stream of bytes;

 Make a blank file named after the DATATORECEIVE;

 Request the file to the layer;

 Receives the file as a byte array;

for *each Byte Reads > - 1* **do**

 Reads the byte array;

end

 Write the bytes into the created file named after DATATORECEIVE;

 Print the received data size along with data name;

end

 Close connection;

end

Algorithm 3 Layer 1(TCP type)

input : DATATORECEIVE the data name that will be requested

output: Receives response of that requested data

```
begin
  for each request do
    Create a socket connection with Layer 2 on a fixed port for TCP type packets;
    Sends output to the socket;
    DATATORECEIVE stores requested name by input stream of bytes;
    Make a blank file named after the DATATORECEIVE;
    Request the file to the layer;
    Receives the file as a byte array;
    for each Byte Reads > - 1 do
      Reads the byte array;
    end
    Write the byes into the created file named after DATATORECEIVE;
    Print the received data size along with data name;
  end
  Close connection;
end
```

3.3.6 Layer 2 configuration:

Similarly, in the network security group of layer 2, we have used static IP address which is 40.113.228.189 for the VM named lmwVM and we have fixed the inbound port to 12345 for the VM. Microsoft Azure has a built-in packet identification method in the network security group that means we do not need to do packet classification manually for layer 2. We have used that built-in procedure for our setup to accept UDP type packets.

Algorithm 4 Algorithm for Layer 2(Lower Middleware)

input : DATATORECEIVE the data name that will be send in response

output: Sends response of that requested data

```
begin
  Create object of socket server using fixed port;
  while true do
    Accepting connectivity of socket server;
    DATATSEND stores the requested data name from the client;
    Call fileSend method and send DATATSEND as parameter;
    Close connection;
  end
  @Method
  Void fileSend takes socket and File name as parameter;
  begin
    Create file object;
    if file exists then
      Convert it to byte array;
      Send the byte array;
      Print the file send confirmation message;
    end
    else
      Print the requested is not found and request has been forwarded to
      upper layer;

      Call FileClient methodand sends DATATSEND as parameter;
    end
  end
  @Method
  Void FileClient receives file name as parameter;
  begin
    Saves file name to FILETORECEIVED;
    Create a socket connection with Layer 3;
    Sends output to the socket;
    Request the file to the layer 3;
    if file type is UDP then
      Receives the file as a byte array;
      for each Byte Reads > -1 do
        Reads the byte array;
      end
      Write the byes into the created file named after FILETORECEIVED;
      Print the received data size along with data name;
    end
  end
end
```

3.3.7 Layer 3 configuration:

To configure the layer 3, we have used 52.165.155.84 static IP address in the network security group. Also, we have fixed the port to 12345 for the VM named lwmVM. Microsoft Azure has a built-in packet identification method in the network security group that means we do not need to add packet classification manually for layer 3. We have used that built-in procedure for our setup to accept TCP type packets.

Algorithm 5 Algorithm for Layer 3(Upper Middleware)

input : DATATOSEND the data name that will be send in response

output: Sends response of that requested data

```
begin
  Create object of a socket server using fixed port with the requested client;
  while true do
    Accepting connectivity of socket server;
    DATATOSEND stores the requested data name from the client;
    Call fileSend method and send DATATOSEND as parameter;
    Close connection;
  end
  @Method
  Void fileSend takes socket and File name as parameter;
  begin
    Create file object;
    if file exists then
      Convert it to byte array;
      Send the byte array;
      Print the file send confirmation message;
    end
    else
      Print the requested is not found and request has been forwarded to
      upper layer;

      Call FileClient methodand sends DATATOSEND as parameter;
    end
  end
  @Method
  Void FileClient receives file name as parameter;
  begin
    Saves file name to FILETORECEIVED;
    Create a socket connection with Layer 4;
    Sends output to the socket;
    Request the file to the layer 4;
    if file type is UDP then
      Receives the file as a byte array;
      for each Byte Reads > -1 do
        Reads the byte array;
      end
      Write the byes into the created file named after FILETORECEIVED;
      Print the received data size along with data name;
    end
    else
      Forward the file to layer 2;
    end
  end
end
```

end

3.3.8 Layer 4 configuration:

Similarly, we have configured the layer by giving 40.74.118.158 static IP address in the network security group of the VM named lwmVM. Also, we have fixed the port to 12345 for the VM. Microsoft Azure has a built-in packet identification method in the network security group that means we do not need to add packet classification manually for the layer 4. We have used that built-in procedure for our setup to accept all types of packets.

Algorithm 6 Algorithm for Layer 4(Cloud)

input : DATATOSEND the data name that will be send in response

output: Sends response of that requested data

begin

 Create object of a socket server using fixed port;

while true do

 Accepting connectivity of socket server;

 DATATOSEND stores the requested data name from the client;

 Call fileSend method and send DATATOSEND as parameter;

 Close connection;

end

 @Method

 Void fileSend takes socket and File name as parameter;

begin

 Create file object;

if file exists then

 Convert it to byte array;

 Send the byte array;

 Print the file send confirmation message;

end

else

 Print the requested is not found;

end

end

end

Chapter 4

Result and Analysis

4.1 Result

4.1.1 UDP Type Data Transfer:

As mentioned earlier, FOG communication model does not have any standard model yet, we have compared our result with traditional cloud computing using Microsoft Azure. The setup of the experimental environment has mentioned earlier. We have experimented with our model at first UDP type request from Layer 1. The Inbound Graph of Virtual Machine that has been used as End Device is given below.

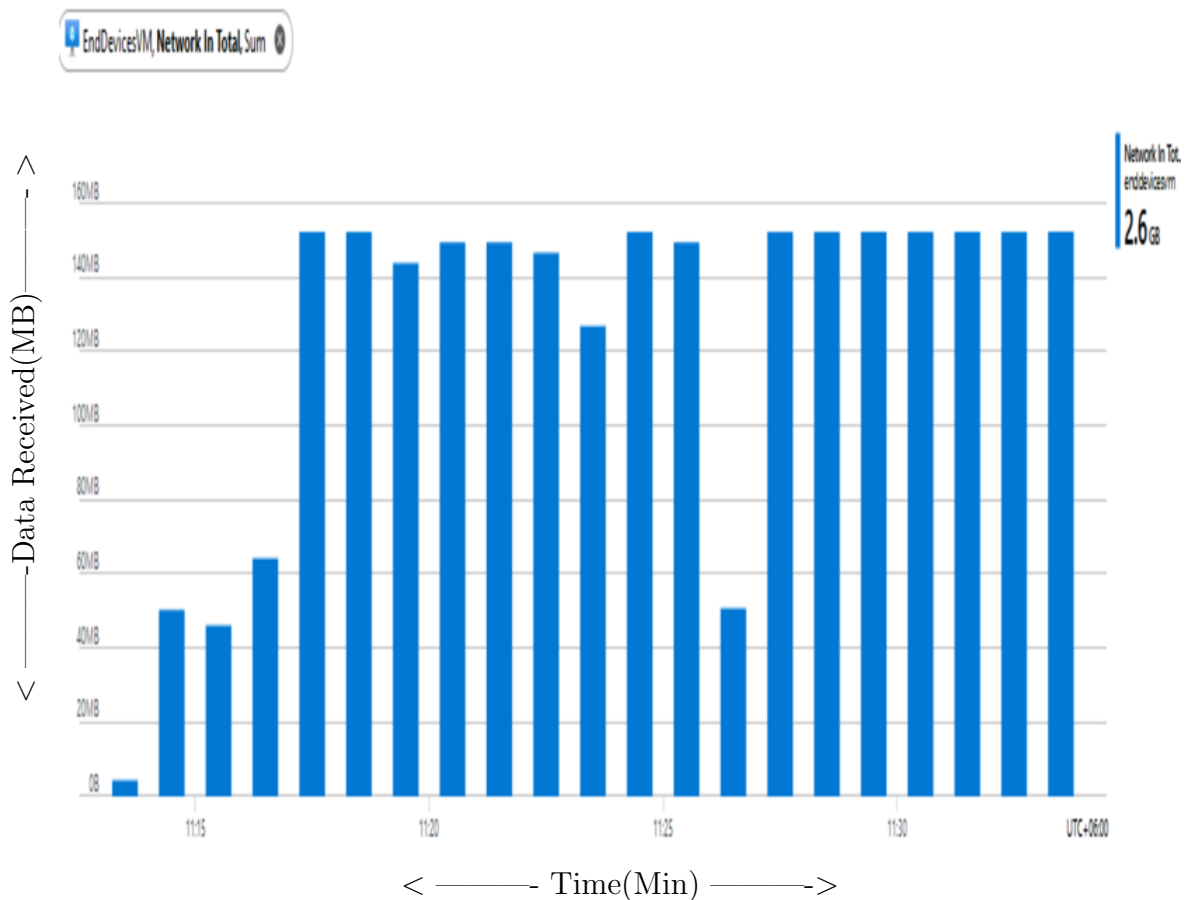


Figure 4.1: Inbound Data graph for UDP type request (End Device)

In the graph of Figure 4.1, X-axis shows the time and Y-axis shows the incoming data per minute. Here we can see that initially, the inbound data rate was very low. The incoming data was less than 5 MB in the first minute. For the next 3 minutes, the inbound data was around 40 MB to 60 MB. After that the incoming data speed becomes very fast as we can see that it was most of the time more than 140 MB per minute. However, we can see that between 11.25 and 11.30 there was a drop in incoming data speed. After that, the rate is more than 140 MB per minute for the rest of the time. Here initially the speed was less because of caching the data from cloud to Layer 2. As we are using the lazy population for caching technique, initially there was no cached data in Layer 2. As Layer 2 started receiving requests, it searched the data and did not find so the request was sent to the cloud via Layer 3. After caching the data for first 3 minutes, all the requests from Layer 1 was handled in Layer 2. As a result, the inbound data rate for End Device increased very high. However, after 11.25 the caching capacity was full so Layer 2 needed to run a caching algorithm to swap and store new data. For this reason, the speed decreases around 50 MB per minute. After the caching, the speed increases again to around 150 MB per minute. Here, 2.6 GB data is transferred in around 20 minutes.

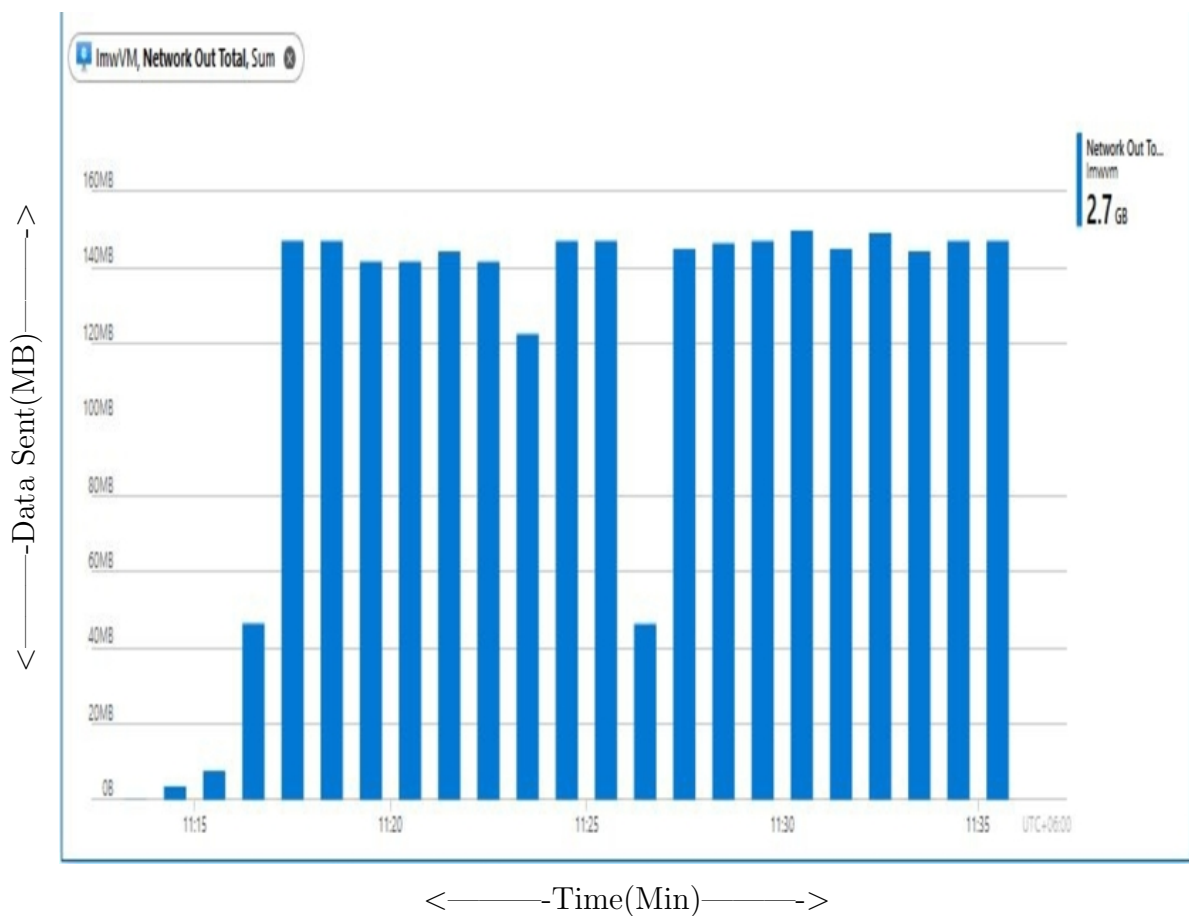


Figure 4.2: Outbound Data Graph of Layer 2 server

The graph of Figure 4.2 X-axis shows the time and Y-axis show the data sending per minute. Here we can see the corresponded UDP server's outbound graph for the incoming request from Layer 1. In the graph, initially, data-sending rate was very low. In the beginning, it was less than 10 MB per minute. Then it increases to more than 40 MB in the next minute. Then the speed becomes more than 140 MB per minute. Between 11.25 to 11.30, there is a drop in speed and then the speed rises again. Here the reason in initially Layer 2 was caching data from the cloud server. As the caching process was the lazy population, initially the outbound data rate was slow due to generate cache. Once the cache is generated, the speed increases very high and continues to send data at high speed. However, between 11.25 to 11.30 the caching space was full and so that to swap new data with stored data, the caching technique took some time which results in the decrease of data transfer. Here from both graphs, we can see that the server sent 2.7 GB data around 20 minutes where 2.6 GB data was sent to End Device and the rest of the data was sent for requesting the missing data from cloud via layer 3. For UDP type data transfer the data speed was up to the mark.

4.1.2 TCP Type Data Transfer:

For TCP type data transfer, Layer 3 or upper middleware of FOG has been used as a cache server. Data request from Layer 1 has been passed to Layer 3 via Layer 2. If the data is not present in Layer 3, then Layer 3 extracts the data from the cloud and sends the reply to Layer 1 via Layer 2. We have sent TCP type request from END Device to extract data and the resulted graph is given below:

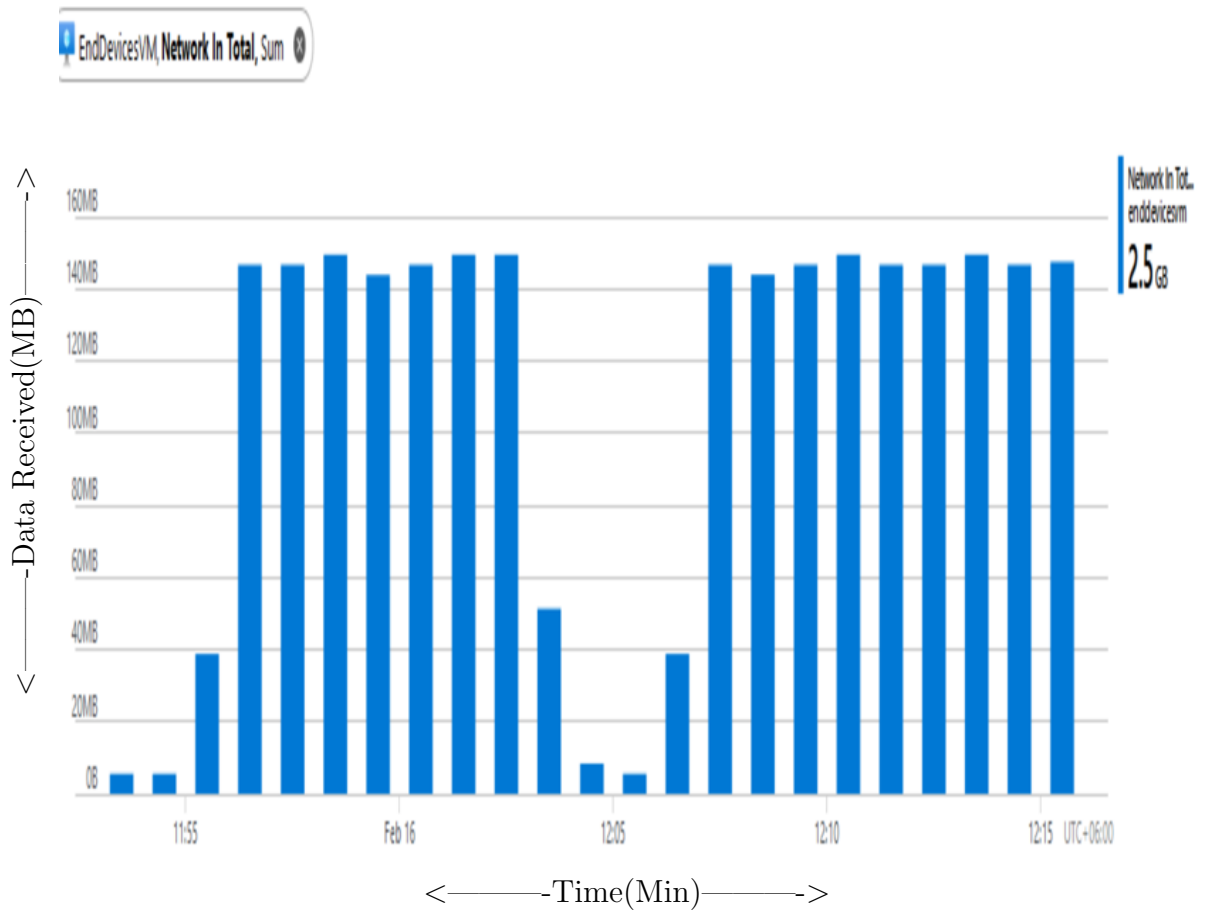


Figure 4.3: Inbound Data graph for TCP type Request (End Device)

Figure 4.3 shows the graph where X-axis represents the time and Y-axis represents the incoming data per minute. Here, incoming data initially was very slow which was less than 10 MB and then around 40 MB. After that, the data-incoming rate increases and becomes more than 140 MB per minute. Between 12:03 to 12:07, there is a drop in the rate, which is very around 40 MB per minute to 10 MB per minute. After that, the rate increases again to more than 140 MB per minute. Here the reason that, initially the rate was very low due to not having the data in Layer 3. As our proposed model has used the lazy population for caching mechanism, it caches the data initially. After data caching, the inbound rate has increased a lot. As a result, more than 140 MB of data per minute was received. Even though the data transfer rate falls drastically between 12:03 to 12:07, it begins to rise more than 140 MB per minute again. The reason for the fall is that in that time cached capacity was full and so to extract new data from the cloud and store it using the caching mechanism, the inbound data transfer rate decreased. End Device of Layer 1 received 2.5 GB of data around 25 minutes.

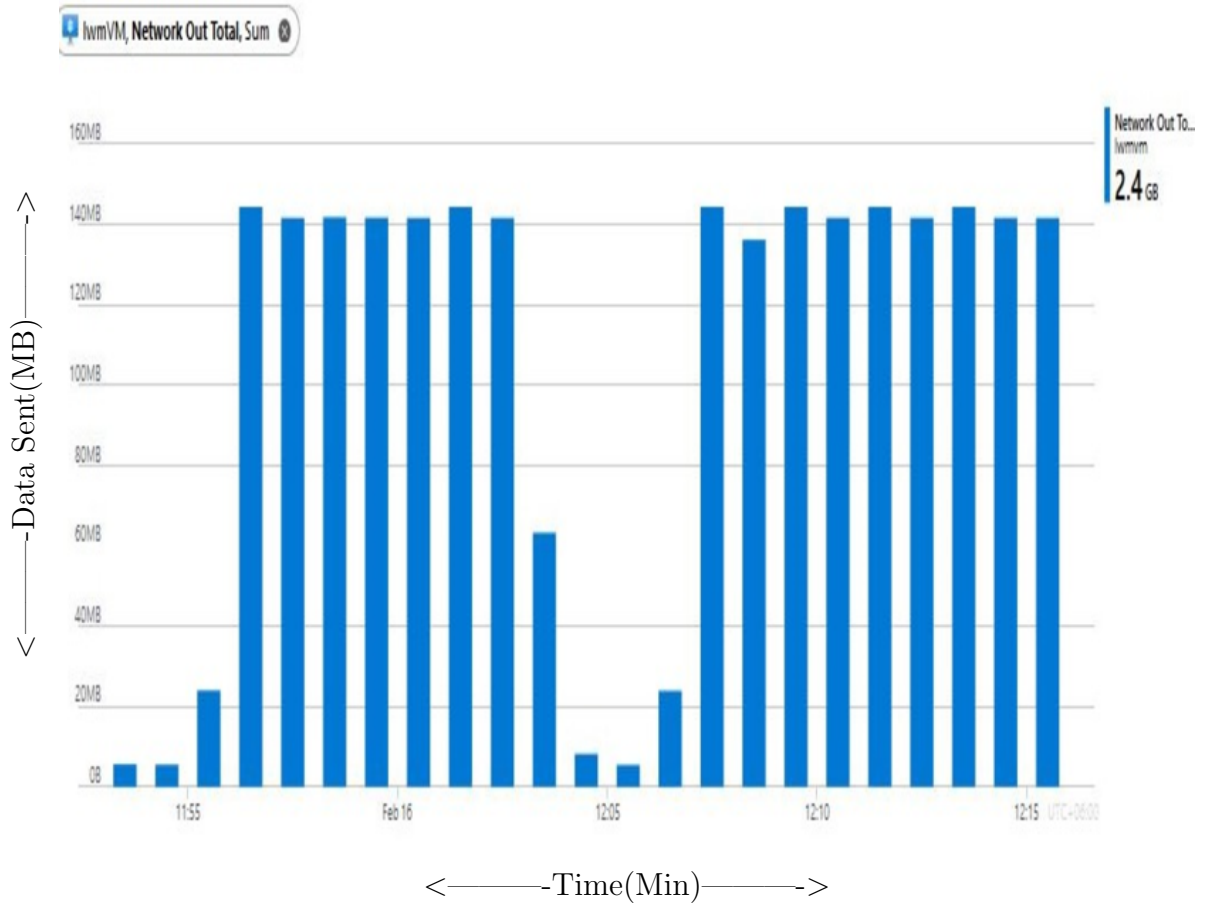


Figure 4.4: Outbound Data Graph of Layer 3 server

In Figure 4.4, X-axis represents the time and Y-axis represents the data sent over time. This graph is the corresponded inbound graph of the Layer 1 End Device. As similar to the inbound graph of the End Device, initially data-sending rate was very low which increased and maintained more than 140 MB per minute. The reason is that initially, the requested type data was not present in the Layer 3 server. As a result, it requested data from the cloud. While replying to the requested data the server cached the data and next time it replied from its layer. Therefore, the speed increased very high which is more than 140 MB per minute. This rate is kept for a time being until the capacity of the cached server is filled. After the cached server is being full, it ran the caching mechanism for swapping old data and restoring new data. As a result, the outbound data rate decreased. Data sending rate increased again after caching new data and the rate stays around 140 MB per minute. The server in Layer 3 sent 2.4 GB data around 25 minutes.

4.1.3 Cloud Server Data Transfer:

To compare our model with the traditional model we simulated the traditional cloud systems using our Virtual Machines. There we sent data requests directly to the cloud and received a reply to End Device. Middle wares of FOG were absent in that scenario. The inbound graph of End Device is given below:

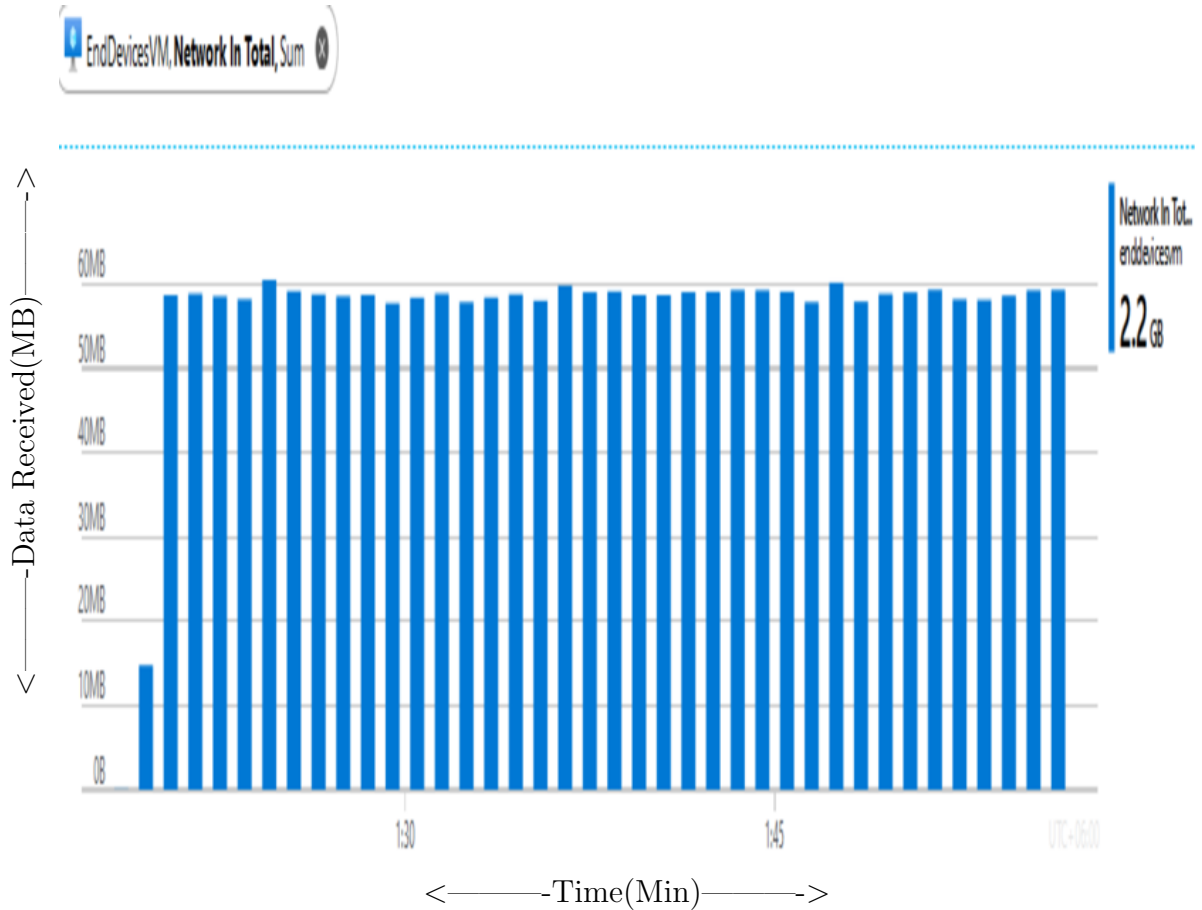


Figure 4.5: Inbound graph of End Device for the traditional cloud model

In the graph shown in Figure 4.5, X-axis represents time and Y-axis represents the received data per minute. As we can see, the data transfer rate at first minute was around 15 MB as it established connection. After that, the data transfer rate becomes 60 MB per minute. The rate stays constant as there is no need for caching and all the data is present in the server. On the other hand, the rate is low comparatively with FOG model, as data need to extract from the distant cloud server. The End Device had received 2.2 GB data for around 30 minutes.

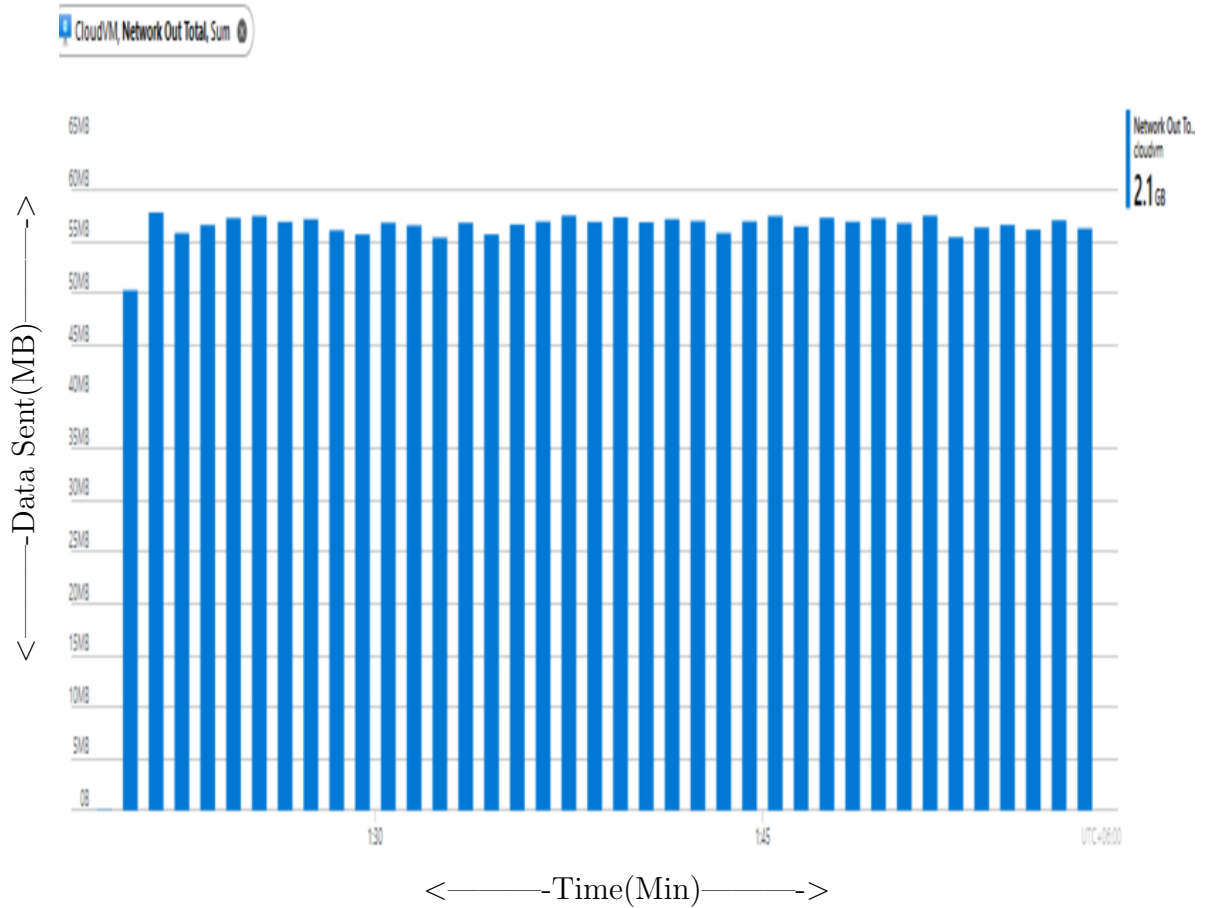


Figure 4.6: Outbound Data graph for traditional cloud server

In the graph shown in Figure 4.6, X-axis represents time and Y-axis represents sending data per minute. As we have found a constant graph for the End Device, the outbound graph is also constant in Cloud server. There is no need for caching the data so the server immediately sent data to the end device without using FOG middlewares. The rate is also similar to the End Device's inbound graph, which is close to 60 MB per minute. The Cloud server sent 2.1 GB data around 30 minutes.

4.2 Analysis:

To analysis our proposed model with the traditional cloud. We have created a table showing the inbound data for TCP and UDP type requests for our model and traditional cloud model. The table shows inbound data of every half minutes in megabytes.

Table 4.1: Inbound Data of End Device in Different Simulation)

SL	Time (Min)	Data1(End Device Cloud Inbound)(MB)	Data2(End Device TCP Inbound)(MB)	Data3(End Device UDP Inbound)(MB)
1	0.5	4.77	3	1.99
2	1	10	2.67	1
3	1.5	31.3	3	23
4	2	27	2.67	27.18
5	2.5	28	20.88	20.08
6	3	30.42	18	26
7	3.5	27	73.34	30
8	4	31.27	73	34.14
9	4.5	29	73	75
10	5	29.09	73.34	77.17
11	5.5	32	74.5	74
12	6	28.24	74	78.17
13	6.5	29	71	70
14	7	29.58	71.86	72.98
15	7.5	29.38	73.64	72.75
16	8	29	73	76
17	8.5	30.27	74.5	72
18	9	28	75	76.75
19	9.5	29.35	74.5	71.68
20	10	29	75	75
21	10.5	30	25.42	62.94
22	11	27.85	26	64
23	11.5	28	4.49	78
24	12	30.18	4	74.32
25	12.5	29	2.67	72.5
26	13	29.43	3	76
27	13.5	27	20	23.79
28	14	30.84	19	27
29	14.5	30	73.68	74.32
30	15	28.2	73	78
31	15.5	29.39	71.94	75.32
32	16	29	72	77
33	16.5	30	73.7	77
34	17	28	73	75.33
35	17.5	29.09	74.51	75.33

SL	Time (Min)	Data1(End Device Cloud Inbound)(MB)	Data2(End Device TCP Inbound)(MB)	Data3(End Device UDP Inbound)(MB)
36	18	29	75	77
37	18.5	29.51	73.68	78
38	19	29	73	74.35
39	19.5	30.56	73.68	75.32
40	20	28	73	77
41	20.5	29.34	74.52	74.41
42	21	29	75	78
43	21.5	30	73.7	72.5
44	22	28.33	73	76

In the table, we have taken data from 0 to 22 minutes for each simulation. The time interval of each data is 30 seconds and so we have 44 data. Now if we plot all the data in the graph, we get the following graph.

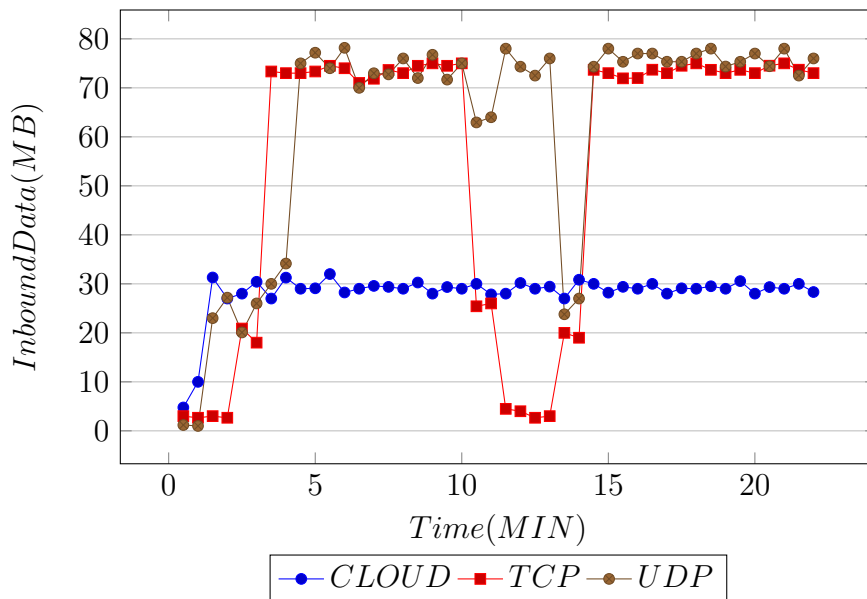


Figure 4.7: Inbound Data Comparison Graph of End Device for Traditional Cloud vs Proposed Model

In Figure 4.7, we can see that initially for both TCP and UDP model the rate is less than the traditional cloud model. After 3 minutes, the graph for both the proposed model's TCP and UDP model has increased very high compared to the traditional cloud model. The reason behind this is that due to our caching mechanism, all the frequent requests have been handled very quickly. Therefore, from End Device's, latency for any request improved in a great margin. Although there is a drop off rate in the middle part, it was occurred due to the swapping of data after being the storage capacity full. After retrieving new data from the cloud and extracting it, the data rate again becomes very high. It shows the improved latency of the proposed

layered architecture FOG model.

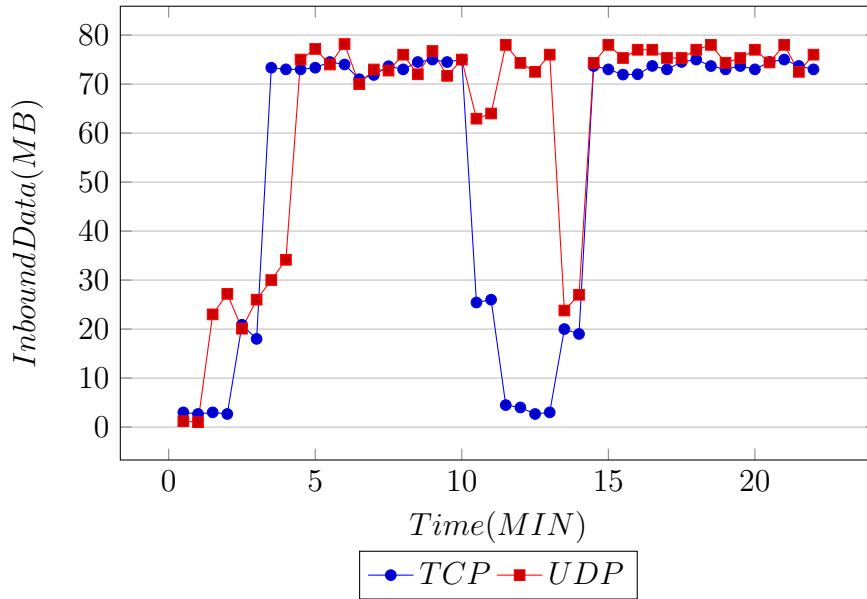


Figure 4.8: Inbound Comparison Graph for TCP vs UDP Request of Proposed Model

Figure 4.8 represents the comparison of inbound data for TCP and UDP type requests for the proposed FOG layered architecture. Here we can see that UDP type data transfer rate increase faster than TCP type data transfer rate. The reason behind this is Layer 2 handles UDP type data while Layer 3 handles TCP type data. As a result, UDP type data transferred faster than TCP type data. If we look between 2.5 to 3.5 minutes of the X-axis, we would see that the TCP type data transfer rate is faster than the UDP type. This occurs because TCP type data is stored the closer to cloud and UDP type data is stored closer to END Device. Besides, if we analyze the missing part of both graphs, we will see that the UDP type request requires less recovery time than the TCP type requests. The reason behind this is that UDP type packet transfer does not require any three-way handshaking while TCP type requests require three-way handshake. As a consequence, when any miss occurs and the layer needs to run the caching algorithm for swapping and storing values, Layer 2 requires less recovery time than Layer 3. Although, when the caching is done and the connection is established, we can see that for both TCP and UDP type frequent requests, the inbound data rate is almost the same which ensures less latency. Therefore, for frequent requests of IoT devices, the proposed FOG model can improve the latency by decreasing the response time.

The whole simulation was done by Microsoft Azure which is a cloud service provided by Microsoft. Microsoft provides dedicated public IP addresses for each Virtual Machine and a very fast Internet connection. For this reason, the incoming and outgoing data from End Devices and different layer's server is very high in this simulated scenario.

Chapter 5

Conclusions and Future Work

5.1 Conclusions:

Communication has been a very important part of human civilization for ages. Communication was mainly between people in the past but nowadays technology allows us to communicate with machines, devices, and gadgets. These days, technologists included advanced sensors with insights into the fundamental gadgets to make them communicate without human input. Therefore, to meet the demand of these gadgets, a huge amount of data needs to be processed within the shortest possible time to ensure faster communication. Cloud computing does the job of processing that huge amount of data but cannot handle it efficiently. Thus, FOG computing comes along with cloud computing to extend the traditional cloud computing process and make the communication process actual faster but again it does not have any standard structure.

The proposed model is designed to develop the communication process of IoT devices based on the FOG communication model to ensure effective communication between IoT sensors and the cloud. The rate of increasing IoT devices in the present world is very high for which only the traditional cloud-based solution approach will not be efficient in the coming future. For this reason, the layered architecture of the FOG communication model is introduced. The main focus of this model is to decrease latency and increase the throughput of the IoT devices using layered FOG model. For ensuring the objectives, the idea of packet classification has been used to identify the packets with proper priority so that packet requests can be replied by maintaining that. According to the priority, the close layer of end devices deals with UDP type packets as UDP type packets require a faster response and on the other hand, upper middleware of FOG deals with TCP type packets. Configuring layers of the FOG model based on packet classification will ensure minimum computation time and minimum latency along with improved throughput. In conclusion, the layered FOG model with packet classification will improve the total data communication for IoT devices.

5.2 Future Work

In the middlewares, there is scope for introducing advance probability theorem to switch between caching mechanism in order to ensure better and optimal performance. In order to do this, the Poisson distribution of the probability theorem is suitable for this. Poisson distribution is a special case of binomial distribution where the number of trials n is very large and probability p is very small. Here the request is considered as the number of trials and miss cases are considered as probability p . Initially, the probability of miss will be high but as time passes, the probability will decrease. Using Poisson distribution there can be set a threshold to switch from different caching techniques about managing cache size such as least access, LRU, time expiry, etc. In addition, in the middleware layers, there can be introduced machine learning and neural network techniques for ensuring data security.

FOG is the future of cloud-based IoT communication. There is a huge opportunity to contribute to this sector as it is still in an early stage of research. As there are middlewares of FOG, there is huge scope for different types of improvements in order to overcome the challenges of the FOG communication model.

Bibliography

- [1] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, “Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing”, in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, IEEE, 2014, pp. 325–329.
- [2] A. Banafa, *Iot: A fog cloud computing model*, 2015.
- [3] D. KG, M. V. Ganesh, *et al.*, “Fog computing: Issues, challenges and future directions.”, *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 7, no. 6, 2017.
- [4] A. Chakrabarty, T. A. Heya, M. A. Hossain, S. E. Arefin, and K. D. D. Joy, “A novel extended-cloud based approach for internet of things”, in *Cloud Computing for Optimization: Foundations, Applications, and Challenges*, Springer, 2018, pp. 303–331.
- [5] F. Uribe, “The classification of internet of things (iot) devices based on their impact on living things”, *Available at SSRN 3350094*, 2018.
- [6] R. Deng, R. Lu, C. Lai, and T. H. Luan, “Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing”, in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 3909–3914.
- [7] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang, “A hierarchical distributed fog computing architecture for big data analysis in smart cities”, in *Proceedings of the ASE BigData & SocialInformatics 2015*, 2015, pp. 1–6.
- [8] K. A. N. Reader, *An era of iot- m2m communication protocols*, 2018. [Online]. Available: <https://medium.com/predict/an-era-of-iot-m2m-communication-protocols-a03c396397a1>.
- [9] T. A. Heya, S. E. Arefin, M. A. Hossain, and A. Chakrabarty, “Comparative analysis of computation models for iot: Distributed fog vs. conventional cloud”, in *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)*, IEEE, 2017, pp. 760–765.
- [10] T. H. Ashrafi, M. A. Hossain, S. E. Arefin, K. D. Das, and A. Chakrabarty, “Iot infrastructure: Fog computing surpasses cloud computing”, in *Intelligent Communication and Computational Technologies*, Springer, 2018, pp. 43–55.
- [11] T. H. Ashrafi, M. A. Hossain, S. E. Arefin, K. D. J. Das, and A. Chakrabarty, “Service based fog computing model for iot”, in *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, IEEE, 2017, pp. 163–172.

- [12] L. Liu, D. Qi, N. Zhou, and Y. Wu, “A task scheduling algorithm based on classification mining in fog computing environment”, *Wireless Communications and Mobile Computing*, vol. 2018, 2018.
- [13] H. T. T. Binh, T. T. Anh, D. B. Son, P. A. Duc, and B. M. Nguyen, “An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment”, in *Proceedings of the Ninth International Symposium on Information and Communication Technology*, 2018, pp. 397–404.
- [14] D. Rahbari and M. Nickray, “Low-latency and energy-efficient scheduling in fog-based iot applications”, *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 27, no. 2, pp. 1406–1427, 2019.
- [15] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, “Mobility-aware application scheduling in fog computing”, *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.
- [16] J. Oueis, E. C. Strinati, and S. Barbarossa, “The fog balancing: Load distribution for small cell cloud computing”, in *2015 IEEE 81st vehicular technology conference (VTC spring)*, IEEE, 2015, pp. 1–6.
- [17] L. Yin, J. Luo, and H. Luo, “Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing”, *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4712–4721, 2018.
- [18] X.-Q. Pham and E.-N. Huh, “Towards task scheduling in a cloud-fog computing system”, in *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, IEEE, 2016, pp. 1–4.
- [19] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, “A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration”, *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.
- [20] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, “Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system”, *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, 2016.
- [21] D. Hoang and T. D. Dang, “Fbrc: Optimization of task scheduling in fog-based region and cloud”, in *2017 IEEE Trustcom/BigDataSE/ICSS*, IEEE, 2017, pp. 1109–1114.
- [22] T. Choudhari, M. Moh, and T.-S. Moh, “Prioritized task scheduling in fog computing”, in *Proceedings of the ACMSE 2018 conference*, 2018, pp. 1–8.
- [23] W.-C. Yeh, C.-M. Lai, and K.-C. Tseng, “Fog computing task scheduling optimization based on multi-objective simplified swarm optimization”, in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1411, 2019, p. 012007.
- [24] M. Rouse, *What is fog computing? - definition from whatis.com*, 2019. [Online]. Available: <https://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging?fbclid=IwAR2xRS4o4uuV-CmKDRcutPSbPOmRfddHq-AExu86-FRiR2ZNrq9pzKrkmVo>.
- [25] M. R. Anawar, S. Wang, M. Azam Zia, A. K. Jadoon, U. Akram, and S. Raza, “Fog computing: An overview of big iot data analytics”, *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

- [26] A. B. ROSS and A. D. D. E. C. INTELIGENTES, “Campus curitiba-centro curso de engenharia de computacao”,
- [27] P. Mishra, S. Kumar, and S. Mishra, “Significance of fog computing in iot”, in *Proceedings of 3rd International Conference on Internet of Things and Connected Technologies (ICIOTCT)*, 2018, pp. 26–27.
- [28] M. Dixit, B. Barbadekar, and A. B. Barbadekar, “Packet classification algorithms”, in *2009 IEEE International Symposium on Industrial Electronics*, IEEE, 2009, pp. 1407–1412.
- [29] M. ŠŤAVA, “Robustní a škálovatelný datový sklad pro systém perun”, PhD thesis, Masarykova univerzita, Fakulta informatiky, 2015.
- [30] *Caching strategies*. [Online]. Available: <https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/Strategies.html>.