Contents lists available at http://qu.edu.iq

Al-Qadisiyah Journal for Engineering Sciences

Journal homepage: https://qjes.qu.edu.iq

An improved throttling algorithm for fog computing networks with an additional management layer

Omar Anwer Nafea* and Turkan Ahmed Khaleel

Department of Computer Engineering, College of Engineering, Mosul University, Mosul, Iraq Iraq

ARTICLEINFO

Article history:
Received 00 December 0000
Received in revised form 00 January 0000
Accepted 00 February 0000

Keywords
Fog Computing
Load balancing
OMNIT++
Response time
Throttled algorithm

ABSTRACT

An emerging networking technique called fog computing extends cloud computing capabilities to the edge network's borders. It is employed to get around the limitations of cloud computing, like latency and bandwidth problems. Fog computing is suitable for IoT systems and applications that require real-time processing, reliable network access, low latency, and strong security. In this work, the objective is to design and implement a fog computing environment to simulate the behavior of a multi-user healthcare application, which represents the monitoring of elderly care homes in Mosul city. Several algorithms were employed to examine the effects of load balancing inside fog computing networks. These algorithms are Random, Round-Robin, and the modified Throttled algorithm, which is modified by adding an extra management layer to be more suitable for fog computing networks. The response time results obtained from implementing this modified method were superior to those of the random algorithm and closely resembled the response time results of the round-robin algorithm. In case QoS1 with 25 clients, the result was (0.246037794) second without the load balancing algorithm, (0.124323358) second in the Random algorithm, (0.115641477) second in the Round-Robin algorithm, and (0.114981575) second for the modified throttled algorithm. thus, making it applicable for fog computing networks and cloud computing networks.

1. Introduction

For the majority of people, "fogging" or "fog computing" is a relatively new idea that Cisco introduced in 2014. A relationship exists between fog and cloud computing; just as fog is typically found in areas closer to the ground than clouds, so too is this the case in technology. It is possible to bring cloud capabilities down to the ground level using fog computing because it is closer to end users [1]. The term "fog computing" refers to a distributed computing paradigm that places computation and data storage closer to end users and devices. The concept of cloud computing is extended to the edge of the network by fog computing. Processing and analyzing data closer to where it is created and eliminating the need for data to be transported to centralized data centers are two key tenets of fog computing, which promises to address the issues of latency, bandwidth, security, and privacy that emerge with traditional cloud computing. In fog computing, several distributed, decentralized, and heterogonous devices are placed closer to end devices, sensors, and actuators at the edge of the network. These edge devices are interconnected and communicate through the fog layer, which is responsible for providing services such as processing, data caching, and analysis. Computing in the fog provides real-time and context-aware decision-making and effective use of network

* *Corresponding author.*
E-mail address: omar.21enp5@student.uomosul.edu.iq (Omar Anwer Nafea)

https://doi.org/10.30772/qjes.2024.146104.1089

Nomenclature:

FogNetSim	Fog Network Simulator	*Msg*	Message
Ini	initialization file	*MTH*	Modified Throttled Algorithm
IOT	Internet of Thing	*N*	Number of Client
LB	Load Balance	*NML*	Network Management Layer
LC	Least Connection	*PPP*	Point to Point
LPCF	Least Processing Cost First	*QoS*	Quality of Service
MIPS	Million Instruction Per sconed	*ToS*	Type of Service
MQTT	Message Queuing Telemetry Transport	*WRR*	Weighted Round Robin

resources. The Internet of Things (IoT) is greatly aided by fog computing, which offers high-quality services with fast response times and a scalable, flexible platform for managing the enormous volumes of data produced by IoT devices. It is a crucial technology for edge computing as well, which processes data locally on edge devices like robots, drones, and smartphones [2][3]. The fog computing architecture is a hierarchy of computing resources arranged into layers, each performing specific functions. This fog computing architecture is designed to solve the limitations of cloud computing, like latency, bandwidth and security issue. By bringing computing resources closer to the edge of network, fog computing can improve the performance, reliability, and security of edge devices and applications and enable new use cases in areas such as smart cities, healthcare, and industrial IoT. Figure 1 shows the fog computing architecture, which is usually composed of three layers: the Cloud Computing layer at the top, the Fog Computing layer in the center, and the End Device layer at the bottom [4][5] [6].

Figure 1. Fog Computing Architecture [6].

Cloud layer: This layer provides additional resources and services, such as data storage, compute power and advanced analytics. It can be used to store data that is not immediately needed or perform complex analytics requiring large amounts of computing power [7].

Fog layer: The layer situated just above the layer intended for end devices is known as the Fog Computing layer. Any device that is capable of storing, processing, and connecting to a network is referred to as a Fog Computing device which includes the Fog nodes that are located closer to the network's edge than cloud data centers and are responsible for providing services such as data analytics, storage, and communication. These Fog nodes can be deployed in various locations, including on-premises, in public spaces, or in vehicles, providing a range of services to end-users in real-time [8].

End Devices, or Data producer layer: This layer is the closest to the end devices, such as sensors, actuators, and mobile devices. It consists of edge devices that collect and transmit data to the fog layer [7].

An extension of cloud computing's services is offered to the Internet of Things (IoT) layer by the fog computing layer. An overview of the most significant of these services will be given in this section, as they are classified into three main categories: compute, storage, and network services [9]

- Storage: The sensor has the capacity to produce large volumes of data. Given the rate at which data is generated with increasing of using IoT technology, the storage capacity of IoT gadgets at the IoT layer is often insufficient to accommodate all the data. Therefore, not all data needs to be moved to the cloud immediately, especially if some of it is redundant or unnecessary. It is advised to perform filtering in such circumstances or store the data in the fog computing layer [10].
- Computing: the limited computing power of IoT-layer devices has led directly to the development of distant processing methods. Enhancing system requirements, ensuring energy efficiency, enabling local processing, and achieving a faster response time are the motivation behind fog layer treatment. As a result, processing tasks can be moved from the cloud computing to the fog computing layer [11].
- Communication: wireless nodes play a crucial role in facilitating communication inside the Internet of Things (IoT), due to the limited resources in the IoT layer, the wireless protocols are specifically designed to operate with minimal energy use, limited bandwidth transmission, and extended coverage [12].

The paper's contribution is adding a network management layer, which will help researchers in the future by adding several other parameters and injecting their own load balancing algorithms.

2. Review of the Literature

This section entails a comprehensive examination of the most recent and advanced studies and related works in the field. This review is structured into two sections. The first section focuses on simulators and frameworks for fog computing, fog computing's applications in healthcare is covered in the second section, which also looks at load balancing techniques in this field.

2.1. Fog Computing Simulator and Frameworks

Nowadays, fog computing is a modern environment where many fog computing technologies are used to support and construct applications in a wide range of fields, including healthcare, smart grids, smart homes, smart buildings, intelligent transportation systems, etc. [13]. These applications and technologies frequently profoundly effect on people's lives. Still,

further consideration is needed because not all of these applications have quite reached a satisfactory degree of maturity. Therefore, the use of simulators was mandated. There are various fog computing environment simulation tools with varying characteristics and modes of operation. We will examine a few of these simulators in this paragraph:

An Edge-Fog cloud simulator was developed by (N. Mohan, et al., 2016). It is built of two layers: the edge device layer and the fog device layer. The simulator is implemented in Python. The Least Processing Cost First (LPCF) method, which assigns jobs to potential nodes, was then developed and incorporated by the authors. Reducing computing time and network expenses is the aim of the assignment [14].

(A. Brogi, et al., 2017) develop a fog computing paradigm comprising qualified deployments, fog infrastructures, IoT applications, and QoS profiles. FogTorch is a simulator tool written in Java that lets developers adjust the fog infrastructure. Developers can specify QoS settings pertaining to latency and bandwidth and inform the simulator of the requirements of their applications. There is no fee model offered by FogTorch [15]. (H. Gupta, et al., 2017) suggested a simulator called iFogSim built on JAVA and is used to model fog networks in IoT environments and assess the effects of resource management strategies on latency, network congestion, energy consumption, and cost, the authors present two case studies that represent an IoT environment and compare different approaches to managing resources. In addition, the simulation toolkit's scalability in terms of RAM usage and running time is validated in a variety of settings. However, iFogSim is not perfect; in particular, it puts too much emphasis on resource management while neglecting other critical areas of fog computing such as infrastructure and mobility [15].

(T. Qayyum, et al., 2018) proposed the FogNetSim++ simulation framework, that gives users a wide range options for setting up a Fog network simulation. It is built on OMNeT++ [17], which is an open-source component-based C++ simulation library and framework that is frequently used in academic settings. Mobility models, handover models, energy models, and Fog node scheduling algorithms can all be executed via FogNetSim++ [18]. The authors (Lera, et al., 2019) presented YAFS, a Python-based simulator designed for fog computing, enabling the simulation of cloud and fog environments. The YAFS architecture enables the incorporation of structural measures into dynamic and customizable strategies, such as workload location, application module placement, service scheduling, and path routing. This is achieved by utilizing complex network theory to model the relationships between applications, infrastructure elements, and network connections. Following that, the authors performed a comparative investigation to investigate the convergence and efficiency of results between the highly acknowledged simulator, iFogSim, and the YAFS simulator [19].

(A. W. Malik, et al., 2021) proposed xFogSim, a framework for an enhanced fog simulator (FogNetSim++) constructed with OMNET++. In order to balance cost, availability, and performance within the fog federation, xFogSim provides multi-objective optimization assistance for latency-sensitive fog layer applications. In order to meet service and energy needs under high load, nearby fog sites can lend resources thanks to location-aware distributed broker node management. The outcomes demonstrate how adaptable, scalable, lightweight, and capable the framework is by handling a large number of user requests through the fog federation's dynamic resource provisioning [20]. Table 1, represents a summary of the most important simulators and frameworks mentioned in the above-related work.

2.2. Fog Computing Application

Global improvements in healthcare, public health, medical research, and technology, as well as increased awareness of personal hygiene, the environment, and nutrition, are all responsible for the notable rise in life expectancy that has been seen in recent decades. Due to rising life expectancies and the high costs of providing healthcare and other well-being services for the elderly, there is an increasing number of older people, which is dangerous for the socioeconomic systems of many nations (S. Majumder, et al.,2017) [21]. Monitoring the health and well-being of the elderly in care homes (smart homes) can be achieved at a low cost through continuous remote surveillance. Smart homes are equipped with environmental and wearable medical sensors, advanced communication technologies, and actuators. This integrated system leverages cutting-edge technology, including powerful processors and wireless communication platforms, to provide healthcare, safety, and well-being services directly to residents. Operating on the principles of the Internet of Things, the smart home connects all sensors and devices within the residence, enabling remote monitoring of occupants' health, environmental conditions, and overall safety and security [21].

In 2019 by (O. Debauche, et al.), A novel architecture for patient and geriatric monitoring based on fog IoT was introduced by Olivier Debauchee et al. The significant growth in the senior population and their desire to live independently, while having age-related medical conditions, calls for the creation of new technologies to guarantee the best possible standard of living for this group. Preventive medical surveillance may also be advantageous for another group of patients, those with life-threatening conditions. By using physiological and environmental signals, they demonstrated a cloud-based health monitoring system for the fog IoT that can provide contextual data for activities of daily living. With the help of this device, healthcare professionals may monitor the health and behavioral changes of elderly or alone patients. Additionally, this technology allows for the tracking of patients' rehabilitation and recovery processes [22]. In 2022 (P. Singh et al.), the authors offer a thorough analysis of several job-scheduling techniques used in fog computing. It examines and contrasts several task-scheduling techniques created for a fog-computing environment to highlight their benefits and drawbacks. Finally, it offers potential study options for other scientists working in the fog-computing environment [23]. Beraldi et al. (2020) provide an extensive compilation of potential research endeavors in the field of fog computing. These studies encompass a range of areas, including heterogeneity, security, diversity, energy consumption, response time, execution time, and load balancing. The authors point out that many researchers in the field have largely ignored these aspects. Hence, it is possible to enhance the effectiveness of scheduling algorithms in the fog-computing setting by integrating different techniques and tang into account crucial performance aspects. The study's findings indicate that basing scheduling decisions on state information that arrives even slightly after the service time greatly reduces the effectiveness of load balancing. An investigation is being conducted on a threshold probe-based technique with little fanfare to address this impact. This technique is more desirable than the current alternative, especially in a geographically accurate situation characterized by a greater degree of unpredictability in the incoming load [24].

According to (V. Kashyap, et al., 2022), numerous algorithms have been proposed that utilize LB to address the issue of unreasonable data in network congestion. Response time, execution time, security, latency, and bandwidth are criteria on which writers have focused in LB. The authors

state that more attention needs to be paid to a few parameters in the field of fog computing. Security is the most crucial factor because when a server is under heavy demand, the processor's limited capacity makes it difficult for it to respond appropriately, which might result in system failure and the loss of user databases [25]. In 2022 by (N. R. O. Al-Rubaie et, al.) introduces an Internet of Things (IoT)-based fog computing paradigm that combines IoT and fog computing (FC). The quality checks were performed using the FogNetSim++ add-on and OMNeT++. According to the study, the scenario with FC is especially useful since it controls data exchange rates, delay time, and channel availability through the use of data exchange rates. Furthermore, learning automata are utilized to incorporate packets from similar directions into the base fog node manager of the network. According to the study, the proposed FC scenario is particularly useful as it applies learning automata to add packets originating from similar directions to the primary fog node manager of the network. It also uses data exchange rates to control channel allocation, delay time, and throughput [26].

suggests using multiple queues. A weighted round-robin algorithm is used to schedule these queues, allocating jobs to them based on the assigned value of the ToS (Type of Service) bits, which are located in the IP header's second byte. The researchers have arrived at the following four conclusions: Primarily, fog computing frameworks facilitate the technological work process and enable developers to experiment with their ideas prior to their implementation in real-world scenarios. Additionally, the use of multiple queues significantly lowers the latency that fog nodes and users experience, which in turn reduces error rates and packet drop rates. Furthermore, as compared to the first-come, first-served scheduling strategy, the weighted round robin yielded greater results than regular round robin. Ultimately, accounting for every aspect will result in a framework that produces results that are as genuine or nearly real as those found in the real world [27].

Table 1. Different Fog Simulation and Framework Comparison

Simulator	Year	Build on	Network Configuration	GUI	License	Infrastructure	Work on
Edge-Fog cloud	2016	Python	×	×	GNU	Edge, fog layer	Establishes a resource network, assigns tasks, and sets configuration parameters.
Fog Torch	2017	Java	×	×	MIT	Cloud, fog nodes, and things	QoS-aware deployment of IoT applications
iFogSim	2017	Java	×	×	N/A	Fog devices, actuators, sensors, and data centers	Comparison of import topologies, resource management strategies, and cost
FogNetSim++	2018	C++, based on OMNeT++	√	√	GNU	Base stations, sensors, fog nodes, broker nodes, and mobile devices	Enables the use of communication protocols and applications and includes pre-installed modules like sensors, mobile devices, fog nodes, and brokers.
YAFS	2019	Python		√	MIT	Cloud, fog, sensors, and actuators	Network design, billing management, and resource allocation analysis
xFogSim	2021	C++, based on OMNeT++	√	√	GNU	Base stations, sensors, fog nodes, broker nodes, and mobile devices	Enables the use of communication protocols and applications and includes pre-installed modules like sensors, mobile devices, fog nodes, and brokers.

(D. B. Abdullah et, al., 2022) employed a fog simulation framework with a smart agent layer established between the end-user device and a fog layer. Rather than using a single queue at the Ethernet layer, the framework

(A.S. Kadhim et, al., 2022), the authors suggested an IoT-based fog-to-server architecture that uses distributed environments and hybrid load balancing to address the issue of packet loss in fog and servers. The

suggested approach uses a combination of weighted round-robin and hybrid load balancing with the least connection in fog nodes to distribute requests to the active servers based on load and time. The first case study does not include a load balancer; the second case study uses the least connection (LC) algorithm as the load balancer; the third case study uses weighted Round Robin (WRR) as the load balancing algorithm; and the fourth case study uses a hybrid approach that combines LC and WRR implemented in each fog node. These case studies serve as the foundation for the proposed system. The load balancing mechanism in the proposed multilayer architecture is effective and allows access control to be adjusted. The findings demonstrate that the suggested solution enhanced network performance by guaranteeing the effective processing of IoT requests originating from the IoT layer by utilizing distributed fog computing services in conjunction with a hybrid load-balancing technique [28].

3. Scheduling and Execution of Tasks

The processing of client-generated tasks starts by determining the amount of the load to be processed and querying the load balancing function to compute an optimal processing location (fog node or broker). so, that the task is scheduled to the designated broker using a new publish message that informs the designated broker about the new task, various data is kept at the scheduling broker to be able later to combine with the data returning after task execution and generate the proper publish message that inform subscribers about the updated topic. This work implemented a functional mechanism that allows for easy selection of one of multiple algorithms to be used via configuration file (".ini") to compute a suitable fog node or broker using one of the following algorithms random, round robin, and throttled.

3.1. Random Algorithm

This is a static algorithm relying on a list of available fog brokers and generating a number between 0 and (list size -1) and using that number as an index to select the suitable fog broker for task execution.

3.2. Round Robin Algorithm

Round robin is also a static algorithm that relies on a list of available fog brokers, it uses a counter to determine the suitable fog broker for task execution incrementing the counter afterward, if the counter value exceeds the list size the counter is reset.

3.3. Throttled Algorithm

The throttled algorithm distinguishes itself from the above-mentioned algorithms by being a dynamic algorithm that uses runtime-obtained data to decide on the best suitable fog broker for task execution. it does that by requesting the status of all known fog brokers generating a list of returned statuses and using the first available fog broker for task execution. The scanning mechanism provided with the traditional implementation of the throttled algorithm does not suit the distributed nature of the fog computing environment, for this reason, an alternative mechanism for collecting the status of broker nodes on the network is used. The alternative mechanism depends on the broker nodes publishing their status periodically via a dedicated message over the network while listening for status messages incoming from the remote broker node. This alternative mechanism will be referred to within the scope of this work as the Network Management Layer.

4. Network Management Layer

The traditional implementation of the throttled algorithm divides the executing devices into two groups, that is, available or busy devices, respectively, where the available segment contains nodes that can be used to execute an incoming task while the busy segment contains nodes that are excluded from task assignment due to overload. the traditional implementation scans the network machines, classifies them into the available and free segments, and then selects the first available machine for task execution. balancing means that it makes the less loaded node take additional loads and makes the highly loaded node free from taking or accepting additional loads. An incoming task must be scheduled for execution even if all executing broker nodes are overloaded. this is worse case event that incurs the most delay. As mentioned at the end of the section, each broker listens to broker status messages as shown in Figure 2, that are periodically published by broker nodes in the fog network. these status messages contain information about the broker's maximum load capacity [MIPS], current load [LOAD], and time of status generation [TS]. The status message as captured by the Wireshark network analysis tool shown in Figure 3. Furthermore, these messages help the receiving node determine the delay along the path connecting the sending and receiving brokers. The status messages are published using the MQTT protocol and have the "NML_" + broker node name prefix, for example, "NML_Broker_loc10" is a Network Management Layer message published by the "Broker_loc10" node to other broker nodes to inform them about its current load status. The information obtained from the status message as shown above was used by the other receiving broker nodes to compile and generate a list of broker nodes that is sorted, depending on the goal of the simulation, from most suitable to least suitable broker nodes for task execution. The sorting of the broker nodes list is implemented as an extendable function to allow for finer simulation control. for this work, the [sort_by_least_delay] is used. other possible sorting criteria are possible, for example [sort_by_least_load], [sort_by_max_capacity], or a user-defined sorting function. As the sorting logic is implemented in C++ code, modifying the sorting function and/or its parameters requires recompiling and linking the simulation application. As with the traditional implementation of the throttled algorithm, querying the target machine's status, the Network Management Layer incurs a slight overhead on the network transport medium to disseminate the broker node status over the network. The publishing of Network Management Layer messages can be turned on or off in the scenario configuration file (".ini" file) via the configuration parameter [enableNML]. for the scope of this work, the Network Management Layer messages are published using MQTT QoS 0 mode, the overhead the Network Management Layer message incurs can be determined using the following equation 1:

N messages = number of broker nodes * number of messages sent per second (1)

The number of VM status request messages sent by the traditional implementation can be determined by equation 2 (this is known as overhead):

N messages = number of nodes * number of requests (2)

After deciding on the best candidate broker to handle an incoming task (which might very well be the scheduling broker node itself), the task is forwarded to the chosen handling broker. in case the chosen handling

broker is the same host, the task is simply queued for execution. if the chosen handling broker is a remote broker node the scheduling broker node uses a dedicated MQTT connection to forward the task to the remote broker node. The Network Management layer fulfills two key roles: the first is a storage for the collected/received status of broker nodes on the network and the second alleviates the coding effort needed to generate broker node's status requests and process incoming responses, the implementation of Network Management layer depends on OMNET++ scheduling API to provide timing events and INET framework API to create data storage structure. in reality, the overhead of scheduling a task to the broker node is nonzero, but for simplicity, the current implementation of the network management layer assumes it to be zero, as it is an order of magnitude smaller than the delays incurred by the network and task queuing. Figure 4 illustrates the modified throttled algorithm used in this work.

5. Network Management Layer (NML) Data Flow

NML messages are sent to known broker nodes directly over an MQTT TCP connection, this completely avoids the need for intermediate processing by other fog Brokers in the network hierarchy, at the cost of a slight increase in load on links, switches, routers, the traffic generated by the NML is periodic and capturing the status of each node at generation time. The traffic is dynamic in the sense it changes according to the status of the broker node, once a request is received, the scheduling broker first determines which handling broker in the fog network is available and better suited for handling the request and then forwards the request to that broker.

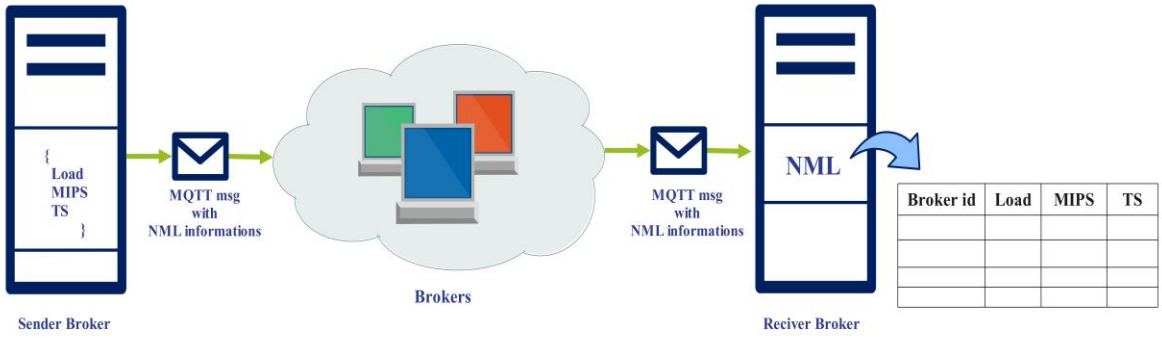

Figure 2. Broker Status Message.

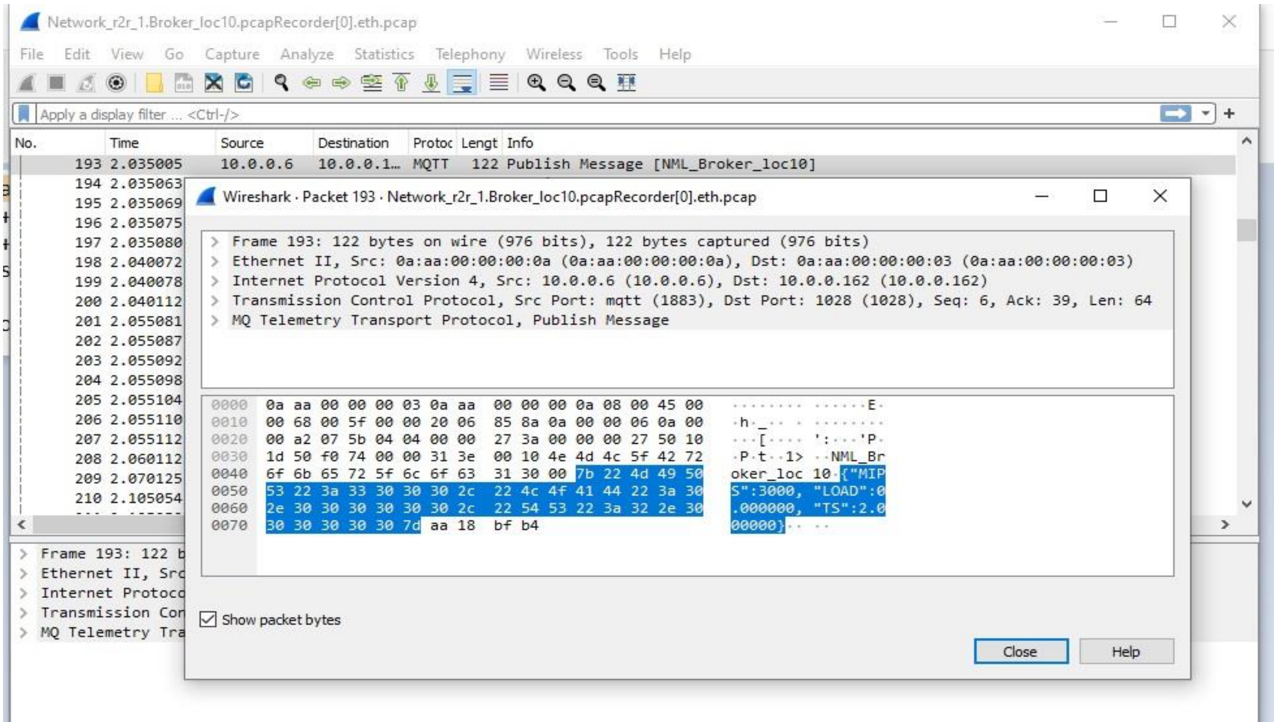

Figure 3. Capture of Broker Status Message.

```
--- BROKER-TO-BROKER SOCKET
01- Create brocker to broker socket
02- listen to NML status update messages
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
03- IF message == NML status update
04-    nml_map.update(message)
05- END IF
06- GOTO 02
---
---
---
---
---
---
---
```

```
--- TIMER
01- Create status publish timer
---
---
---
02- on NML timer event
03- nml_data = generate_status_message()
04- FOR each brokerSocket in brokerSocketMap
05-    IF brokerSocket is connected
06-       send(brokerSocket, nml_data)
07-    END IF
08- END FOR
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
---
```

```
--- BROKER-TO-CLIENTS SOCKET
01- Create brocker to clinets socket
02- listen to client messages
---
---
---
---
---
---
---
---
---
---
---
---
03- brocker_id = localhost
04- IF nml_map.size() != 0 THEN
05-    brocker_id = get_best_broker(nml_map)
06- END IF
07- schedue_client_message(brocker_id)
---
---
---
---
---
---
03- brocker_id = localhost
04- IF nml_map.size() != 0 THEN
05-    brocker_id = get_best_broker(nml_map)
06- END IF
07- schedue_client_message(brocker_id)
---
```

Figure 4. Modified Throttled Algorithm

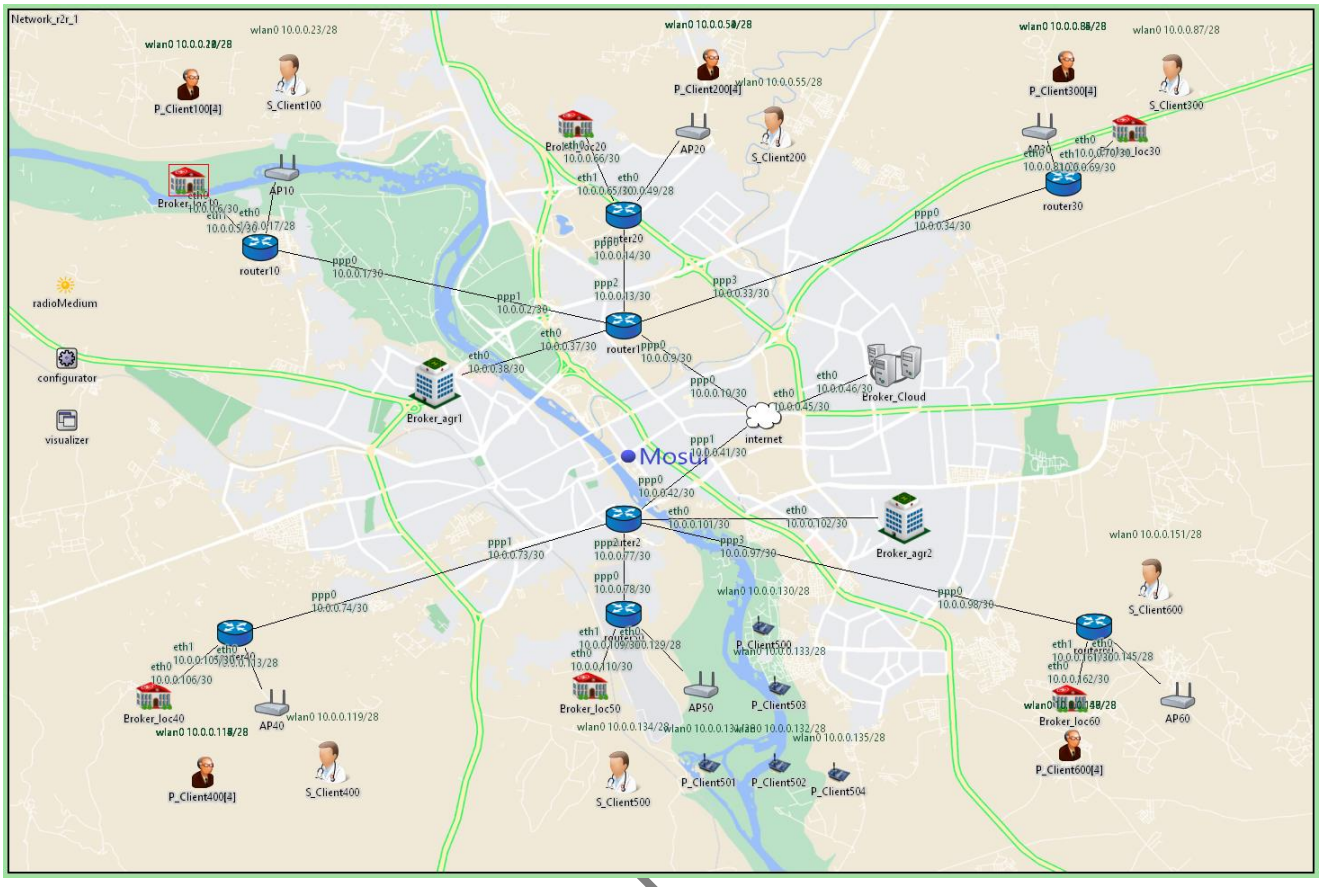

Figure 5 System Model

6. System Model Components

Numerous components were employed in the construction of the network in the proposed system, as shown in figure 5 which represents the fog computing environment of Mosul City's elderly care homes healthcare system. These components include:

a. IoT devices layer components consist of two main components:
   - Publisher nodes: which are responsible for generating medical sensor data and have the following static sensors: Blood Sugar, Blood Pressure, Temperature, Oxygen, and heart rate. The publisher symbol can be defined as a single patient or multiple patients configured by the [numusr] parameter.
   - Subscriber nodes: which act as the destination for sensor data.

b. Fog network layer which consists of two sublayers:
   - Local Fog nodes: responsible for receiving, combining, and scheduling data processing from end devices.
   - Aggregation Fog Broker: responsible for managing fog nodes, processing scheduled tasks, and distributing aggregated data messages from local fog nodes to other local nodes and the cloud data center.

c. The cloud layer is responsible for distributing published messages to aggregate fog brokers if they have a subscription for the data and can also process scheduled tasks.

d. An access point is a versatile access point that supports various wireless radios. It is provided by the INET framework and offers base station capabilities through the "Ieee80211MgmtAp" model. Configured to operate at 54 Mbps.

e. The router is an IPv4 router that is capable of supporting Ethernet and PPP interfaces. It is specifically setup to utilize static routing. The router is linked to local nodes by an ethernet interface and to faraway routers through PPP interfaces.

f. Radio Medium Model: The radio medium model is a component of the IEEE 802.11 physical layer model. It is necessary to utilize it together with the Ieee80211Radio model or any models that are developed from it. This model offers practical and reasonable preset values for the radio medium parameters that are utilized in IEEE 802.11 simulations.

g. Ipv4 Network Configurator: This module is responsible for allocating IPv4 addresses and configuring static routing for an IPv4 network. The system assigns IP addresses on a per-interface basis, prioritizes subnet considerations, and optimizes routing tables by consolidating routing entries.

h. Network Interface in INET simulations, network interface modules are the primary means of communication between network nodes. Network interfaces can be further categorized as wired and wireless; they conform to the IWiredInterface and IWirelessInterface NED types, respectively, which are subtypes of INetworkInterface. Wired network interfaces are compound modules that implement the

IWiredInterface interface. INET has many wired network interfaces such as Ethernet and PPP:

- PPP: this module is responsible for encapsulating network datagrams into PPP frames and decapsulating incoming PPP frames. It has the option to establish a direct connection with the network layer or can be set up to retrieve outgoing messages from an output queue. The module collected data on the transmitted and discarded packages. Wireless network interfaces are complex modules that support the IWirelessInterface interface, such as IEEE 802.11 and IEEE 802.15.4.

7. Task Execution Response Time

The time required for scheduling and executing a task within the fog network until the response is sent back to the scheduling fog node. This response time can be calculated according to equation 3.

Response Time = (response arrival time - msg scheduling time) (3) [29]

8. Results and Discussion

Three scenarios (25, 50, and 100 clients) are applied for QoS 1 of the MQTT protocol (Which have three levels of Quality of Services QoS0, QoS1, and QoS2) . The comparisons depend on two factors: the number of clients (publishers), each of whom has five sensors, and the resource specification (MIPS). The results are obtained for one broker only and calculated for the maximum load when the subscriber completes the subscription process for all the publishers (clients), below the three scenarios with their results.

- First Scenario when the number of clients (n) = 25:
In this scenario, two cases according to the MIPS are presented, the first one implemented using 3000 MIPS and the second one implemented using 6000 MIPS. Tables 2 and 3 include the results for these two cases. Each one of these two cases is presented in four sub-cases, using three algorithms (Random, Round Robin, and Modified Throttled (MTH)) and the no-load balance sub-case, as seen in two tables. Each one of these two tables includes many statistical results (mean, standard deviation (StdDev), minimum (Min), and maximum (Max) of the values).

Table 2. Response time of the scheduling algorithms, n = 25 with 3000 MIPS in QoS1.

Algorithms	Mean	StdDev	Min	Max
No load balance	0.2460377	0.1298361	0.003333	0.562399
Random	0.1243233	0.0493304	0.023573	0.363461
Round-Robin	0.1156414	0.0397468	0.023575	0.227693
MTH	0.1149815	0.0435132	0.023573	0.446891

Table 3. Response time of the scheduling algorithms, n = 25 with 6000 MIPS in QoS1

Algorithms	Mean	StdDev	Min	Max
No load balance	0.115886	0.061419	0.001666	0.2834
Random	0.085915	0.027781	0.021906	0.1937
Round-Robin	0.080996	0.024197	0.021924	0.1518
MTH	0.082441	0.026611	0.021906	0.2383

The Round-Robin and the Random algorithms are usually used in cloud and fog computing, while the throttled algorithm is used in cloud computing (usually obtained accepted results). In this work, a modified throttled (MTH) algorithm is proposed to be dependent on an active broker status collection algorithm within the fog computing environment, the essential algorithm (throttled) is enhanced, improved, and developed for further use, finally, the modified algorithm (MTH) is tested beside the other algorithms (Round-Robin and Random). The MTH algorithm has given better results when compared with the Random algorithm, and is very adjacent to the Round-Robin algorithm results (tables 2, and 3).

-Second Scenario when the number of clients (n) = 50:
The same as that of the first scenario, steps same steps are applied for 50 of clients instead of 25, also the modified algorithm (MTH) was better than the Random algorithm and very adjacent to Round-Robin as shown in tables (4, and 5), each one of these two tables includes many statistical results (mean, standard deviation (StdDev), minimum (Min), and maximum (Max) of the values).

Table 4. Response time of the scheduling algorithms, n = 50 with 3000 MIPS in QoS1.

Algorithms	Mean	StdDev	Min	Max
No load balance	0.4676908	0.2602113	0.003666	1.081894
Random	0.1736960	0.0784785	0.023568	0.529553
Round-Robin	0.1673693	0.0681790	0.023575	0.373850
MTH	0.1668997	0.0736710	0.023572	0.993809

Table 5. Response time of the scheduling algorithms, n = 50 with 6000 MIPS in QoS1.

Algorithms	Mean	StdDev	Min	Max
No load balance	0.2291594	0.1288844	0.001666	0.53376
Random	0.1100991	0.0400511	0.021906	0.27325
Round-Robin	0.10497932	0.03612002	0.0219068	0.22615
MTH	0.10752787	0.0379681	0.021906	0.50719

-Third Scenario when the number of clients (n) = 100:
Again, the same steps that were used in both the first and the second scenario are applied for this scenario with 100 clients, the modified MTH algorithm was doing well when tested beside the other algorithms (better than the Random algorithm, and very adjacent to the Round-Robin algorithm). Tables 6, 7 then figures 6, and 7 include the obtained results in this scenario.

Table 6. Response time of the scheduling algorithms, n = 100 with 3000 MIPS in QoS1

Algorithms	Mean	StdDev	Min	Max

No load balance	10.133673	5.8587900	0.0001776	24.907923
Random	0.2853683	0.1416781	0.0008949	0.8218739
Round-Robin	0.2757815	0.1278717	0.0001091	0.6173702
MTH	0.2739725	0.1294588	0.0002059	1.3480387

Table 7. Response time of the scheduling algorithms, n = 100 with 6000 MIPS in QoS1

Algorithms	Mean	StdDev	Min	Max
No load balance	0.452677	0.255464	0.001666	0.99328
Random	0.158580	0.066647	4.79E-05	0.41677
Round-Robin	0.154271	0.059658	6.8E-06	0.33125
MTH	0.155760	0.066747	5.59E-05	0.92558

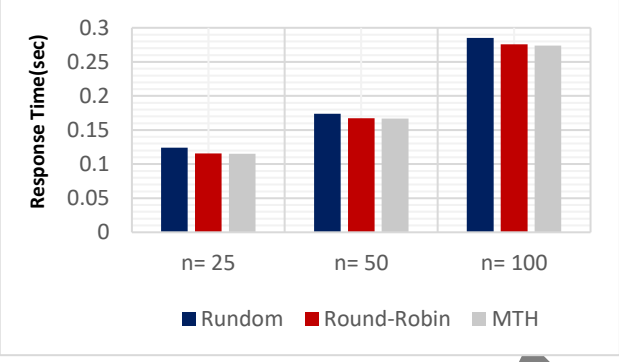

Figure 6. Response Time with 3000 MIPS

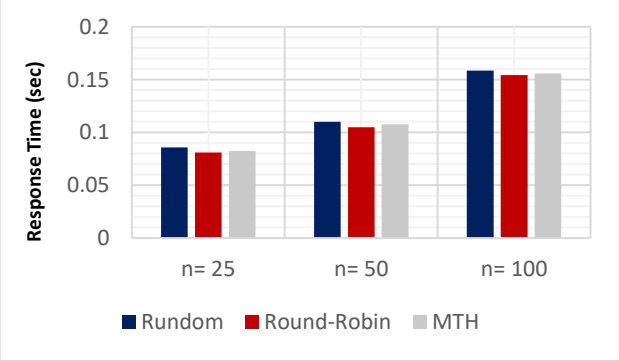

Figure 7. Response Time with 6000 MIPS

9. Conclusions

In this work, the evaluation and use of various load-balancing algorithms lead to a smooth distribution of computational tasks across the network. The appropriate load balancing algorithm will improve the effectiveness of the fog system due to better resource utilization; this effect was clear when the response time was reduced by using load balancing algorithms (Random, Round-Robin, and Modified Throttled) compared to the response time without the load balancing mechanism. Adding extra management layers to the traditional Throttled algorithm, made it more suitable to be employed in fog computing as the other dependent algorithms (Round-Robin, Random), according to the given response time results of the scheduling algorithm. The integration of cloud computing and fog computing adds new flexibility to network resources and traffic management. Some of the suggested future works and open problems are outlined below:

- Green fog computing to expand the breadth of the analysis to include the study of power and energy consumption analysis and find ways to optimize and reduce the overall consumption footprint and reduce harmful impacts on the surrounding environment.
- Develop advanced dynamic resource allocation algorithms to aid in the exploration and evaluation of many strategies for optimal resource allocation and sharing in a fog computing environment.
- Extending the management layer (NML) implementation to consider the response time delay.

Authors' contribution

All authors contributed equally to the preparation of this article.

Declaration of competing interest

The authors declare no conflicts of interest.

Funding source

This study didn't receive any specific funds.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments

The authors express their gratitude to the University of Mosul, College of Engineering, Computer Engineering Department, for their priceless help in enhancing the efficacy of this research.

REFERENCES

[1] A. M. and M. G. Rabeea Basir, Saad Qaisar, Mudassar Ali, Monther Aldwairi, Muhammad Ikram Ashraf, "Fog Computing Enabling Industrial Internet of Things: State-of-the-Art and Research Challenges," Sensors, Vol. 19, No. 21, p. 4807, Nov. 2019. https://doi.org/10.3390/s19214807

[2] V. N. H. Sabireen, "A Review on Fog Computing: Architecture, Fog with IoT, Algorithms and Research Challenges," ICT Express, Vol. 7, No. 2, pp. 162–176, Jun. 2021. https://doi.org/10.1016/j.icte.2021.05.004

[3] A. Varfolomeev and L. Al-Farhani, "Blockchain Fog-based scheme for identity authentication in smart building," Al-Qadisiyah J. Eng. Sci., Vol. 16, No. 3, pp. 218–227, Oct. 2023, https://doi.org/10.30772/qjes.1999.180617

[4] R. Neware and U. Shrawankar, "Fog Computing Architecture, Applications and Security Issues," Int. J. Fog Comput., Vol. 3, No. 1, pp. 75–105, Jan. 2020. https://doi.org/10.4018/IJFC.2020010105

[5] M. Rahimi, M. Songhorabadi, and M. H. Kashani, "Fog-Based Smart Homes: A Systematic Review," J. Netw. Comput. Appl., Vol. 153, No. C, p. 102531, Mar.2020. https://doi.org/10.1016/j.jnca.2020.102531

[6] O. Alani, T. Khaleel, and O. Al-Abdulqader, "A Review on Fog Computing: Research Challenges and Future Directions," Al-Rafidain Eng. J., Vol. 28, No. 1, pp. 341–350, Mar. 2023. https://doi.org/10.33899/rengj.2022.136642.1211

[7] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on Fog Computing: Architecture, Key Technologies, Applications, and Open Issues," J. Netw. Comput. Appl., Vol. 98, No. C, pp. 27–42, Nov. 2017. https://doi.org/10.1016/j.jnca.2017.09.002.

[8] R. K. Naha, S. Garg, and A. Chan, "Fog Computing Architecture: Survey and Challenges," in Big Data-Enabled Internet of Things, Institution of Engineering and Technology, 2019, pp. 199–223. https://doi.org/10.48550/arXiv.1811.09047

[9] B. Negash, A. M. Rahmani, P. Liljeberg, and A. Jantsch, "Fog Computing Fundamentals in the Internet-of-Things," in Fog Computing in the Internet of Things, Cham: Springer International Publishing, 2018, pp. 3–13. https://doi.org/10.1007/978-3-319-57639-8_1

[10] A. M. Rahmani et al., "Exploiting Smart e-Health Gateways at The Edge of Healthcare Internet-of-Things: A Fog Computing Approach," Futur. Gener. Comput. Syst., Vol. 78, pp. 641–658, Jan. 2018. https://doi.org/10.1016/j.future.2017.02.014

[11] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog Computing Based Face Identification and Resolution Scheme in Internet of Things," IEEE Trans. Ind. Informatics, Vol. 13, No. 4, pp. 1910–1920, Aug. 2017. https://doi.org/10.1109/TII.2016.2607178

[12] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities," IEEE Wirel. Commun., Vol. 20, No. 6, pp. 91–98, Dec. 2013. https://doi.org/10.1109/MWC.2013.6704479

[13] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog Computing for the Internet of Things: A Survey," ACM Trans. Internet Technol., Vol. 19, No. 2, pp. 1–41, May 2019. https://doi.org/10.1145/3301443

[14] N. Mohan and J. Kangasharju, "Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations," in International Conference on Cloudification of the Internet of Things, IEEE, 2016, pp. 1–6. https://doi.org/10.1109/CIOT.2016.7872914.

[15] A. Brogi and S. Forti, "QoS-Aware Deployment of IoT Applications Through the Fog," IEEE Internet Things J., Vol. 4, No. 5, pp. 1185–1192, Oct. 2017. https://doi.org/10.1109/JIOT.2017.2701408

[16] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments," Softw. Pract. Exp., Vol. 47, No. 9, pp. 1275–1296, Sep. 2017. https://doi.org/10.1002/spe.2509

[17] T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid, and S. U. Khan, "FogNetSim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment," IEEE Access, Vol. 6, pp. 63570–63583, 2018. https://doi.org/10.1109/ACCESS.2018.2877696

[18] A. Varga, "OMNeT++," in Modeling and Tools for Network Simulation, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59. https://doi.org/10.1007/978-3-642-12331-3_3

[19] I. Lera, C. Guerrero, and C. Juiz, "YAFS: A Simulator for IoT Scenarios in Fog Computing," IEEE Access, Vol. 7, pp. 91745–91758, 2019. https://doi.org/10.1109/ACCESS.2019.2927895

[20] A. W. Malik, T. Qayyum, A. U. Rahman, M. A. Khan, O. Khalid, and S. U. Khan, "xFogSim: A Distributed Fog Resource Management Framework for Sustainable IoT Services," IEEE Trans. Sustain. Comput., Vol. 6, No. 4, pp. 691–702, Oct 2021. https://doi.org/10.1109/TSUSC.2020.3025021

[21] S. Majumder, E. Aghayi, M. Noferesti, Z. Pang, and M. Deen, "Smart Homes for Elderly Healthcare—Recent Advances and Research Challenges," Sensors, Vol. 17, No. 11, p. 2496, Oct. 2017. https://doi.org/10.3390/s17112496

[22] O. Debauche, S. Mahmoudi, P. Manneback, and A. Assila, "Fog IoT for Health: A new Architecture for Patients and Elderly Monitoring.," Procedia Comput. Sci., Vol. 160, pp. 289–297, 2019. https://doi.org/10.1016/j.procs.2019.11.087

[23] P. Singh, R. Kaur, J. Rashid, and S. Juneja, "A Fog-Cluster Based Load-Balancing Technique," Sustainability, Vol. 14, No. 13, p. 7961, Jun. 2022. https://doi.org/10.3390/su14137961

[24] R. Beraldi, C. Canali, R. Lancellotti, and G. Proietti Mattia, "Randomized Load Balancing under Loosely Correlated State Information in Fog Computing," in Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Nov. 2020, pp. 123–127. https://doi.org/10.1145/3416010.3423244

[25] V. Kashyap and A. Kumar, "Load Balancing Techniques for Fog Computing Environment: Comparison, Taxonomy, Open issues, and Challenges," Concurr. Comput. Pract. Exp., Vol. 34, No. 23, p. e7183, Oct. 2022. https://doi.org/10.1002/cpe.7183

[26] N. R. O. Al-Rubaie, R. N. N. Kamel, and R. M. Alshemari, "Simulating Fog Computing in OMNeT++," Bull. Electr. Eng. Informatics, Vol. 12, No. 2, pp. 979–986, Apr. 2022. https://doi.org/10.11591/eei.v12i2.4201

[27] D. B. Abdullah and H. H. Mohammed, "DHFogSim: Smart Real-Time Traffic Management Framework for Fog Computing Systems," in 4th International Conference on Advanced Science and Engineering (ICOASE), Sep. 2022, pp. 60–65. https://doi.org/10.1109/ICOASE56293.2022.10075605

[28] A. S. Kadhim and M. E. Manaa, "Hybrid Load-Balancing Algorithm for Distributed Fog Computing in Internet of Things Environment," Bull. Electr. Eng. Informatics, Vol. 11, No. 6, pp. 3462–3470, Dec. 2022. https://doi.org/10.11591/eei.v11i6.4127

[29] "Measurement — INET 4.5.0 documentation." https://inet.omnetpp.org/docs/showcases/measurement (accessed Oct. 25, 2023).