sustainability

MDPI

Article

Energy-Efficient and Secure Load Balancing Technique for SDN-Enabled Fog Computing

Jagdeep Singh [1,2], Parminder Singh [1,3,*], El Mehdi Amhoud [3] and Mustapha Hedabou [3]

[1] School of Computer Science and Engineering, Lovely Professional University, Phagwara 144411, India; jagdeepmalhi@gndec.ac.in
[2] Department of Information Technology, Guru Nanak Dev Engineering College, Ludhiana 141006, India
[3] School of Computer Science, Mohammed VI Polytechnic University, Ben Guerir 43150, Morocco; elmehdi.amhoud@um6p.ma (E.M.A.); mustapha.hedabou@um6p.ma (M.H.)
* Correspondence: parminder.16479@lpu.co.in

Abstract: The number of client applications on the fog computing layer is increasing due to advancements in the Internet of Things (IoT) paradigm. Fog computing plays a significant role in reducing latency and enhancing resource usage for IoT users' tasks. Along with its various benefits, fog computing also faces several challenges, including challenges related to resource overloading, security, node placement, scheduling, and energy consumption. In fog computing, load balancing is a difficult challenge due to the increased number of IoT devices and requests, which requires an equal load distribution throughout all available resources. In this study, we proposed a secure and energy-aware fog computing architecture, and we implemented a load-balancing technique to improve the complete utilization of resources with an SDN-enabled fog environment. A deep belief network (DBN)-based intrusion detection method was also implemented as part of the proposed techniques to reduce workload communication delays in the fog layer. The simulation findings showed that the proposed technique provided an efficient method of load balancing in a fog environment, minimizing the average response time, average energy consumption, and communication delay by 15%, 23%, and 10%, respectively, as compared with other existing techniques.

Keywords: fog computing; load balancing; software defined network; resource management; intrusion detection

check for updates

Citation: Singh, J.; Singh, P.; Amhoud, E.M.; Hedabou, M. Energy-Efficient and Secure Load Balancing Technique for SDN-Enabled Fog Computing. *Sustainability* **2022**, *14*, 12951. https://doi.org/10.3390/su141912951

Academic Editor: Enrique Rosales-Asensio

Received: 19 August 2022
Accepted: 19 September 2022
Published: 10 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The past two decades have witnessed continuous growth in the demand for greater network capacity. To cope with this growing demand, an enormous amount of research has been conducted to enhance the performance of both optical-fiber-based and wireless networks [1]. Moreover, in recent years, network traffic has been exponentially growing at an unprecedented pace. This enormous increase is mainly due to massive amounts of video streams and modern technologies such as cloud services, social networks, and the traffic generated from the IoT. Fog computing was developed to provide IoT networks with lower latency and higher-quality user services. Fog computing is a deconcentrated computing framework in which storage, data, and applications are placed between the data source and cloud. Fog is used in real-time situations, and information is required to be processed quickly. Fog computing is a hypervisor technology that provides a real-time user interference for IoT users. Fog computing provides the distribution of edge data centers, the geographical arrangement of nodes, and location awareness. Moreover, fog computing provides excellent privacy, security, bandwidth, and latency. Fog computing overcomes several issues faced by IoT users. It also solves the reoccurring issues experienced by cloud users. Nodes are spread in several formats in fog computing frameworks and can gather essential data from different IoT sensors. Real-time users obtain information after processing information in the fog-IoT framework [2]. Edge devices were developed near cloud servers

after the implementation of fog computing. Unique nodes are used in fog computing to transmit data between IoT devices and convey messages. This provides a framework for cloud servers and forms a perfect foundation for networking between the real-time user and the resource. The fog layer gives information to the cloud server, and automated resource management is considered for all QoS variables. Resource provisioning, resource scheduling, and load balancing come under the category of resource management [3,4].

Fog computing expands the ability of cloud servers to accept the challenges of conventional cloud computing [5]. In fact, fog computing provides a method for developing an architecture to receive information from various computer devices and deal with different load balances. A proper scheme has been designed that involves the use of dynamic service migration to balance the load in fog computing. The aim of this method is to ensure load balancing in all fog nodes and cloud servers. The end-user layer is the first layer of fog computing [6]. This layer comprises end-users that develop several requests. Sets of workloads are transferred to underutilized nodes. Nano-data servers, databases, and train routers are parts of this fog layer. The cloud layer catches data from the fog layer and stores it for upcoming purposes [7]. In addition, fog computing is essential to curb cyber-attacks, which are common in contemporary times. Security issues may affect the availability of fog computing resources in the fog layer. The availability of fog resources increases with the implementation of load balancing. With the upward trend of IoT in the current era of technology, secure load balancing is mandatory in fog computing environments [8,9].

The motivation for this research was the fact that energy-efficiency and security are two of the most important criteria for load balancing in fog computing [10,11]. Load balancing is a vital part of fog frameworks. The primary function of load balancing is to avoid a particular situation with overloaded or under-loaded fog nodes. Load balancing techniques can enhance QoS parameters such as efficiency, performance, response time, and energy utilization. Load balancing is essential to dividing a significant amount of data across several servers. Data resources may be successfully utilized by implementing the equitable division of work framework. The features of a load-balanced system are as follows: equal workloads for nodes, the effective utilization of data, enhanced performance of the server, low energy consumption, a low response time, and higher levels of user satisfaction. Load balancing effectively forwards requests from users across several servers. Flexibility is provided to the cloud servers with the help of load balancing. Any new server can be aligned to the framework at any time. Nodes promptly process the requests of end-users in such systems [12,13]. A load balancer keeps a regular record of the servers within the framework. When it receives work from users, it examines resource availability and then allocates the workload across all computer resources [14].

In recent years, the IoT has expanded, with a regular upward trend in real-time applications [15,16]. Therefore, the need for a fair distribution of tasks has increased in the fog environment. Load balancing guarantees that no resource is overused or overburdened. Through the equal dispensation of the workload, the operational cost can be curbed and end-user applications can be optimized. Accurate load balancing can extend the predicted value of a system. Under half-failure conditions, a load balancing architecture can be employed to enhance the workplace's failure tolerance. The simplest load balancing framework can be used to cut back the execution time of the workload [17].

This article is organized as follows. The related work based on load balancing within fog computing is discussed in Section 2. The load balancing mechanism and architecture of the proposed system are discussed in Section 3 based on load balancing in the fog-IoT environment. Section 4 covers the implementation of these techniques in the simulation setup based on the workload. The experimental setup and performance evaluation metrics are also discussed in Section 4. The results are explained in Section 5. Section 6 concludes the paper and sets forth our research perspectives.

2. Related Work

In this section, we discuss the load balancing methods, algorithms, and techniques related to different aspects of fog computing. Table 1 summarizes the various proposed techniques, features, evaluation tools used, and the drawbacks identified by authors related to load balancing in the fog computing environment.

Table 1. Techniques for load balancing in fog computing proposed by various authors.

Article	Technique(s)	Environment	Features	Drawback(s)
[18]	Particle Swarm optimization	Arduino, Openssh	Maximum Resource Utilization	Greater Complexity, Lower Security
[19]	Whale Optimization Technique	NS2	Less delay, Energy Consumption, High Throughput	High Cost and Response Time
[20]	Hybrid Meta-Heuristic for Energy-Aware	NS2	High Scalability, Low Energy Consumption	More Complex
[21]	Load Balancing Scheme (LBS)	iFogSim	Low Complexity, Low latency, High Scalability	Less Reliable, Less Secure
[22]	Trust-Aware Offloading	Python	Low Complexity, Low Latency, High Security	Less Scaleable
[23]	Cluster-Enabled, Capacity-Based LB approach	NS2	Less Energy Consumption, Minimum Network Delay	Less Security, More Complexity
[24]	NAT-Based LB Method	Not Mention	Improved Utilization of Resources	Appropriate Evaluation Method not Used
[25]	Service Scheduling Approach	CloudSim	Lower Failure Rate, Reduced Bandwidth Consumption, More Privacy Issues	Less Scalability
[26]	PSW Fog Clustering LB Algorithm	iFogSim	Minimized Resource Wastage, Improved Execution Time	Low Scalability, and Reliability
[27]	Hybrid Priority-Assigned Laxity	iFogSim	Minimize Response Time, Delay Time, Processing Time	More Complex and Less Scalable
[28]	Fog-Cluster Based LB Method	iFogSim	Improve Execution Time, Resource Usage	More Energy Consumption, Missing Optimization Technique
[29]	Greedy and Coalitional Dame	NS3	Low Delay and Energy	Less Scalable, More Complexity

Kadhim et al. [30] worked on merging the IoT with a software-defined network (SDN) and fog computing to assist fog-IoT nodes for parked vehicles. The proposed method helped to decrease the number ofmigrated tasks and enhance the capacities of fog nodes. In addition, the load balancing method, proposed on the basis of a proactive approach, balanced the load globally and locally using SDN controllers and local fog managers. The simulation results showed that these methods were more effective than IoT-Fog-Cloud and VANET-Fog-Cloud.

Hameed et al. [23] proposed capacity-enabled and dynamic clustering load balancing techniques for energy-aware vehicular fog computing networks for IoT user requests. The authors divided a vehicular network into fog clusters to manage the fog resources and used inter- and intra-clustering load balancing for IoT jobs. The authors considered three scenarios—vehicles in an urban area, vehicles running on highways, and parked vehicles—for investigation and evaluation. The results showed that urban areas and parked vehicles had a high potential to use fog resources, whereas vehicles on the highway consumed higher energy and exhibited lower throughput. The authors also employed learning-based strategies to increase the network's performance and energy usage.

Karthik et al. [19] studied the use of a microgrid-connected wireless sensor network to optimize and assimilate a fog computing network. The developed FGWHO fog model was applied to evaluate the power generation, demand, and grid distance within the fog network. The grid-connected energy model was formed to assess the performance of girds. The proposed model enhanced the network's performance, including improvements in the residual energy, packet delivery ratio, and throughput parameters.

Maswood et al. [31] studied the integration of a fog-cloud to reduce the cost of resources and minimize delays in real-time applications and operations. The authors proposed an optimization model to provide load balancing, reduce bandwidth cost, and evaluate the efficient use of networks and fog-server resources. The simulation results were recorded in heterogeneous and homogeneous network resources as demand, generated by means of cluster points (CPs). The performance of the proposed model was investigated in terms of links and server utilization, bandwidth cost, and several machines used.

Beraldi et al. [32] first addressed the problem related to the distribution of loads in a fog environment and proposed adaptive forwarding and sequential forwarding algorithms that aimed to balance loads within the fog computing network. The authors considered controlled, simplified, and realistic scenarios for the design of a smart city fog computing infrastructure [33]. The performance evaluation of these algorithms was based on different responsible factors, such as the response time and the drop rate. To overcome the problem of the load distribution in fog nodes, Beraldi et al. [34] also suggested a probe-based load balancing method. They were primarily concerned with picking fog nodes to distribute incoming workloads. The suggested method was based on simulation and mathematical models. The authors presented two load-sharing algorithms that aimed to offload work to nearby nodes. The authors also tested the efficiency of the algorithm in a realistic scenario based on a real-world smart city monitoring application.

Rehman et al. [35] proposed an energy-efficient resource-allocation-based dynamic load balancing technique. The authors' primary focus was on offering an energy-aware load balancing solution for edge and fog devices. They evaluated simulation results using CloudSim. To evaluate the proposed technique, the authors looked at energy efficiency and cost characteristics and found that energy was decreased by 8.67%, and the cost was lowered by 16.77% when compared to the DRAM approach.

Singh et al. [36,37] discussed different load balancing techniques, algorithms, and taxonomies by conducting a comparative investigation based on their research. The authors concluded that the round robin load-balancing algorithm was the most simplistic method to implement in a fog computing network. The IP Hash load balancing algorithm was used in the fog computing network. S.P. Singh et al. [38] proposed a fuzzy-based load balancing method. The authors created a fuzzy-based load-balancer with several levels of the fuzzy control architecture. The data center layer, fog access layer, fog device layer, and core layer were the four levels of the proposed load balancer. These athors were mainly concerned with fog network traffic splitting.

Sangaiah et al. [39] deployed intrusion detection systems (IDSs) as a protective measure in critical situations. Several metrics were utilized to assess IDSs. The most effective feature selection approach was used for identifying harmful and lawful activity. The authors designed this study to find an efficient feature selection strategy to improve the accuracy of the classifier in intrusion detection. The "Hybrid Ant-Bee Colony Optimization

(HABCO)" approach was developed and compared with different techniques to turn the attribute selection issue into an optimization process. Sangaiah et al. [40] also proposed a new protocol, "CLustering Multi-Layer Security Protocol (CL-MLSP)," with "Ad-hoc On-Demand Distance Vector (AODV)", to detect malicious nodes. The clustering approach determined the shortest distance for every node, related to mobility, dispersion, and energy. The efficiency of CL-MLSP was evaluated using NS2, and the parameters included the network lifespan, packet loss, latency, and security.

Abbasi et al. [41] suggested a four-layer fog model with smart grid integration. Their effective system's primary goal was to manage the smart grid's resources, with recommended design and work dispersed around the globe, such as in six different continents. The round-robin (RR), particle swarm optimization (PSO), active virtual machine (VM), and throttled methods for load balancing were applied in this study, and their results were evaluated based on their cost.

Problem and Motivations

The load balancing process in fog computing consists of three main stages: data gathering, correct choice, and data movement. In the first step of the load balancing process, the data have to be gathered for task allocation and the identification of the imbalanced load. Then, the best feasible distribution of the data is determined in the second stage. Subsequently, data are communicated from an overloaded node to an underloaded node in the final stage. In fog computing, the paradigm of an optimization approach with load balancing is applied to minimize the chances of unnecessary allocation [36].

IoT devices produce a massive quantity of data, which have to be analyzed quickly. Fog nodes require energy, both when they are active and when they are idle. Based on previous surveys, a few solutions need to be proposed to lower the energy usage of the fog-IoT-based model. An effective technique is required to aid the appropriate usage of the fog node. The end-users may experience issues when processing queries due to the overcrowded requests from a large number of IoT devices, so the fog-IoT request needs to have improved response and execution times.

The amount of electric power consumed by fog nodes is defined as the energy consumption at the fog computing layer. In the fog environment, different types of devices, such as servers, gateways, and routers, use energy when executing activities. Load balancing is primarily used to scale down the total energy consumption of the fog computing system. Other factors also contribute to the cost of sustaining resources saved by load balancing in the fog environment, including energy usage and maintenance costs. If some resources are used more heavily, but others are underused, maintenance is required for these resources. As a result, the maintenance cost will be high, and load balancing is essential to lower the maintenance and implementation costs [42].

As per the above-discussed research, we can outline our objectives in relation to load balancing in fog computing, with the overall aim of improving upon the performance of existing techniques, as follows:

- To increase the performance of the fog-IoT computing architecture and develop a technique for processing end-user queries in terms of energy consumption and cost-effectiveness.
- To propose a load-balancing technique to improve resource utilization with SDN-enabled fog computing in the fog layer and decrease task migration to the cloud layer.
- To implement the intrusion detection method with fog computing to reduce communication delays in the the fog layer and the workload of fog nodes.

3. Proposed Work

3.1. System Architecture

Figure 1 shows the proposed architecture of our energy-efficient, SDN-enabled load-balancing technique for fog computing. The system consists of a three-tier architecture model, i.e., the IoT device tier, the fog tier, and the cloud tier. In the IoT tier, different IoT de-

vices, such as health sensors, agriculture sensors, smartwatches, smart cars, and smart home devices, interact with the fog gateway. In the fog tier, different fog nodes are connected with a fog server within one region that belongs to one geographical area. The open-flow protocols support the fog device. In each fog region, one local fog server manager is connected to each fog node within that region. The fog server manages each node and stores information related to each task and the available resources. The fog nodes are divided into different regions depending on the node's storage capacity, computing power, and networking strength. The fog region may contain the nodes with high storage capacity, which are combined to make storage fog regions. Similarly, one fog server or cluster manager manages nodes with high computing capacity to make computing fog regions. Mixed fog resource regions with different fog nodes contain resources related to storage, computing, and networking to handle the job requirements of these resources. The fog server manager contains each connected node's information regarding the devices' capacity.

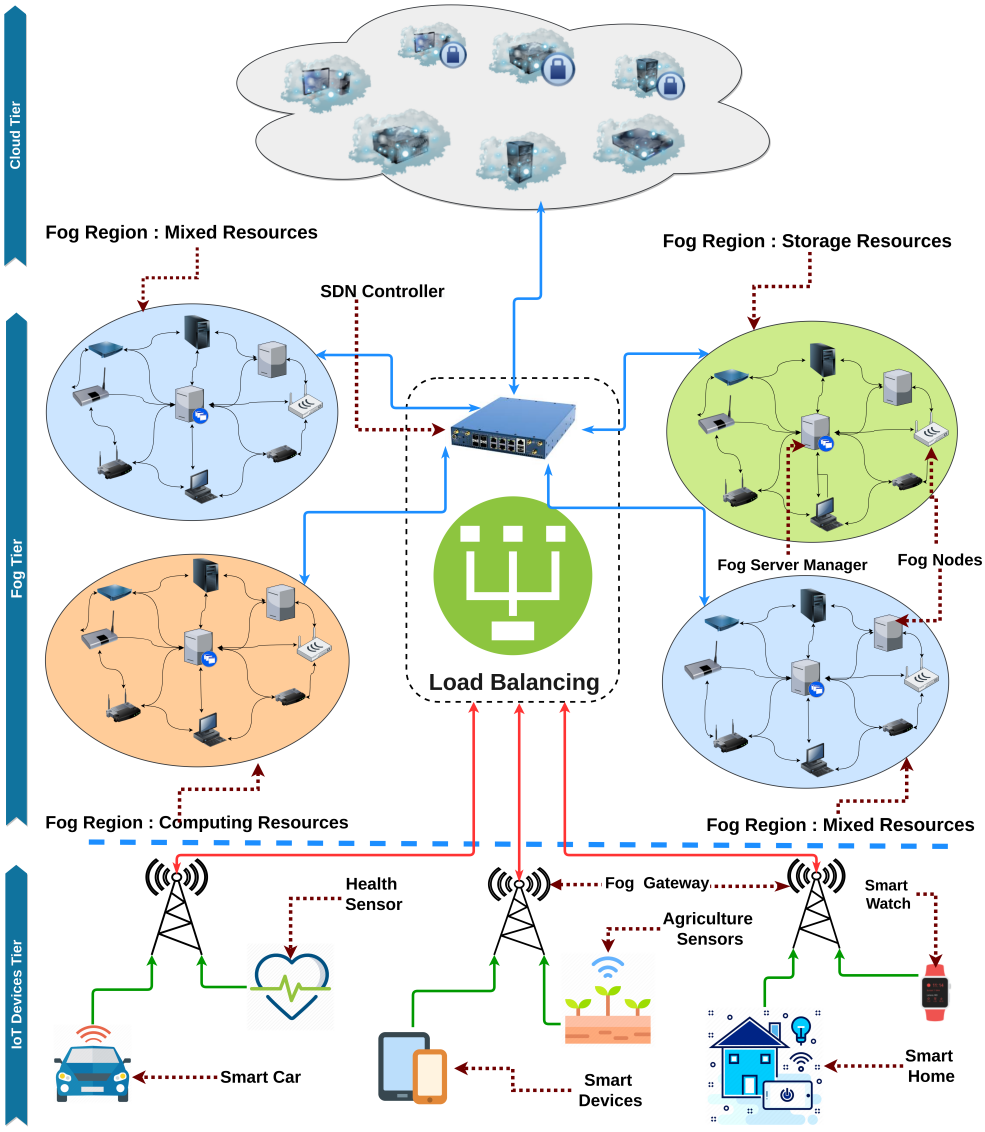

Figure 1. Proposed system architecture for load balancing in fog computing.

The different regions are connected by means of the SDN controller, enabling them to communicate with each other. Moreover, each fog server is also connected to an SDN controller to balance the loads of different fog regions. The SDN controller enables complete information network communication within fog regions and the fog-cloud layer. Furthermore, the fog regions are also connected with cloud services through the use of SDN controllers to communicate with the cloud tier devices with high computing requirements.

In our work, we make the following assumptions. (a) The fog servers at the fog layer are heterogeneous in nature. (b) The fog nodes support open-flow protocols. (c) The SDN controller and fog servers have complete information about each fog device's load, computing capacities, and task resource requirements. The IoT devices send the users' task requests for computing through the fog gateway to fog devices. The fog gateway creates a new link between the IoT and fog devices. These fog devices send information about the user's task to the nearby fog server and update the information based on the network connection within the fog region. Furthermore, they send a signal to the SDN controller to enable information updates. The clustering of the fog devices for each fog region is determined based on the Algorithm. At this point, fog servers manage the load at the fog node according to the user's task requests. The fog servers assist the fog nodes in executing the task after balancing the load within the fog region. The information about the task is updated to the SDN controller. If resources are unavailable within the fog region, the task will be migrated to another fog region with the support of fog servers and the SDN controller. The fog network's complete load is balanced with the help of the fog servers and the SDN controller [43].

3.2. Load-Balancing Mechanism

The IoT devices send the request through the fog gateway and connect to the upper fog layer to send task requests to fog nodes. The load-balancing technique is implemented on the fog layer with the help of a fog server or cluster head within the fog region or outer regions. First IoT devices send the requests to the fog node through the fog gateway. At the fog level, various heterogeneous resources, such as memory, CPU computation, networks, and storage, are accessible in order to satisfy user resource requirements. According to the proposed architecture, the fog server manager or cluster head is responsible for assigning resources according to the demands of the user tasks within the fog region. To balance the load of IoT requests, we first determine the resource requirements of each task, and classification and machine learning techniques are applied as already implemented in previous work [44]. After the identification of the task's resource requirements, the load balancer sends a request to the fog region regarding the demand. The fog region or cluster head is selected based on the available free resources and the task's resource demands. The fog regions are divided into high-storage-capacity regions, high-computing-capacity regions, and mixed-resource-capacity regions. For example, if the tasks require more computation, then the tasks are sent to the nearby computation fog region to complete the job. On the other hand, if the tasks require more storage, then the task is sent to the storage fog region.

After selecting the fog region, the fog gateway node connects to fog servers based on the calculated threshold value. The deadline of the task is considered the threshold value. In the selected fog region, the fog server manager connects to other fog nodes within the fog region to balance the loads of the tasks. The fog server manager keeps track of all available free resources within the fog region and provides the updated information required by the load balancer and the SDN controller [45]. The load balancer and SDN check the available fog regions to schedule the tasks. According to the network conditions and the number of nodes available, multiple fog regions are connected through the SDN controller. The fog servers and the SDN controller have complete information about the available resources and task requests, which plays an essential role in load balancing and the energy conservation for the complete system. The fog server manager or cluster head distributes the tasks to fog nodes within the region based on the capability of the fog nodes.

The task requests are further shifted from one fog region to another if the ideal load is achieved at the fog cluster head region. SDN also manages all fog region communication processes and connects to cloud services [46,47].

The prediction of the current-stage load was already achieved by Sharma et al. [48] based on the use of an artificial neural network (ANN). The ANN method is a fundamental and easy-to-implement machine-learning approach for predicting a fog node's current stage. This approach helps in distributing the workload to multiple fog nodes when one fog node is in an idle state or overloaded. The back-propagation learning algorithm (BPLA) has also been used to distribute the workloads on all fog nodes uniformly. This approach requires the availability of an adequate training set in order to manage the fog nodes effectively. The ANN can find out the present demands of different tasks and allocate resources accordingly. Load-balancing plays an active role in improving the energy consumption and throughput of the overall fog computing process. The input layer, hidden layer, and output layer are the three layers that the proposed approach works upon. The input layer processes the immediate load of 'm' fog nodes.

The computation of the instant load is expressed as in Equation (1):

$$Load(F_m) = \frac{\sum_{n=1}^{N} S(t)}{t} \tag{1}$$

Here, $S(t)$ represents the size of the task, and "t" is the time of the simulation. N is the number of tasks processed at the individual fog node F_m. The average load is computed as per Equation (2):

$$A_{Load^i} = \frac{\sum_{j=1}^{M} [RT_j(N) + TET(N)]}{M} \tag{2}$$

In the above equation, RT_j represents the remaining time of an instantly processed execution at the node. M is the number of fog nodes currently used at the instant "i". $TET(N)$ is the execution time of all tasks. According to the present workload, the weight value is processed in the hidden layer of each fog node. The balanced workload for all M fog nodes is predicted at the output layer.

Algorithm 1 shows the clustering mechanism and selection procedure for the fog region or cluster head F_{CH} in the fog layer. The initialization of fog nodes, the fog gateway, fog node capacity type, and fog region or cluster head is presented in the first five lines of the algorithm, and further steps show the selection procedure of F_{CH}. The minimal Euclidean distance between the fog gateway and the fog nodes is calculated. The selection criteria of F_{CH} are defined based on the following conditions: (a) if the fog node's distance D is inside the fog gateway's range, (b) verifying time T through a comparison with the threshold value. Furthermore, we check the capacity types of fog nodes in terms of storage, computing, and mixed resources. Furthermore, the fog cluster head (F_{CH}) is associated with the neighbor fog nodes to build the new cluster or fog region. This process is necessary to handle the dynamic nature of fog because the fog region/fog cluster head members may change in the fog layer.

Algorithm 1: Clustering fog devices for fog region C_{FD}.

```
Initialization: Number of fog nodes or devices ⟶ N
Fog gateway ⟶ F_g
Fog Region or cluster head ⟶ F_CH
Distance ⟶ D
Fog Node capacity type ⟶ FN_c
Procedure Cluster(C_N):
    for n to C_N number of fog-nodes do
        Euclidean distances: D (F_g, n)
        Minimum(D)
        if (T_time ≤ threshold) & (D ≤ range ) then
            if ( FN_c == Storage) then
                F_CH(Storage) ⟵ n
            else if ( FN_c == Computing) then
                F_CH(Computing) ⟵ n
            else
                F_CH(MixedResources) ⟵ n
            end
        end
    end
    for n to C_N number of fog-nodes do
        Euclidean distances: D (F_CH , n)
        if (D ≤ range) then
            Neighbor (n)
        end
    end
return C_FD
```

The fog gateway receives the task from the IoT layer after selecting the fog cluster head and then transmits the request to the fog server manager to balance the loads of IoT user tasks. Algorithm 2 shows the load-balancing technique in the fog layer. First, we initialize the number of fog cluster regions as F_{CR}, the head cluster region as H_{cr}, the neighbors of fog regions as N_{cr}, the capacity of F_{CR} as C, and distance as D. The procedure of the algorithm starts with a cluster (F_{CR}) up to a number of clusters from (n) to F_{CR}). If the capacity of the task is greater than $C(H_{cr})$, then it is necessary to find the neighboring region having the minimum Euclidean distance (D). The task is assigned to the region where the task capacity is less than that of the neighboring region cluster and which has a minimum Euclidean distance. The proposed technique manages the state of the ideal load and overloaded fog nodes to reduce the network congestion and resource/energy consumption level. The algorithm exhibits the load-balancing process within the fog region. Moreover, in the case in which the load is at the ideal level on the selected fog cluster/region head, requests are migrated to the nearest neighboring fog region as per the information available at the SDN controller. Table 2 represents the list of notations used in Algorithms.

Algorithm 2: Energy-efficient load balancing (L_B).

```
Initialization: Number of fog cluster Regions ⟶ F_CR
Head Cluster Region ⟶ H_cr
Neighbour of fog Regions ⟶ N_cr
Capacity of F_CR ⟶ C
Distance ⟶ D
Procedure Cluster(F_CR):
    for number of cluster from n to F_CR do
        if (C(H_cr) ≤ capacity(tasks)) then
            for Neighbour N_j of N_cr do
                if C(N_j) ≤ capacity(tasks) then
                    N_j ⟵ tasks
                else
                    H_cr ⟵ tasks
                end
            end
        else
            Euclidean distance - D(n_i1, n_j1)
            Minimum(D)
            if ((D ≤ range) and (T_time ≤ threshold)) then
                if (C(n_j1) ≤ capacity(tasks)) then
                    for Neighbour N_j of N_cr do
                        if C(N_j) ≤ capacity(tasks) then
                            N_j ⟵ tasks
                        else
                            H_cr ⟵ tasks
                        end
                    end
                end
            end
        end
    end
return L_B
```

Table 2. List of notations.

Symbol	Definition
T_I	Time interval at t^th instance
F_{CH}	Fog cluster head or fog region server manager
A_{It}	Active task(s) at T_I
F_g	Fog gateway
D_t	Departing tasks at the end of T_I
R_N	Networking resources
SDN	Software-defined network
S_e	Simulation Execution at interval t

The workflow of the load-balancing process in the fog layer is shown in Figure 2. It explains the steps from deploying the IoT and fog nodes to completing the task execution process in the fog layer. The primary process of the fog nodes starts with deploying IoT devices and fog nodes to handle user requests. The IoT devices send the user task requests through a fog gateway. Then, a DBN-based intrusion detection method is implemented

to reduce the workload and delay. The fog gateway also connects the nearby fog region through the use of the SDN. The fog cluster heads are selected within the fog region, and clustering is proposed based on the clustering technique. After this, the fog region's capacity is checked, and the load distribution is implemented per the task request within one or more fog regions. In the fog region, cluster heads or fog servers are already in place to handle task requests. The load-balancing technique can check the capacity of a fog region to handle requests; if it does not have sufficient capacity, the request is transferred to a nearby fog region. The execution of the task is completed in that region, and the response is sent back to the user.

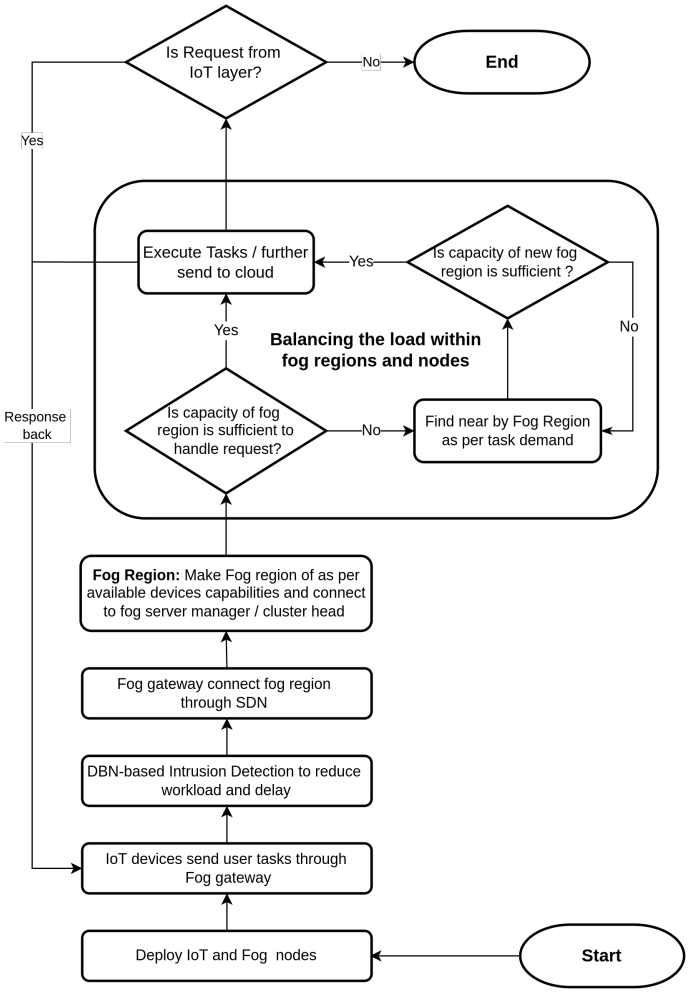

Figure 2. Workflow of load distribution in the fog layer.

In the fog layer, the resources are dynamic, and devices can move from one location to another. In fact, sometimes problems occur when the task request is sent to a node that is moving from one fog region to another fog region. In that situation, there is always a need to complete the task without any failure. To handle these problems, troubleshooting is required in order to regularly check and update the information at the fog server manager and the load balancer. The load balancer and SDN controller immediately shift the task to other available fog nodes or regions in that situation. Our proposed technique also handles the abovementioned troubleshooting problem and helps to balance the load within the fog layer.

3.3. Intrusion Detection

In the system architecture proposed above, intrusion detection is performed between the IoT and the fog layer after receiving the user requests. The data are exchanged among

several detection nodes to complete the detection operation. Algorithm 3 describes the basic steps involved in the intrusion detection method. In the first step, the data requests are received through the IoT-fog gateway and the deep belief network (DBN) and revamped DBN model are implemented to detect an intrusion in the IoT-fog computing environment [49]. The primary goal of the DBN is to find the DBN structure that provides the best rate of precision in the system. In the next step, the gateway sends the standard flag to the DBN and analyzes the data request. The control system manages the DBN outcomes and assesses the data to make a decision. In the next step, the gateway receives the results from the system to remove the false-flag data and takes the accurate flagged output of the data. Finally, it sends the actual flagged data to the load balancer and the fog server manager. We divided this categorization process into three parts to obtain the necessary level of accuracy. The training and testing datasets are created in the first stage by capturing incoming user requests. In the second stage, the incursion features are extracted. Finally, the DBN is used to classify the results. RBM is a crucial component of a deep belief network and this helps with pattern detection, data reconstruction, and data categorization. An undirected graph model's hidden and visible layers are used in RBM. The layers are not linked at the same level but are linked with separate layers.

Algorithm 3: IoT-fog-tier interaction for intrusion detection.

Start

1. IoT-fog gateway (FG) received the user data requests
2. Implementation of DBN-based Intrusion Detection Method
3. FG sends the Standard Flag to DBN
4. DBN analyzed data requests
5. Control System (CS) managed and received DBN-based outcomes
6. CS assessed data and made a decision
7. FG received the system's results and reduced the False flag data
8. True flag data is the final output
9. Send the True flag data to the fog server manager or Load balancer

end

4. Implementation and Evaluation

The implementation and evaluation of the proposed work were carried out based on the two scenarios discussed below. In Scenario I, the main focus was on implementing load-balancing techniques to evaluate the results. In Scenario II, the implementation of the proposed intrusion detection was carried out on the UNSW-NB15 dataset to reduce the workload.

4.1. Scenario I

We used the COSCO [50] simulator to set up a fog environment to implement the proposed work. This is a coupled simulation and container orchestration framework that works for integrated fog-cloud computing environments. It is a straightforward Python-based software solution that allows researchers to create, test, simulate, and deploy scheduling policies to balance the load. The proposed technique is Implemented with the seamless integration of load balancing and simulated back-ends for improved decision creating. Using a custom Stats logger, the simulation support system monitors metrics in real-time, logs them, and generates consolidated graphs. To implement the proposed technique, the object-oriented programming model is used in Python. A complete overview of different classes used in the implementation process is provided in Figure 3, and these work together to complete the simulation cycle. The Simulator class defines Balancer, Stats, host list, and taskList objects, which are further connected to the Task, Host, Stats, and Balancer classes. The different functions related to setting up the simulation, allocating tasks, migrating tasks, destroying tasks, and task placement are defined in the simulator class. The datacenter class is used to simulate the hosts that give the host's object list to the

simulator. The workload class is used to generate the workload with the support of static application trace data. The user has control over the number of jobs created during each pause. To complete the simulation, RAM, IPS, Bandwidth, and Disk classes are required to provide the Simulator class resources. The Host class is used to create virtual host objects with capacity and utilization metrics for RAM, IPS, Bandwidth, and Disk space. The Host class is further connected to the PowerModel class to measure the power consumption. Furthermore, the Load Balancer class provides for task selection and placement functions that help make effective load balancing decisions in the simulation process. The Simulator class is the central access point for running the simulation, which simulates fog jobs using a pre-implemented balancer, datacenter setups, workload model, and resource modules. The simulator class also allocates new workloads, determines if allocation or migration is required, runs event-driven simulations, and destroys finished workloads. The Stats class is used to monitor and obtain the log data from hosts, workloads, and containers during the complete execution of the simulation process.

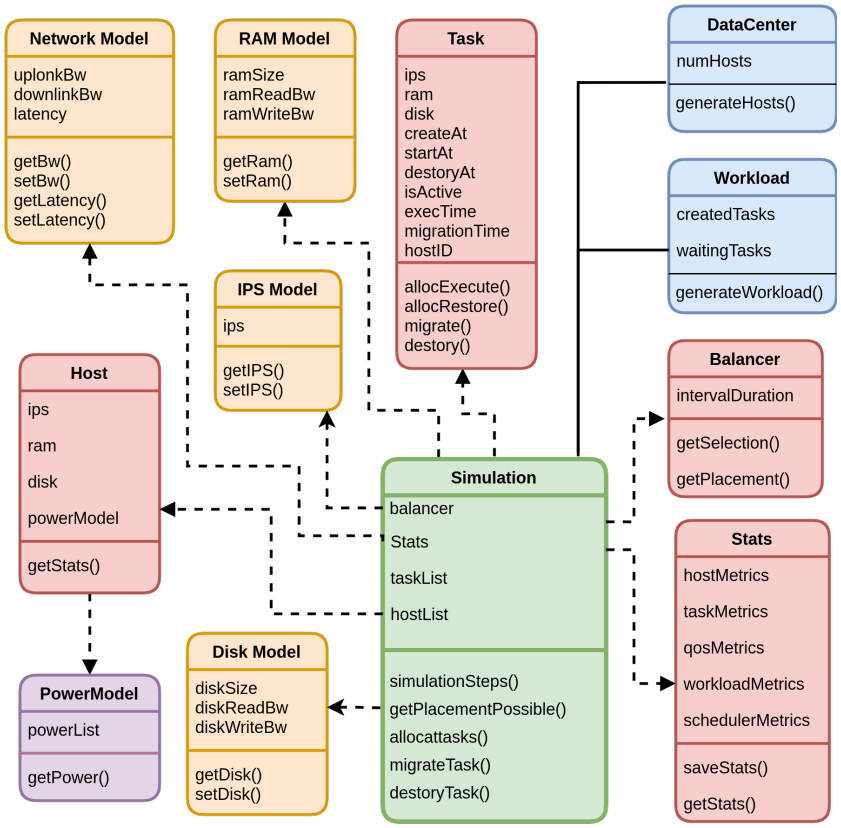

Figure 3. Workflow of the simulation environment.

4.1.1. Workload

The results are obtained using *Bitbrain traces* to generate the workload for the training and testing of the proposed technique in comparison baseline techniques [51]. This workload was considered due to its non-stationarity, its extremely fluctuating workload, and its resemblance to several real-world applications. The dataset contains realistic traces of resource utilization statistics from 1750 virtual machines (VMs) working on BitBrain's distributed data center. The workload applications operating on each of these systems come from a wide range of industries, such as computational statistical programs used by banks, insurers, and credit operators, which are utilized for evaluating fog-cloud modeling techniques. The dataset contains workload parameters for every timestamp, separated by five minutes, including CPU cores on-demand, CPU used in million instructions per second (MIPS), and RAM on demand with networking and read/write disc bandwidth features. These various types of workload traces are used to produce the trained data

and run the simulation process. The files of Bitbrain traces are organized by traces: Rnd and fast storage. The first trace, fast storage, is made up of 1250 VMs, which are linked to high-speed storage area network (SAN) devices. The second trace, Rnd, is made up of 500 VMs linked to rapid SAN equipment or slower network attached storage devices. Because of the increased performance of the storage connected to the fast storage devices, the fast storage traces contain a greater proportion of compute nodes than the Rnd traces. In contrast, the Rnd traces show a higher proportion of management machines that only involve space, with reduced performance and far less frequent availability.

4.1.2. Experimental Setup

The simulation setup was carried out using the following hardware and software, in order to conduct the experiments. The results used in this paper were obtained based on the following setup.

- Software details: Operating System: Ubuntu 22.04.3 LTS 64 bit. Run-time programming environment: Python 3.8.10. Infrastructure Management software: Influxdb, Ansible 2.0, Grafana, and PIP Packages based on Python libraries and Vagrant.
- Hardware details: Processor: IntelCore i5-4200M CPU @ 2.50 GHz × 4, RAM Memory: 16 GB 3200 MHz, Disk: PCI Gen6 SSD and GPU: Intel HD-Graphics 4600 (HSW GT2 Manufacturer: Intel Corporation, Santa Clara, CA, USA).

In the simulation environment, the experiments were run based on 50 host machines, which represented a scaled-up version of ten machines. On a considerably larger scale, the fog computing environment was scrutinized as per a previous study [52], in which each category had five times the number of instances, resulting in a total of fifty machines. Because we could not install simulated nodes in far-flung places, we represented their networking features and latency as shown in Table 3, which was used in the simulator. The experiments were conducted for a total of 100 scheduled intervals, each of which was 300 s long.

Table 3. Host features in the fog environment.

Server Name	Quantity	Core Count	RAM	MIPS	Network Bandwidth	Ping Time	Disk Bandwidth
Fog Layer							
S1	2	4	16 GB	8102	1024 MB/s	2 ms	10 MB/s
S2	4	2	4 GB	4029	1024 MB/s	3 ms	13 MB/s
Cloud Layer							
C1	2	8	64 GB	2100	2500 MB/s	70 ms	13 MB/s
C2	2	6	16 GB	8102	1000 MB/s	7 ms	11 MB/s

4.2. Scenario II

The main focus in implementing the proposed intrusion detection mechanism was reducing the workload and communication delays. The UNSW-NB15 [53] dataset contains assaults that are both routine activities and synthetic contemporary assaults. The Tcpdump instruments were used to create a bundle of the dataset's rudimentary system, which included 49 characteristics. Twelve algorithms, including Bro-IDS and Argus, were used to generate the class marks. There were 25,400,443 records in total. Various leveling approaches were used to partition the whole dataset into training and test sets. There were 175,341 entries in the preparation dataset. The testing dataset contained 82,332 entries. The allocated information collection had 43 features, with six highlights from the whole dataset, and was divided into 10 categories. Among the nine attacks were worms, secondary passages, shellcode, DoS, inspection, observation, abuse, nonexclusive, and fuzzes. The new training records for balancing the UNSWNB15 dataset are shown in Figure 4.

Due to unbalancing difficulties in the training instances, oversampling was necessary for the UNSW-NB15 data. An oversampling strategy was used to balance the training data. To balance the training data, new entries were created. Figure 4 depicts the balanced training data achieved via an oversampling method. To assess the effectiveness of the planned and current approach, a total of 20 snort nodes were placed in the network. Snorting alerts may be used to extract a set of eight characteristics (destination and source port no., destination and source Internet Protocol (IP) address, priority, categorization, description, and packet type). Wang et al. [54] (ML-EdgeIDS) conducted similar experiments to investigate differences in delay and workload.

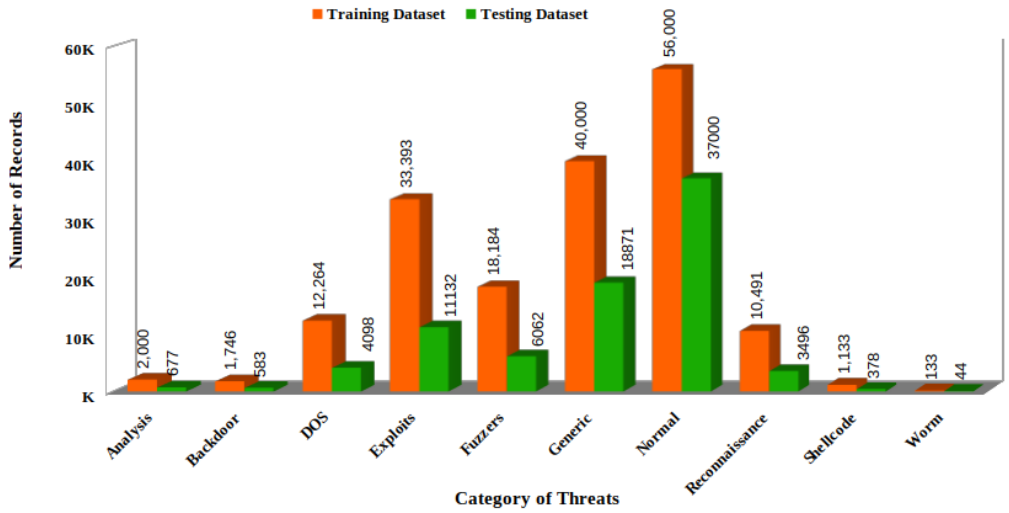

Figure 4. Balancing the UNSW-NB15 dataset with new training records.

4.3. Performance Evaluation Metrics

The experimentation was carried out using the COSCO simulation platform to compare the performance of the proposed technique with the baseline methods. To evaluate the performance of the proposed technique, we focused on the response time and energy consumption within the fog computing environment.

4.3.1. Average Energy Consumption (A_{EC})

The average energy consumption was formalized as the energy consumption of different fog and IoT devices at any interval of time T that is normalized by the maximum power of the nodes. Equation (3) is defined as follows at the instant T :

$$A_{EC}T = \frac{\sum_{d_i \epsilon D} \int_{t=S_e(T_I)}^{S_e(T_{I+1})} Power_{d_i}(T)dt}{|A_{It}| \sum_{d_i \epsilon D} Power_{d_i}^{max} \; x \; (T_{i+1} - T_i)} \tag{3}$$

Here, $Power_{d_i}(T)$ denotes the power function of device host(s) d_i at period interval T, and $Power_{d_i}^{max}$ denotes the maximum potential power of host d_i and D is the set of host(s) in the resource layer.

4.3.2. Average Response Time (A_{RT})

The average response time for all departing tasks (D_t), normalized by the maximal response time up to the present interval, was defined as the average response time for an interval T_I. Equation (4) was defined as follows:

$$A_{RT}T = \frac{\sum_{d_t^j \epsilon D_t} ResponseTime(d_t^j)}{|D_t| max_{S_e \leq t} \ max_{d_j^{S_e} \epsilon D_{S_e}} ResponseTime(d_j^{S_e})} \quad (4)$$

4.3.3. Cost per Container on Average

The performance was also evaluated based on the average cost per container in the simulation setup. Equation (5) defines the average cost per container on different tasks:

$$AC_T = \frac{\sum_T \sum_{d_i \epsilon D} \int_{T=T_i}^{T_{i+1}} C_{d_i}(T)dt}{\sum_T |D_t|} \quad (5)$$

$C_{d_i}(T)$ denotes the cost function of the device host d_i at time interval T.

5. Results and Discussions

In this section, we examined the outcomes of the proposed technique using simulations. Figure 5 illustrates the average energy consumption with 50 hosts, demonstrating the differences between balanced and imbalanced loads. The energy consumption increased in both balanced and imbalanced loads when the number of SIM intervals increased. Overall, it was seen that both imbalanced loads consumed more energy compared to balanced loads in the entire chart as the number of SIM interval values increased. The average energy consumption of the balanced load technique was lower compared to the imbalanced technique.

Figure 6 illustrates the average execution time with the simulation of 50 hosts. The figure represents the different results obtained when the balanced and imbalanced load techniques were implemented in the simulation. In summary, when the number of SIM interval values increased, balanced and imbalanced load techniques showed an increasing trend across the entire chart. Compared to unbalanced load techniques, balanced load techniques took less time to execute. Figure 7 illustrates the average response time with a 50-host simulation setup, desmonstrating the results of the balanced and imbalanced load techniques. The balanced load technique took a shorter response time to complete the task than the imbalanced load technique. The graph also shows that the response time increased up when the number of SIM intervals increased for the balanced and imbalanced loads. In summary, balanced load approaches were more time-efficient compared to imbalanced load approaches, as shown by the increased SIM values required to complete the tasks.

Figure 8 illustrates the average cost per container on the simulator with 50 hosts. The bar chart indicates the differences between the balanced and imbalanced load balancing techniques. It can be seen that both balanced and imbalanced loads showed an upward trend in the entire chart when the simulation intervals were increased. The average cost of the balanced approach was slightly higher at 10 SIM intervals, but it decreased when the SIM intervals were increased. Overall, the imbalanced load techniques had a higher average cost as compared to the balanced techniques.

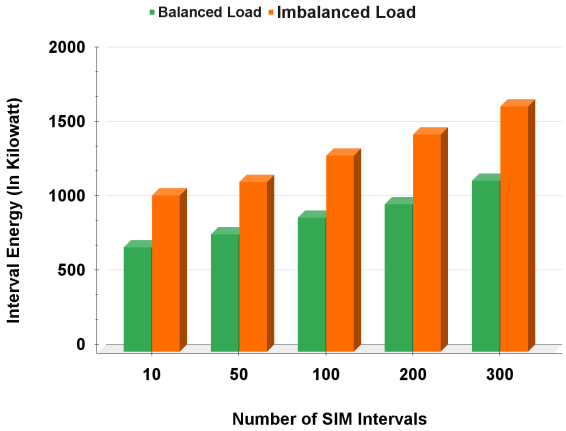

Figure 5. Average energy consumption with 50 hosts.

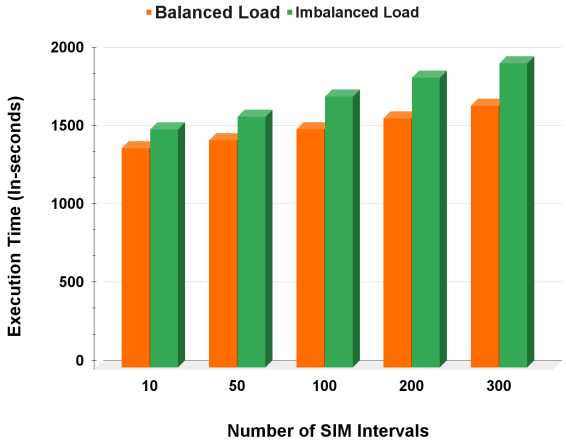

Figure 6. Average execution time with 50 hosts.

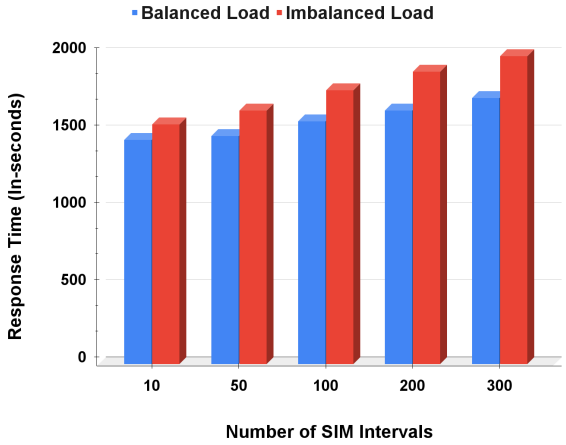

Figure 7. Average response time with 50 hosts.

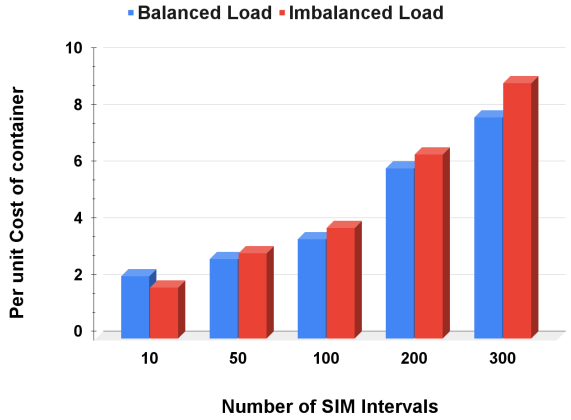

Figure 8. Average cost per container on a simulator with 50 hosts.

Figure 9 shows a comparison of the average response time obtained with the baselines techniques A3C, LR-MMT GA, GOBI, MAD-MC, POND, and DQLCM on the simulator with 50 hosts, which were already discussed by the authors in [50]. The POND techniques exhibited the highest average response time compared with the other techniques. The A3C and GA techniques took a similar time to complete the task with a simulation setup of 50 hosts. The proposed technique exhibited a lower average response time as compared with the others. Similarly, Figure 10 shows a comparison of the average energy consumption of other baselines techniques in the simulation with 50 hosts. The LR-MMT technique consumed much more energy than other techniques as per the experimental setup. The MAD-MC technique consumed the second highest amount of energy. The GOBI, A3C, and DQLCM techniques consumed almost the same amount of energy, with only minor differences between them. The proposed technique consumed less energy, compared with the other techniques.

We compared the proposed intrusion detection method to the most closely related research to assess the performance. Wang et al. [54] improved on this research by incorporating machine learning into the edge-fog computing environment. Figure 11 presents a comparison between the workload reduction and communication delay improvement based on network interactions and the available data. In comparison to ML-EdgeIDS [54], the proposed method was able to reduce the workload by up to 7% and improve delays by up to 5% with an equal processing capacity. The reason for this enhancement is resource stability, as fog-IoT computing devices connect to the network on an ad hoc basis. Another major challenge with fog computing is network capacity. The experimental assessment revealed that the suggested method was capable of improving the false alert rate with DBN, as well as reducing delays and workload reductions in the fog-IoT environment.

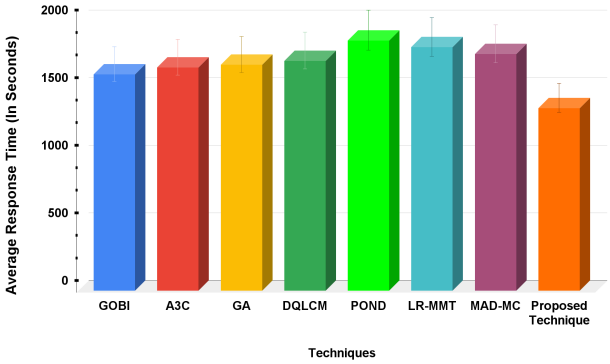

Figure 9. Comparison of the average response time of the proposed technique and that of other baseline techniques in the simulation with 50 hosts.

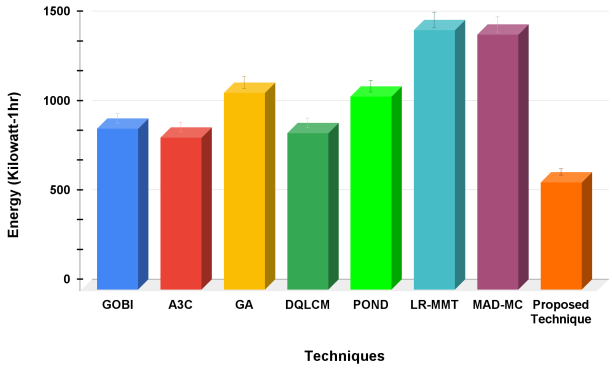

Figure 10. Comparison of average energy consumption of the proposed technique and that of other baseline techniques in the simulation with 50 hosts.

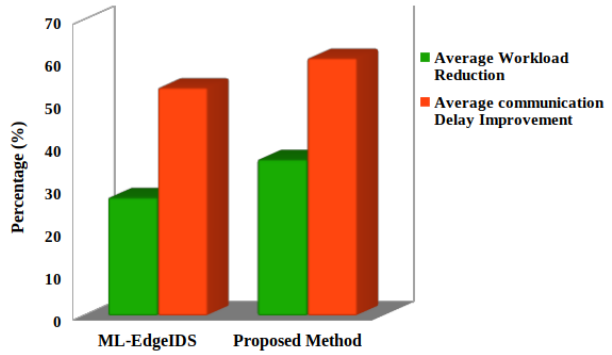

Figure 11. Comparison of workload and communication delays in ML-EdgeIDS and the proposed technique.

6. Conclusions

In this paper, we have proposed the energy-efficient and secure load balancing technique in an SDN-enabled fog computing environment. Due to the massive increase in IoT devices and user applications, balancing the load in the fog layer is essential. We proposed the energy-aware fog-IoT-based computing architecture and load-balancing technique to improve resource utilization by means of SDN-enabled fog computing at the fog layer and to reduce task migration at the cloud layer. Moreover, we implemented an intrusion detection method to reduce the workload and communication delays in the fog layer. Experiments were conducted in a simulation environment, and the method's performance was assessed considering the energy consumption, average cost, workload reduction, and average response time as metrics. The results were compared with the imbalanced and balanced load balancing techniques. The balanced technique showed improvements in terms of the energy consumption, cost, and average response time. The results of the proposed technique were also compared with those of other baselines methods implemented by other authors. The simulation experiments showed improvements in the average response time, average energy consumption, and communication delays of 15%, 23%, and 10%, respectively. In future works, we propose to extend this work with more security and performance parameters. To address the limitations of the proposed technique, this can be implemented with a real test-bed environment and optimization techniques. In addition, there is a need to improve the integration of IoT, edge, dew, fog, and cloud environments to reduce the workload and delays of the overall network.

Author Contributions: Conceptualization, J.S.; data curation, E.M.A.; formal analysis, J.S. and P.S.; funding acquisition, E.M.A. and M.H.; investigation, J.S.; methodology, J.S. and P.S.; project administration, E.M.A. and M.H.; resources, E.M.A.; supervision, P.S. and M.H.; validation, P.S. and M.H.; writing—original draft, J.S.; writing—review & editing, E.M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Amhoud, E.M.; Chafii, M.; Nimr, A.; Fettweis, G. OFDM with Index Modulation in Orbital Angular Momentum Multiplexed Free Space Optical Links. In Proceedings of the 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), Helsinki, Finland, 25–28 April 2021; pp. 1–5.
2. Wan, J.; Chen, B.; Wang, S.; Xia, M.; Li, D.; Liu, C. Fog Computing for Energy-Aware Load Balancing and Scheduling in Smart Factory. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4548–4556. https://doi.org/10.1109/TII.2018.2818932.
3. Li, C.; Zhuang, H.; Wang, Q.; Zhou, X. SSLB: Self-Similarity-Based Load Balancing for Large-Scale Fog Computing. *Arab. J. Sci. Eng.* **2018**, *43*, 7487–7498. https://doi.org/10.1007/s13369-018-3169-3.
4. Fan, Q.; Ansari, N. Towards Workload Balancing in Fog Computing Empowered IoT. *IEEE Trans. Netw. Sci. Eng.* **2018**, *7*, 253–262. https://doi.org/10.1109/TNSE.2018.2852762.
5. Hedabou, M. Cryptography for Addressing Cloud Computing Security, Privacy, and Trust Issues. In *Computer and Cyber Security*; Auerbach Publications: New York, NY, USA, 2018; pp. 281–304.
6. Singh, J.; Singh, P.; Gill, S.S. Fog Computing: A Taxonomy, Systematic Review, Current Trends and Research Challenges. *J. Parallel Distrib. Comput.* **2021**, *157*, 56–85. https://doi.org/10.1016/j.jpdc.2021.06.005.
7. Sharma, S.; Saini, H. A novel four-tier architecture for delay aware scheduling and load balancing in fog environment. *Sustain. Comput. Inform. Syst.* **2019**, *24*, 100355. https://doi.org/10.1016/j.suscom.2019.100355.
8. Zahid, M.; Javaid, N.; Ansar, K.; Hassan, K.; KaleemUllah Khan, M.; Waqas, M. Hill Climbing Load Balancing Algorithm on Fog Computing. In *Lecture Notes on Data Engineering and Communications Technologies*; Springer International Publishing: Cham, Switzerland, 2019; Volume 24, pp. 238–251. https://doi.org/10.1007/978-3-030-02607-3_22.
9. Téllez, N.; Jimeno, M.; Salazar, A.; Nino-Ruiz, E.D. A Tabu search method for load balancing in fog computing. *Int. J. Artif. Intell.* **2018**, *16*, 106–135.
10. Lin, Z.; Lin, M.; Wang, J.B.; de Cola, T.; Wang, J. Joint Beamforming and Power Allocation for Satellite-Terrestrial Integrated Networks With Non-Orthogonal Multiple Access. *IEEE J. Sel. Top. Signal Process.* **2019**, *13*, 657–670. https://doi.org/10.1109/JSTSP.2019.2899731.
11. Lin, Z.; An, K.; Niu, H.; Hu, Y.; Chatzinotas, S.; Zheng, G.; Wang, J. SLNR-based Secure Energy Efficient Beamforming in Multibeam Satellite Systems. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, 1–4. https://doi.org/10.1109/TAES.2022.3190238.
12. Talaat, F.M.; Ali, S.H.; Saleh, A.I.; Ali, H.A. Effective Load Balancing Strategy (ELBS) for Real-Time Fog Computing Environment Using Fuzzy and Probabilistic Neural Networks. *J. Netw. Syst. Manag.* **2019**, *27*, 883–929. https://doi.org/10.1007/s10922-019-09490-3.
13. Manju, A. Efficient Load Balancing Algorithm for Task Preprocessing in Fog Computing Environment. *Smart Intell. Comput. Appl.* **2019**, *2*, 291–298. https://doi.org/10.1007/978-981-13-1927-3.
14. Kashani, M.H.; Ahmadzadeh, A.; Mahdipour, E. Load balancing mechanisms in fog computing: A systematic review. *arXiv* **2020**, 1–19. arXiv:2011.14706.
15. Lin, Z.; Lin, M.; De Cola, T.; Wang, J.B.; Zhu, W.P.; Cheng, J. Supporting IoT with rate-splitting multiple access in satellite and aerial-integrated networks. *IEEE Internet Things J.* **2021**, *8*, 11123–11134.
16. Lin, Z.; Niu, H.; An, K.; Wang, Y.; Zheng, G.; Chatzinotas, S.; Hu, Y. Refracting RIS aided hybrid satellite-terrestrial relay networks: Joint beamforming design and optimization. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 3717–3724.
17. Kaur, M.; Aron, R. *A systematic Study of Load Balancing Approaches in the Fog Computing Environment*; Springer: New York, NY, USA, 2021; Volume 77, pp. 9202–9247. https://doi.org/10.1007/s11227-020-03600-8.
18. Baburao, D.; Pavankumar, T.; Prabhu, C. Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Appl. Nanosci.* **2021**. https://doi.org/10.1007/s13204-021-01970-w.
19. Karthik, S.S.; Kavithamani, A. Fog computing-based deep learning model for optimization of microgrid-connected WSN with load balancing. *Wirel. Netw.* **2021**, *27*, 2719–2727. https://doi.org/10.1007/s11276-021-02613-2.
20. Qun, R.; Arefzadeh, S.M. A new energy-aware method for load balance managing in the fog-based vehicular ad hoc networks (VANET) using a hybrid optimization algorithm. *IET Commun.* **2021**, *15*, 1665–1676.
21. Asghar, A.; Abbas, A.; Khattak, H.A.; Khan, S.U. Fog Based Architecture and Load Balancing Methodology for Health Monitoring Systems. *IEEE Access* **2021**, *9*, 96189–96200. https://doi.org/10.1109/ACCESS.2021.3094033.
22. Mazumdar, N.; Nag, A.; Singh, J.P. Trust-based load-offloading protocol to reduce service delays in fog-computing-empowered IoT. *Comput. Electr. Eng.* **2021**, *93*, 107223.
23. Hameed, A.R.; ul Islam, S.; Ahmad, I.; Munir, K. Energy- and performance-aware load-balancing in vehicular fog computing. *Sustain. Comput. Inform. Syst.* **2021**, *30*, 100454. https://doi.org/10.1016/j.suscom.2020.100454.

24. Lai, C.F.; Weng, H.Y.; Chou, H.Y.; Huang, Y.M. A novel NAT-based approach for resource load balancing in fog computing architecture. *J. Internet Technol.* **2021**, *22*, 513–520. https://doi.org/10.3966/160792642021052203002.
25. Alqahtani, F.; Amoon, M.; Nasr, A.A. Reliable scheduling and load balancing for requests in cloud-fog computing. *Peer-to-Peer Netw. Appl.* **2021**, *14*, 1905–1916. https://doi.org/10.1007/s12083-021-01125-2.
26. Kaur, M.; Aron, R. An Energy-Efficient Load Balancing Approach for Scientific Workflows in Fog Computing. *Wirel. Pers. Commun.* **2022**, *125*, 3549–3573. https://doi.org/10.1007/s11277-022-09724-9.
27. Singh, S.P.; Kumar, R.; Sharma, A.; Abawajy, J.H.; Kaur, R. Energy efficient load balancing hybrid priority assigned laxity algorithm in fog computing. *Clust. Comput.* **2022**, *0123456789*. https://doi.org/10.1007/s10586-022-03554-x.
28. Singh, P.; Kaur, R.; Rashid, J.; Juneja, S.; Dhiman, G.; Kim, J.; Ouaissa, M. A Fog-Cluster Based Load-Balancing Technique. *Sustainability* **2022**, *14*, 7961. https://doi.org/10.3390/su14137961.
29. Yan, J.; Wu, J.; Wu, Y.; Chen, L.; Liu, S. Task Offloading Algorithms for Novel Load Balancing in Homogeneous Fog Network. In Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Dalian, China, 5–7 May 2021; pp. 79–84. https://doi.org/10.1109/CSCWD49262.2021.9437748.
30. Kadhim, A.J.; Naser, J.I. Proactive load balancing mechanism for fog computing supported by parked vehicles in IoV-SDN. *China Commun.* **2021**, *18*, 271–289. https://doi.org/10.23919/JCC.2021.02.019.
31. Maswood, M.M.S.; Rahman, M.R.; Alharbi, A.G.; Medhi, D. A Novel Strategy to Achieve Bandwidth Cost Reduction and Load Balancing in a Cooperative Three-Layer Fog-Cloud Computing Environment. *IEEE Access* **2020**, *8*, 113737–113750. https://doi.org/10.1109/ACCESS.2020.3003263.
32. Beraldi, R.; Canali, C.; Lancellotti, R.; Mattia, G.P. Distributed load balancing for heterogeneous fog computing infrastructures in smart cities. *Pervasive Mob. Comput.* **2020**, *67*, 101221. https://doi.org/10.1016/j.pmcj.2020.101221.
33. Bentajer, A.; Hedabou, M.; Abouelmehdi, K.; Igarramen, Z.; El Fezazi, S. An IBE-based design for assured deletion in cloud storage. *Cryptologia* **2019**, *43*, 254–265.
34. Angfeng, J.D.; UiLi, H.; Aodong, X.X.; ngjun Shi, M.; HuiW, J.; AnyiZhou, X.; Ongdi an an, R.H. A Random Walk based Load Balancing Algorithm for Fog Computing. In Proceedings of the Fog and Mobile Edge Computing (FMEC), Paris, France, 20–23 April 2020; pp. 46–53. https://doi.org/10.1109/FMEC49853.2020.9144962.
35. Rehman, A.U.; Ahmad, Z.; Jehangiri, A.I.; Ala'Anzy, M.A.; Othman, M.; Umar, A.I.; Ahmad, J. Dynamic Energy Efficient Resource Allocation Strategy for Load Balancing in Fog Environment. *IEEE Access* **2020**, *8*, 199829–199839. https://doi.org/10.1109/ACCESS.2020.3035181.
36. Singh, S.P.; Kumar, R.; Sharma, A.; Nayyar, A. Leveraging energy-efficient load balancing algorithms in fog computing. *Concurr. Comput.* **2022** , *34*, e5913. https://doi.org/10.1002/cpe.5913.
37. Singh, J.; Warraich, J.; Singh, P. A Survey on Load Balancing Techniques in Fog Computing. In Proceedings of the 2021 International Conference on Computing Sciences (ICCS), Phagwara, India, 4–5 December 2021; pp. 47–52. https://doi.org/10.1109/ICCS54944.2021.00018.
38. Singh, S.P.; Sharma, A.; Kumar, R. Design and exploration of load balancers for fog computing using fuzzy logic. *Simul. Model. Pract. Theory* **2020**, *101*, 102017. https://doi.org/10.1016/j.simpat.2019.102017.
39. Sangaiah, A.K.; Javadpour, A.; Ja'fari, F.; Pinto, P.; Zhang, W.; Balasubramanian, S. A hybrid heuristics artificial intelligence feature selection for intrusion detection classifiers in cloud of things. *Clust. Comput.* **2022**, 1–14. https://doi.org/10.1007/s10586-022-03629-9.
40. Sangaiah, A.K.; Javadpour, A.; Ja'fari, F.; Pinto, P.; Ahmadi, H.; Zhang, W. CL-MLSP: The design of a detection mechanism for sinkhole attacks in smart cities. *Microprocess. Microsyst.* **2022**, *90*, 104504. https://doi.org/10.1016/j.micpro.2022.104504.
41. Abbasi, S.H.; Javaid, N.; Ashraf, M.H.; Mehmood, M.; Naeem, M.; Rehman, M. Load Stabilizing in Fog Computing Environment Using Load Balancing Algorithm. In *Proceedings of the International Conference on Broadband and Wireless Computing, Communication and Applications*; Springer International Publishing: Cham, Switzerland, 2019; Volume 25, pp. 737–750. https://doi.org/10.1007/978-3-030-02613-4.
42. Aleisa, M.A.; Abuhussein, A.; Alsubaei, F.S.; Sheldon, F.T. Examining the Performance of Fog-Aided, Cloud-Centered IoT in a Real-World Environment. *Sensors* **2021**, *21*, 6950. https://doi.org/10.3390/s21216950.
43. Adnan, M.; Iqbal, J.; Waheed, A.; Amin, N.U.; Zareei, M.; Umer, A.; Mohamed, E.M. Towards the Design of Efficient and Secure Architecture for Software-Defined Vehicular Networks. *Sensors* **2021**, *21*, 3902. https://doi.org/10.3390/s21113902.
44. Singh, J.; Bagga, S.; Kaur, R. Software-based Prediction of Liver Disease with Feature Selection and Classification Techniques. *Procedia Comput. Sci.* **2020**, *167*, 1970–1980. https://doi.org/10.1016/j.procs.2020.03.226.
45. Fröhlich, P.; Gelenbe, E.; Fiołka, J.; Chęciński, J.; Nowak, M.; Filus, Z. Smart SDN Management of Fog Services to Optimize QoS and Energy. *Sensors* **2021**, *21*, 3105. https://doi.org/10.3390/s21093105.
46. Llorens-Carrodeguas, A.; Leyva-Pupo, I.; Cervelló-Pastor, C.; Piñeiro, L.; Siddiqui, S. An SDN-Based Solution for Horizontal Auto-Scaling and Load Balancing of Transparent VNF Clusters. *Sensors* **2021**, *21*, 8283. https://doi.org/10.3390/s21248283.
47. Albowarab, M.H.; Zakaria, N.A.; Zainal Abidin, Z. Directionally-Enhanced Binary Multi-Objective Particle Swarm Optimisation for Load Balancing in Software Defined Networks. *Sensors* **2021**, *21*, 3356. https://doi.org/10.3390/s21103356.
48. Sharma, S.; Saini, H. Efficient solution for load balancing in fog computing utilizing artificial bee colony. *Int. J. Ambient Comput. Intell.* **2019**, *10*, 60–77. https://doi.org/10.4018/IJACI.2019100104.

49. Singh, P.; Kaur, A.; Aujla, G.S.; Batth, R.S.; Kanhere, S. DaaS: Dew Computing as a Service for Intelligent Intrusion Detection in Edge-of-Things Ecosystem. *IEEE Internet Things J.* **2021**, *8*, 12569–12577. https://doi.org/10.1109/JIOT.2020.3029248.
50. Tuli, S.; Poojara, S.R.; Srirama, S.N.; Casale, G.; Jennings, N.R. COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 101–116. https://doi.org/10.1109/TPDS.2021.3087349.
51. Shen, S.; Van Beek, V.; Iosup, A. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 4–7 May 2015; pp. 465–474. https://doi.org/10.1109/CCGrid.2015.60.
52. Tuli, S.; Basumatary, N.; Gill, S.S.; Kahani, M.; Arya, R.C.; Wander, G.S.; Buyya, R. HealthFog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments. *Future Gener. Comput. Syst.* **2020**, *104*, 187–200. https://doi.org/10.1016/j.future.2019.10.043.
53. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 military communications and information systems conference (MilCIS), Canberra, ACT, Australia, 10–12 November 2015; pp. 1–6.
54. Wang, Y.; Meng, W.; Li, W.; Liu, Z.; Liu, Y.; Xue, H. Adaptive machine learning-based alarm reduction via edge computing for distributed intrusion detection systems. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e5101.