Library Progress International
Vol.44 No. 3, Jul-Dec 2024: P. 18473-18485

Print version ISSN 0970 1052
Online version ISSN 2320 317X

Original Article

Available online at www.bpasjournals.com

Design and Implementation of an Enhanced Load Balancing Algorithm in Fog Computing

Roa'a Mohammed Mahdi[1], Hassan Jaleel Hassan[2], Ghaidaa Muttasher Abdulsaheb[3]

[1,2,3]Department Of Computer Engineering, University of Technology, Baghdad, Iraq
21.04@grad.uotechnology.edu.iq , hassan.j.hassan@uotechnology.edu.iq

How to cite this article: Roa'a Mohammed Mahdi,Hassan Jaleel Hassan, Ghaidaa Muttasher Abdulsaheb (2024) Design and Implementation of an Enhanced Load Balancing Algorithm in Fog Computing**.** *Library Progress International,* 44(3), 18473-18485.

ABSTRACT:

This study aims to improve load balancing in fog computing networks, which is especially crucial for latency-sensitive IoT applications. An innovative algorithm utilizing Q-learning was designed to optimize load distribution. By incorporating Q-learning agents within fog nodes, the system monitors workloads and resource availability in real-time, enabling adaptive and efficient load balancing decisions. The proposed solution was evaluated using the OMNeT++ simulation framework, yielding significant performance enhancements: a 37% reduction in packet drop rates, a 42% decrease in response times, and a 28% increase in throughput compared to conventional methods.

Keywords: load balancing, fog computing, Q-learning, optimize, OMNET++

1. Introduction

The demands of emerging applications have led to the evolution of computing paradigms to deal with the increasing use of networked devices and the explosion of data in the digital era. Fog computing is the extension of the cloud computing paradigm to the edge of the network, focusing on providing horizontal, system-wide, low-latency, and hence latency-sensitive, context-rich, and data-demanding services in domains such as the Internet of Things (IoT), smart cities, and hyperphysical systems [1]. Fog networks, the underlying abstraction layer of the fog computing infrastructure, play a significant role in managing the distributed processing, storage, and communication of data among various devices and platforms [2]. Such networks facilitate data exchange among devices, including sensors and actuators, smartphones, and servers, enabling task offloading and resource sharing [3-4].Fog enables enterprises to distribute computing, storage, resources, and services closer to user clinging in local infrastructure, facilitating real-time data operation, operation efficiency, and user experience to deploy edge-centric applications [5]. Fog networks are a collection of fog nodes comprising different computing resources with varied capabilities and capacities [6]. These entities are placed at the edge or close to the user, IoT device, or data sources, enabling low latencies and consuming less bandwidth for quick response times. By designing the fog nodes in the cluster way, administrators can provide fault tolerance, load balancing, and resource allocations to the fog node for high availability, scalability, and reliability of a fog computing infrastructure [7-8]. Data packets in fog networks are forwarded across different layers in the network stack and experience different processing and transformation operations while they are processed from source to destination. Network protocols and the set of rules and algorithms for packet routing, forwarding, and network delivery control the transmission of data packets. Fog nodes must serve as intermediaries in packet processing to inspect and filter packets and then aggregate the packets to reduce the load on the backhaul link and ensure low transmission delay [9]. Even though fog networks are resilient, possible dropouts and throughput fluctuations could still happen due to network congestion, channel interference, hardware failure, or software bugs [8]. These problems affect the operation and dependability of networked programs. Congestion control

algorithms, error recovery mechanisms, and Quality of Service (QoS) policies have been proposed in fog networks to reduce dropouts and improve the overall throughput and service reliability of real-time and safety-critical network applications [10].

Resource utilization optimization and data processing efficiency enhancement in cloud-fog networks are only possible if load balancing is implemented. Centralized cloud environments have used traditional load-balancing methods like Round Robin (RR), Least Connections (LC), and Weighted Round Robin (WRR). However, the dynamic and decentralized nature of fog environments demands load balancing mechanisms that can adaptively perform the distribution of workloads based on real-time data [11]. Newer systems are incorporating machine learning-based algorithms like Support Vector Machines (SVMs), Decision Trees (DTs), K-means clustering, and Self-Organizing Maps (SOM) for workload pattern prediction and load-balancing decisions [12]. For adaptive load balancing in fog networks, it is shown that a Reinforcement Learning (RL) approach, namely Q-learning, is promising [13-14]. Studies by Wul et al. [15], Xu et al. [16], and Gupta et al. [17] Resource allocation can optimize load balancing, and this research uses load balancing with the support of RL-based approaches. This paper utilizes RL-based load balancing with optimized resource allocation. Regarding the research gap, although previous work has shown the promise of RL-based load balancing, there is a lack of work that examines and implements these techniques in a fog computing environment. This work tries to fill this gap by introducing a Q-learning-based load balance algorithm and evaluating its enhanced performance for fog networks.

Fig. 1 illustrates three-level cluster components of the fog computing network: central databases/brokers, servers/routers, and smart devices/users. This represents the interconnection and data flow between clouds, networks, and end devices. A hierarchical topological diagram shows how cloud and fog nodes cooperate in the fog ecosystem.

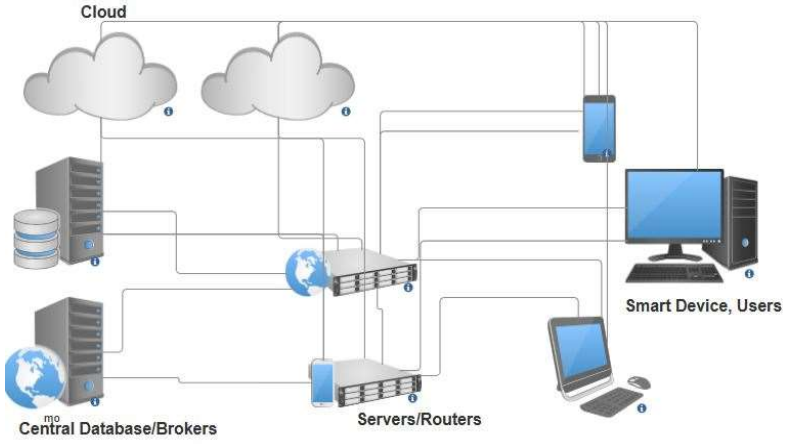


Fig. 1 Fog network with three-level cluster components

In this paper, an improved Q-learning-based load balancing algorithm has been proposed for fog node load balancing to update the policy for load distribution. Q-learning agents monitor live workload and resource availability and take actions to achieve the best load balance possible. This work evaluates the proposed algorithm using the FogNetSim library, implemented using the OMNeT++ simulation framework. Beautiful simulation results indicate that the proposed Q-learning load balancing algorithm can significantly lower the drop rate and response times and increase the throughput when no Q-learning is involved.

The rest of the paper is organized as follows: Section 2 gives an overview and survey on load balancing techniques in fog computing. The preliminaries and system model are described in Section 3. The proposed Q-learning load balancing approach is described in Section 4. The setup and the parameters of this simulation are detailed in Section 5. The experimental results are introduced and analyzed in Section 6. The last Section, 7, gives a conclusion and possible future research directions.

2. Literature Review

The cloud-fog network load balancing is the key to improving resource use and data processing efficiency. This section briefly surveys the load-balancing techniques and related work on load-balancing over fog cloud architecture. In a traditional sense, the load balancer's goal has been to balance the load of work being put onto the computing resources at the back end to keep response times low and throughput high. Some algorithms for the centralized cloud environment are Round Robin (RR), Least Connections (LC), and Weighted Round Robin (WRR). For instance, RR distributes incoming requests in a circular order, LC distributes the new request on that server with a minimum number of active connections, and WRR assigns a weight mark on incoming servers according to capacity. The dynamic and distributed nature of fog environments necessitates implementing dynamic load-balancing approaches. They consider real-time information such as the availability of resources, the status of the network, and the demands of applications to decide how to distribute workloads dynamically. Predictive is where predictive algorithms anticipate future workload trends and provision for the workload as needed for that prediction, and Reactive is where resources are reactively adjusted when workloads change in real-time: load, demand, and resource availability. The literature has also considered using machine learning-based algorithms to enhance the load balancing of fog networks [18]. A subsequent section focuses on the ability to predict workload patterns using supervised learning algorithms and subsequently make load-balancing decisions. For example, any variant of SVM or DTs [19]. Further, similar workloads can be clustered together, and resources can be allocated dynamically using unsupervised learning methods such as K-means Clustering and Self-Organizing Maps (SOM) [20].

To the best of the researchers' knowledge, RL has been proposed as a candidate method to balance the load requested by fog networks adaptively. As a result, by using different RL algorithms, fog nodes can act and learn interactively with the environment and make decisions that let them distribute the load in the best way possible.

Fog computing Q-learning, a popular RL algorithm, has been utilized in fog computing to learn optimal load balancing; like-wise, reward and punishment are given for each action [21]. Many studies have considered the load balance techniques particularly suitable for fog computing environments, according to the study of Wul et al. [15]The fuzzy logic-based load balancing proposed is used for dynamic resource allocation in fog networks. In another study by Talaat et al., the fuzzy rule-based mechanism dynamically changes resource allocation according to workload properties and network circumstances [22], the authors present a hierarchical load-balancing scheme in a fog computing environment. This scheme appropriately distributes the fog nodes in a hierarchy and offloads all the balancing decisions directly from edge to fog controller. Such a hierarchical design facilitated effective workload replication with minimal communication overhead. Other research by Gupta et al. has studied game theory for load balancing in fog networks [17]. This work considers fog nodes like players in a non-cooperative game and suggests that maintaining load equilibrium among adversaries is possible using the Nash Equilibrium theory. Combining the individual and collective utility functions allows this algorithm to incentivize each fog node to make choices based on their rationality, resulting in a stable load distribution. In addition to the studies mentioned above, recent works from Firouzi et al. utilize the machine learning-based load balancing technique in fog computing foundations [23]. These studies showed that the reviewed methods prove that RL could efficiently redistribute resources in real time and improve system performance. Moreover, Liang et al. [24] present a genetic algorithm-based load-balancing strategy formulated for fog computing with task allocation and resource utilization, where the results show such a process is essential in a highly dynamic and heterogeneous fog environment.

Talaat et al. [25] propose a new Deadlock Management System (DMS) to improve load balancing in fog computing environments. A DMS comprises five modules working together, namely a receiver, matchmaker, deadlock predictor, dispatcher, and a ranking module, allowing the system to perform the prioritization of jobs, the calculation of optimal server match, the prediction of deadlocks, and the efficient dispatching of jobs. Thus, reduces the burden on the network and latency to computation and improves the load distribution. Results demonstrate that this technique outperforms the existing method regarding degradation in resource provisioning and fog computing.

Abdulazeez and Askar. proposed in [14] an enhanced offloading strategy for IoT applications. For certain DQN-based fog-cloud systems, fuzzy logic with reduced task scheduling and load balancing has been implemented to avoid latency, power, and network usage overhead, enhancing their performance.

Shruthi. et al. [26] propose a new approach for load balancing in fog computing environments. Recently, the Deep Residual Network (DRN) has been applied to predict resource consumption, while the Mutated Leader Algorithm (MLA) is designed for an optimal allocation of tasks. It redistributes tasks from overloaded Virtual Machines (VMs) to underloaded VMs to balance loads, resulting in lower performance, such as the centralized execution time, cost, and load balance metrics of different metrics for different scheduling algorithms. Experimental results have shown that MLA provides minimal execution time, lower cost, and a balanced load compared to conventional methods.

Malik et al. [27] present an intelligent load-balancing framework for fog-enabled communication in healthcare, a fog-based framework for improving healthcare communication systems. The system uses Artificial Intelligence (AI) integrated into fog computing to direct and manage the large data flow from smart medical sensors. The framework is oriented towards low latency, inexpensive, and real-time responsiveness. This consists of grouping virtual machines to allocate work and the load more efficiently. The results show better service quality, less response time, and applicability available for critical healthcare applications, mainly in a post-COVID-19 context.

Here, M. Ebrahim and A. Hafid, in their paper [28], recommend using the ELECTRE method of load balancing in fog computing. ELECTRE (an outranking Multi-Criteria Decision Analysis (MCDA)) algorithm for optimal workload distribution to individual workers, considering multiple criteria such as compute and network load. The result of this strategy is to increase resource utilization and reduce time-to-completion. The simulation results show that the ELECTRE-based method works with 67% better performance concerning well-established mechanisms like random algorithms, Round-Robin, and Nearest node selection.

3. Preliminaries

OMNeT++ is a discrete-event simulation library used to model and build complicated systems. Its fundamental principle is to become the central focus for agent-based simulation. It uses a modular and extensible architecture, enabling users to implement custom-built simulation models and scenarios. The behavior and interactions of the various wireless network components will be integrated into OM-NeT++, for subsequent simulation. In this paper, OMNeT++, will be used to simulate the proposed wireless model. INET Framework is an OMNeT++ extension for modeling and simulation of communication networks. It comprises a complete set of predesigned modules, protocols, and models based on many network technologies, including wired and wireless networks. The INET will be used in the proposed simulation to define wireless network components, as shown in Fig. 2, which illustrates a test simulation network using the Fognetsim INET Framework in OMNeT++. The Figure comprises routers, compute brokers, access points, and users. This configuration is used to model fog computing environments and shows the interactions and communication paths between different nodes in the network. This diagram illustrates the fog network, showing the organized layout and functional relationships. FogNetSim library includes more modules and features designed specifically for fog and edge computing simulation. FogAT supports the abovementioned technologies regarding the INET framework and extends its capabilities to implement fog computing architectures, edge devices, and communication protocols. Importing the FogNetSim library within the simulation allows for the simulation of fog computing use cases and the analysis of the results of algorithms and techniques related to fog. The Network Element Driver (NED) file defines the network topology, components, and parameters in OMNeT++. Moreover, it provides hierarchically structured information regarding network elements and describes their characteristics. In this work, the NED file will contain declarations to create fog nodes, access points, routers, wireless modules/channels, and any other network components for various parts of the wireless network topology as part of the setup.

The configurator module is activated at the beginning of a simulation to define and set up the network topology and parameters. This establishes the condition and relation of each network node, assigns IP addresses, and allows channels to different machines, among other peripheral things. The

radio medium module functions as the communication medium for wireless buses and involves signal propagation of an electromagnetic source between two nodes. This model simulates the effects of attenuation, interference, and fading on wireless transmissions.

This approach allows for investigating communication protocol performance when combined with the proposed algorithms, enabling their evaluation within a suitable approximation of a real-world transmission setting. The main simulation parameters are presented in Table 1. In contrast, the lifecycle controller tracks and maintains network nodes and components alive in the simulations. It also handles node initialization, startup, shutdown tasks, and lifecycle events, guaranteeing everything is neatly called and synchronized. In addition, the lifecycle controller module and the response from network nodes are simulated under various operational conditions. These components and modules combine with the OMNeT++ simulation framework to build a wireless network system with complete fog and edge computing behaviors. Using the computing system, this setup will be used as the benchmark to implement, experiment, and optimize the Q-learning algorithm for bool resource management, routing, and task scheduling before and after fog enablement.

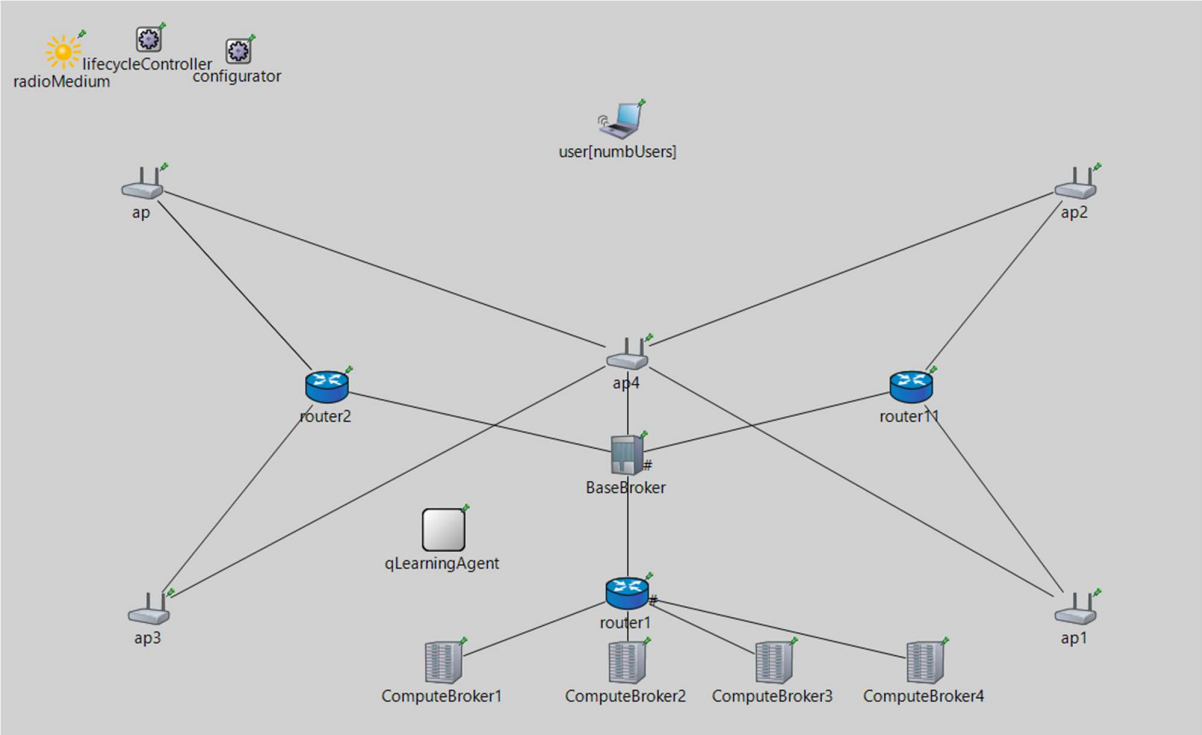


Fig. 2 Test Simulation Network, in Fognetsim INET Framework, OMNeT++

Table 1 Simulation parameters

Routing Protocol Type	ECRP CBPR LEACH AODV DS DV DSR and OSLR
Simulation Time (sec)	900
Simulation Area (m)	1500*1500
Traffic Pattern	CBR
Frequency	2.4 GHz
Mobility Model	Random any point
Transmission Range (m)	300
Peed of Node (m/)	2,5,8,10,14
Transport Layer	UDP
CBR packet size (Byte)	64
Mac Type	802.11b

Packet size (Byte)	64
Antenna Type	Omni directional Antenna
Simulator	OMNeT++

4. Proposed Load Balancing

This section introduces the proposed Q-learning-based approach for fog network load balancing problems. It is an RL method to implement to learn procedures from experiences to adapt the load-balancing policies of fog nodes autonomously. Integrating Q-learning agents into fog nodes enables dynamic and adaptive load balancing, which can help maximize resource usage, ensuring minimal response times. The technique mentioned above runs Q-learning agents on fog nodes periodically to monitor the workload and resources.

In many works, the agents are designed to learn Q-learning-based algorithms and decide on load balancing while considering the current workload, available information on resources, and network scenarios. Agents learn and explore through multiple iterations to optimize load distribution per fog node for better resource utilization and a more effective performing system.

Load balancing in wireless networks refers to distributing network traffic and computational work across various nodes in the network. This helps enhance resource usage, reduce congestion, and improve network performance. For instance, the Q-learning reinforcement learning algorithm can be utilized to realize load balancing. The network nodes integrate their decisions on who should do what and where based on experience through training according to previous reward signals.

4.1. Q-Learning agent implementation

The Q-learning algorithm is embedded in a class, the Q-learning agent, which is derived from cSimpleModule of OMNeT++, as shown below. This class encapsulates the generic Q-learning agent's behavior, which includes initialization, message handling, selection of a policy call to update the Q-value, and a method for printing Q-values.

4.2. Parameters of Q-Learning

Following are the parameters used in simulating Q-Learning

(1) Epsilon (ε): Exploration rate is how likely the agent is to try new actions rather than choose its best-known ones. Bit balancing is being able to explore (find new strategies) and exploit (use the known strategy). Higher epsilon will encourage exploration.

(2) Alpha (α): Learning rate, which controls the impact of new information on updating Q-values. It determines how quickly the agent adapts its Q-values based on observed rewards. A higher alpha lead to faster learning but may result in less stability.

(3) Gamma (γ): The discount factor signifies the significance of future rewards when the agent makes decisions. It affects how much the agent values long-term versus immediate rewards. The higher the gamma values, the more future rewards are considered.

(4) Number of States (numStates): Represents the total number of states in the environment. In a wireless network, states could correspond to different network conditions or configurations.

(5) Number of Actions (numActions): Indicates the total number of actions available to the agent in each state. In load balancing, actions might include routing traffic to different nodes or adjusting transmission power levels.

4.3. Q-learning workflow

(1) Initialization: The Q-learning agent initializes its Q-values table, setting all Q-values to zero or random values.

(2) Policy Selection: In each state, the agent selects an action based on the epsilon-greedy policy,

balancing exploration and exploitation. With probability ε, it chooses a random action; otherwise, it determines the action with the highest Q-value for that state.

(3) Action Execution: The selected action is executed in the environment, resulting in a new state and a reward.

(4) Q-Value Update: The agent updates its Q-values based on the observed reward and the transition to the next state. Based on Eq. (1) representing the update equations, the reward received, and maximum Q-value in the next state are multiplied with the learning parameter alpha before adding to the existing Q(s,a) value.

$$Q(S,A) \leftarrow (1-a) \times Q(S,A) + a^{*}\left(R + y^{*} \max Q\right) \quad (1)$$

The equation further modifies the Q-value for a state-action pair, making it dependent on the reward observed (R), the maximum Q-value(maxQ) of the next state, and the learning rate (α) and discount factor(γ). The epsilon-greedy policy entails choosing a random action with the likelihood of ε or selecting the action with the highest Q-value minus ε. For convergence, while Q-learning proceeds through several episodes, the algorithm converges over time to an increasingly optimized policy for receiving the best cumulative rewards.

4.4. Role of parameters in Q-learning

(1) Epsilon: Give the balance between exploration and exploitation. Increasing ε will also lead to more exploration, which can help find better policies but at the cost of slower convergence.

(2) Alpha: This depends on how frequently the agent updates over its previous Q-value based on rewards encountered in a new state. Larger alpha values will result in more aggressive updates and, in turn, may help to converge faster, but they could be unstable.

(3) Gamma: Gamma affects how much the agent considers future rewards. Higher Gamma values long-term rewards more, making the agent prioritize getting a high cumulative reward over time.

(4) The number of states and Actions: Define the agent's decision level. This gives rise to a finer-grained control, and with more states and actions, can develop even more sophisticated strategies; however, the computational cost increases. Q-learning thus provides a flexible and adaptive way to perform load balancing in wireless networks, where nodes can learn how to adapt their behavior over time according to the learned reward received and experiences of past external stimuli. With appropriate parameter tuning and the introduction of domain-specific information to guide action selection,

Q-learning can solve load-balancing problems and enhance network performance in time-varying and diverse setups. Algorithm 1 illustrates the steps of the proposed fog network load balance-based Q-learning steps.

Algorithm 1

Fog Network Load Balance Q-Learning
1: Initialization: Q(s,a)←random or zero∀(s,a).
2: Set Hyperparameters:
3: Input→ε
4: Input→α
5: Input→γ
6: Input→ (numStates)
7: Input→ (numActions)
8: For each Episode:
9: Initialize State S randomly or using a predefined method
10: Repeat until Termination:
11: i. Choose Action using epsilon-greedy policy:
12: →With probability epsilon, select random action
13: →Otherwise, select the action with the highest Q-value for state S

14: ii. Execute Action A:
15: →Observe Reward RandnewStateS '
16: iii. Update Q-value for state-action pair(S,A):
17: Calculate the maximum Q-value for state S^ (maxQ)
18: Update Q-value using:
19: Q(S, A) ← (1-α)*Q(S, A) +α*(R+γ*maxQ)
20: iv. Update State S to S^(maxQ)
21: Repeat for a fixed number of episodes or until convergence
22: criteria are met
23: After Learning:
24: - Q-table contains optimal Q-values for each state-action pair
25: Real-World Actions:
26: - Choose actions based on learned Q-values
27: - Update Q-values in real-time based on observed rewards and state
28: transitions
29: Optional:
30: - Fine-tune hyperparameters and Q-learning algorithm based on
31: performance evaluation and feedback

5. Simulation Setup

Exploration vs. Exploitation: A higher epsilon value encourages more exploration, which can help the agent discover optimal policies. However, too high epsilon may lead to inefficient exploration, while a low epsilon may result in premature convergence to suboptimal policies. Start with a relatively high epsilon value to encourage initial exploration. As learning progresses, it gradually decreases epsilon to shift towards exploitation. Typical values for epsilon range from 0.1 to 0.5. Alpha determines the weight given to new information when updating Q-values. Higher alpha will make learning faster, too high, and it becomes unstable learning with oscillations, which keeps the fast learning while maintaining stability. An alpha that is too high can result in the agent overfitting to noisy rewards, while an alpha that's too low may not allow for learning fast enough. Typically, alpha will vary between 0.1 to around 0.5 using common implementations of Gamma. This is the discount rate. It reduces the contribution of future rewards: How much are future rewards compared to immediate rewards? The higher value of Gamma tries to maximize the long-term return (which means it cares more about distant rewards). Lower values make the agent consider a more extended sequence of events by putting more weight on immediate reward.

Gamma should be chosen depending on the time span of the problem. Lower Gamma can be used for short-term tasks (precisely one step of planning into the future). Gamma varies typically from 0.9 to 0.99 when considering the range of gamma values. It can be seen that the number of states is the granularity in understanding how much the agent knows the environment. More number states mean learning at a finer level but increases the complexity as one would have more combinations to work out. The balance between complexity and performance: The number of states is chosen based on the complexity of the problem; on one side, it is better to have as many parameters as possible to represent the environment, but at the same time, that requires the need to take into considerations available computational resources. Figs. 3 and 4 show the simulation and the simulation parameters.

Typical Range: The number of states for a problem can vary widely from tens to thousands, depending on the application.

Dimension of action space: It defines the number of dimensions. Increasing the action space is a valid approach, but it is important to note that each new path may require additional exploration effort. Ultimately, the best values for these parameters often must be found by experimenting and tuning over many iterations within any given environment or application-specific context. Modifying these parameters to suit empirical results and domain knowledge can help enhance performance and converge it for the Q-learning algorithm. Fig. 5 shows the results of the drop rate for both cases without and with Q-learning, while Fig. 6 illustrates the response time for both cases, i.e., without and with Q-learning, respectively. Finally, Fig.7 depicts the network throughput for the same two cases. The results

confirm the role of Q-learning in enhancing the network's performance by improving the drop rate, response time, and the network's throughput.

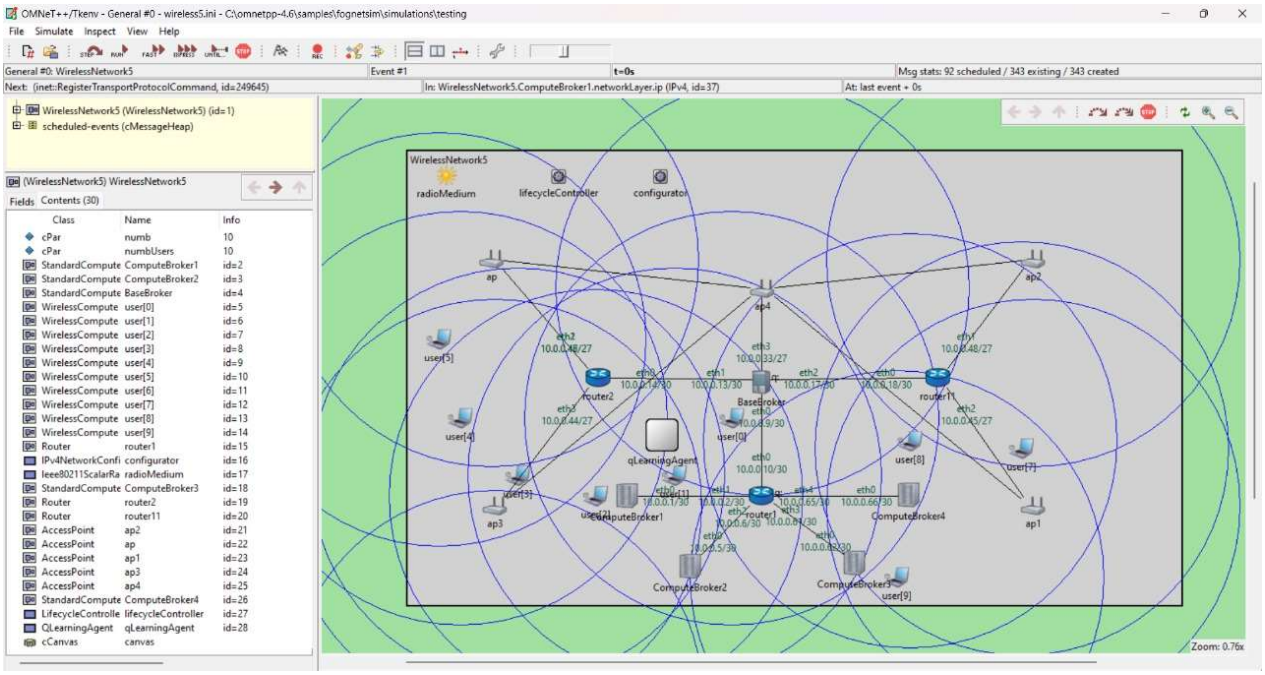
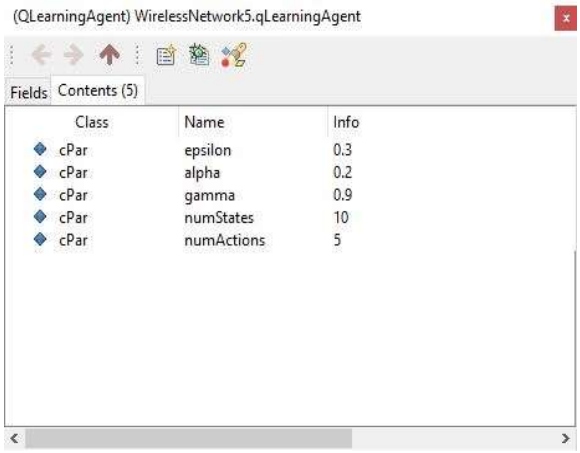
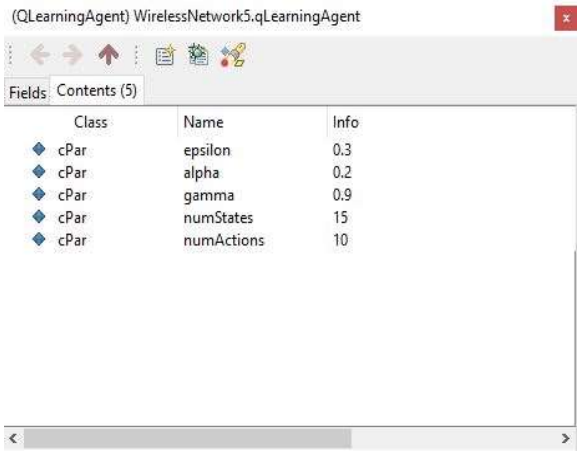


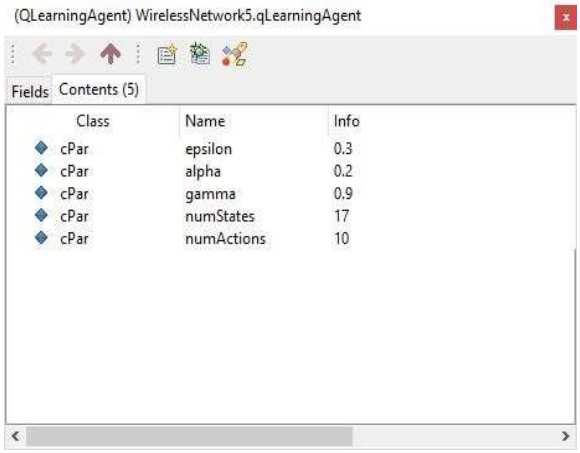
Fig. 3 Network with 21 smart devices (users) at simulation with Q-Learning Agent Module



(a) With number of state equal 10

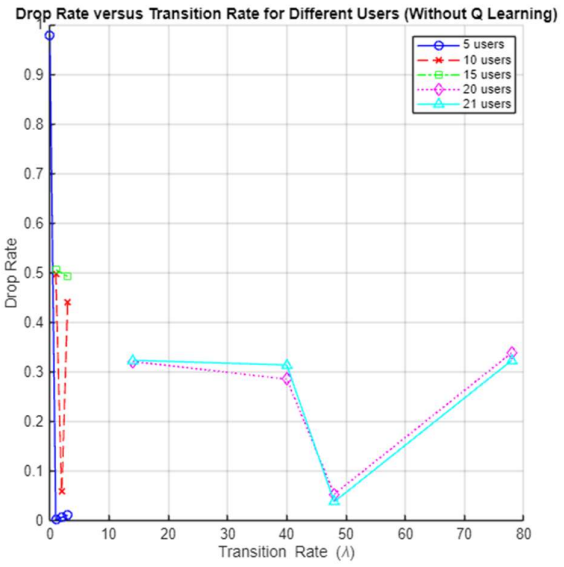


(b) With number of state equal 15

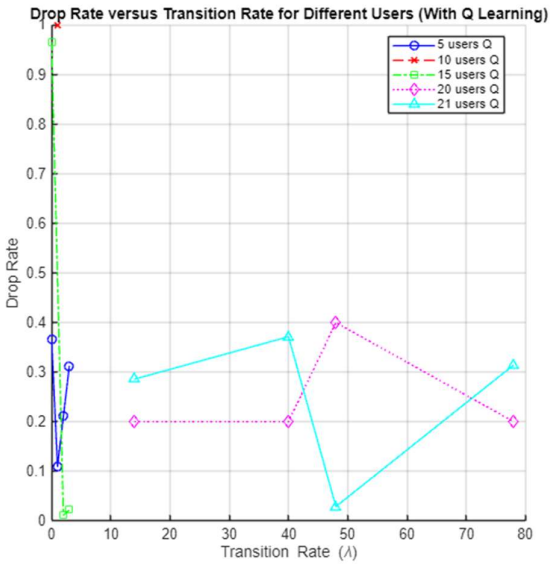


(c) With number of state equal 17

Fig. 4 Different Q-learning parameters cPar values for networks with varying user

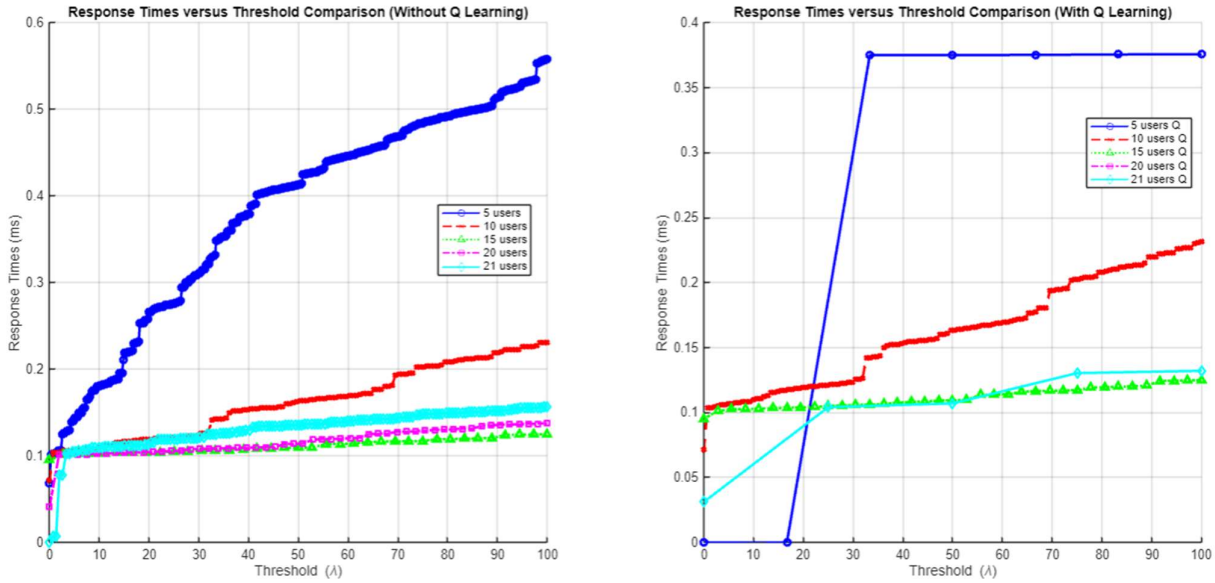


(a) Without Q-Learning



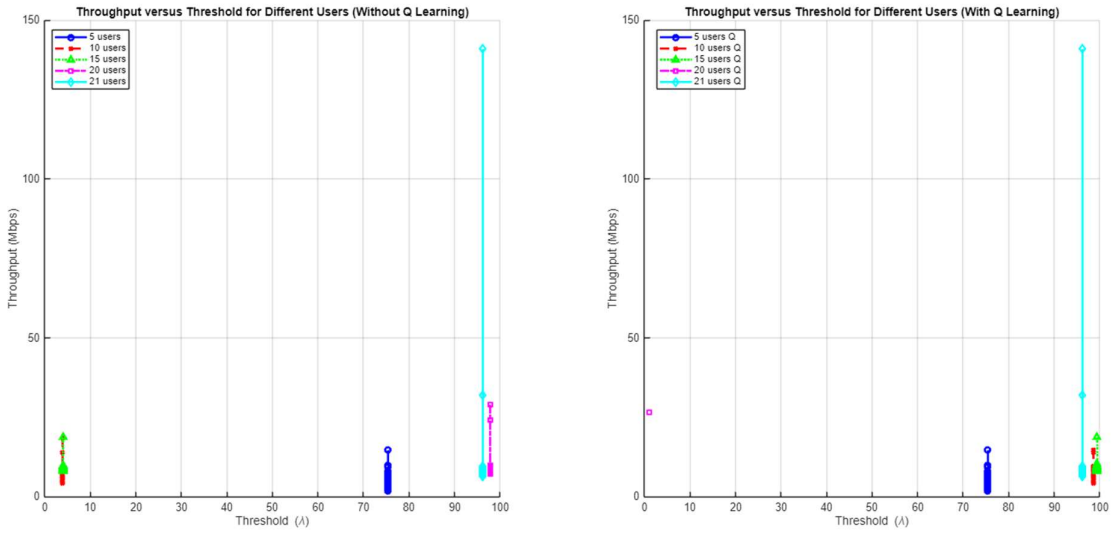
(b) With Q-Learning

Fig. 5 Drop Rates results for with and without using Q-Learning



(a) Without Q-learning (b) With Q-learning

Fig. 6 Response time results for with and without using Q-Learning



(a) Without Q-learning (b) With Q-learning

Fig. 7 Throughput results or with and without using Q-learning

6. Conclusion

In this paper, a Q-learning-based load-balancing algorithm is presented and designed for fog computing issues. Extensive simulations are conducted using OMNeT++ with FogNetSim library to analyze the performance of the proposed approach in comparison to classical load balancing solutions. Moreover, essential properties, like packet loss rates and response times, were examined, and network throughput was assessed. According to the results, Q-learning can enhance the operation of the network management system. Specifically:

1. Reduced packet Drops: Q-learning decreases the packets dropped, making the network more reliable and consistent. The results show a 37% reduction in packet drop rates compared to networks without Q-learning.

2. Response time: The overall response times are also greatly enhanced, while the part where Q-learning boosts quicker responses is due to how its network responds to real-time usage purposes.
3. Throughput: The throughput increases, improving network efficiency.

Q-Learning uses RL to allow nodes in the network to make independent, intelligent decisions, which helps improve resource allocation and routing efficiency. The Q-learning algorithms establish a definite measure by optimizing the resource allocation policy based on dynamic adjustment of the routing path, transmission power, and buffer management policies. As a result, there are fewer packet drops and, therefore, improved QoS. Moreover, Q-learning enables nodes to send the packet through a much more efficient route, aiming to minimize the queuing delay and transmission latency. More detailed study and optimization for scalability purposes and changes within the environment in wireless networks are required for the future development of Q-learning algorithms. It could also further improve performance and manageability to investigate integration with other emerging machine learning algorithms, e.g., deep reinforcement learning. This research is essential because it has shown that the Q-learning algorithm is a suitable method to organize the wireless network functions so that the network administration makes the right decision and adjusts to available actions more efficiently. Thus, it is better to distribute resources, reduce latencies, and, most importantly, balance the throughput to improve the network and user satisfaction.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] H. R. Abdulqadir, A. Ali, B. Ahmed, C. Khalid, D. Ehsan, and E. Hameed, "A Study of Moving from Cloud Computing to Fog Computing," Qubahan Academic Journal, vol. 1, no. 2, pp. 60-70, Augst 2021.
[2] A. Hazra, P. Rana, M. Adhikari, and T. Amgoth, "Fog Computing for Next-Generation Internet of Things: Fundamental, State-of-The-Art and Research Challenges," Computer Science Review, vol. 48, pp. 100549, February 2023.
[3] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in Fog Computing for IoT: Review, Enabling Technologies, And Research Opportunities," Future Generation Computer Systems, vol. 87, pp. 278-289, September 2018.
[4] S. Waheeda, A. Sakran, M. Rahim, N. Suaib, F. Najjar, K. Kadhim, A. Salim, and M. Adnan, "Design a Crime Detection System based Fog Computing and IoT," Malaysian Journal of Fundamental and Applied Sciences, vol. 19, no. 3, May 2023.
[5] B. P. Bhuyan, R. Tomar, and A. K. Yadav, "Edge Computing in Smart Agriculture," in Future Connected Technologies: 1st ed. Florida :CRC Press, 2023.
[6] M. Aazam, S. Zeadally, and K. A. Harras, "Fog Computing Architecture, Evaluation, and Future Research Directions," IEEE Communications Magazine, vol. 56, no. 5, pp. 46-52, November 2018.
[7] Q. D. La, M. V. Ngo, T. Q. Dinh, T. Q. Quek, and H. Shin, "Enabling Intelligence in Fog Computing to Achieve Energy and Latency Reduction," Digital Communications and Networks, vol. 5, no. 1, pp. 3-9, October 2019.
[8] R. M. Mahdi, H. J. Hassan, and G. M. Abdulsaheb, "A Review Load Balancing Algorithms in Fog Computing," in BIO Web of Conferences, EDP Sciences, pp.33-36, March 2024.
[9] I. Ahammad, M. A. R. Khan, and Z. U. Salehin, "Software-Defined Dew, Roof, Fog and Cloud (SD-DRFC) Framework for IoT Ecosystem: The Journey, Novel Framework Architecture, Simulation, and Use Cases," SN Computer Science, vol. 2, no. 3, p. 159, October 2021.
[10] N. H. Hussein, C. T. Yaw, S. P. Koh, S. K. Tiong, and K. H. Chong, "A Comprehensive Survey on Vehicular Networking: Communications, Applications, Challenges, and Upcoming Research Directions," IEEE Access, vol. 10, pp. 86127-86180, January 2022.
[11] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras, "Decentralized Edge-To-Cloud Load Balancing: Service Placement for The Internet of Things," IEEE Access, vol. 9, pp. 64983-65000, January 2021.
[12] D. Soni and N. Kumar, "Machine Learning Techniques in Emerging Cloud Computing Integrated

Paradigms: A survey and Taxonomy," Journal of Network and Computer Applications, vol. 205, p. 103419, July 2022.

[13] H. T. Dang, S. Bhardwaj, T. Rahim, A. Musaddiq, and D. S. Kim, "Reinforcement Learning Based Resource Management for Fog Computing Environment: Literature Review, Challenges, and Open Issues," Journal of Communications and Networks, vol. 24, no. 1, pp. 83-98, March 2022.

[14] D. H. Abdulazeez and S. K. Askar, "Offloading Mechanisms Based on Reinforcement Learning and Deep Learning Algorithms in the Fog Computing Environment," IEEE Access, vol. 11, pp. 12555-12586, August 2023.

[15] D. Wu, J. Li, A. Ferini, Y. Xu, M. Jenkin, S. Jang, X. Liu, and G. Dudek, "Reinforcement Learning For Communication Load Balancing: Approaches and Challenges," Frontiers in Computer Science, vol. 5, pp.256-289, January 2023.

[16] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Load Balancing for Ultradense Networks: A Deep Reinforcement Learning-Based Approach," IEEE Internet of Things Journal, vol. 6, no. 6, pp. 9399-9412, May 2019.

[17] M. Gupta, S. Chinchali, P. Varkey, and J. G. Andrews, "Forecaster-Aided User Association and Load Balancing in Multi-Band Mobile Networks," IEEE Transactions on Wireless Communications, October 2023.

[18] F. M. Talaat, M. S. Saraya, A. I. Saleh, H. A. Ali, and S. H. Ali, "A Load Balancing and Optimization Strategy (LBOS) Using Reinforcement Learning in Fog Computing Environment," Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 11, pp. 4951-4966, September 2020.

[19] C. V. Oikawa, V. Freitas, M. Castro, and L. L. Pilla, "Adaptive Load Balancing Based on Machine Learning for Iterative Parallel Applications," in 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE Press, pp. 94-101, December 2020.

[20] M. Kulkarni, P. Deshpande, S. Nalbalwar, and A. Nandgaonkar, "Cloud Computing Based Workload Prediction Using Cluster Machine Learning Approach," in International Conference on Computing in Engineering & Technology, Springer, pp. 591-601, February 2020.

[21] M. A. Amin, H. B. Zubir, M. S. Khalid, A. Ahmad, and M. H. Arshad, "Renewable Energy Maximization for Pelagic Islands Network of Microgrids Through Battery Swapping Using Deep Reinforcement Learning," IEEE Access, August 2023.

[22] F. M. Talaat, S. H. Ali, A. I. Saleh, and H. A. Ali, "Effective Load Balancing Strategy (ELBS) For Real-Time Fog Computing Environment Using Fuzzy and Probabilistic Neural Networks," Journal of Network and Systems Management, vol. 27, pp. 883-929, October 2019.

[23] F. Firouzi, K. Chakrabarty, S. Uysal, K. L. Lee, M. Song, and A. Y. Zomaya, "Fusion of Iot, AI, Edge–Fog–Cloud, and Blockchain: Challenges, Solutions, and A Case Study in Healthcare and Medicine," IEEE Internet of Things Journal, vol. 10, no. 5, pp. 3686-3705, July 2022.

[24] J. Liang, Y. Long, Y. Mei, T. Wang, and Q. Jin, "A Distributed Intelligent Hungarian Algorithm for Workload Balance in Sensor-Cloud Systems Based on Urban Fog Computing," IEEE Access, vol. 7, pp. 77649-77658, June 2019.

[25] M. Talaat, A. Saleh, M. Moawad, and J. Zaki, "Fog Computing Effective Load Balancing and Strategy for Deadlock Prediction Management," Ain Shams Engineering Journal, vol. 14, no. 12, p. 102561, December 2023.

[26] G. Shruthi, M. R. Mundada, S. Supreeth, and B. Gardiner, "Deep Learning-Based Resource Prediction and Mutated Leader Algorithm Enabled Load Balancing in Fog Computing," International Journal of Computer Networks and Information Security, vol. 15, no. 4, pp. 84-95, August 2023.

[27] S. Malik, F. Ahmed, M. Shafiq, R. Ali, S. Anwar, and H. M. Yasin, "Intelligent Load-Balancing Framework For Fog-Enabled Communication in Healthcare," Electronics, vol. 11, no. 4, p. 566, February 2022.

[28] M. Ebrahim and A. Hafid, "Resilience and Load Balancing in Fog Networks: A Multi-Criteria Decision Analysis Approach," Microprocessors and Microsystems, vol. 101, p. 104893, September 2023.