Computing (2025) 107:159
https://doi.org/10.1007/s00607-025-01498-9

REGULAR PAPER

Efficient dynamic load balancing in software-defined networks using policy gradient: a strategy for enhanced QoS and reduced energy consumption

Mohammad Amin Zare Soltani[1] · Seyed Amin Hosseini Seno[1] · AmirHossein Mohajerzadeh[2]

Received: 15 February 2025 / Accepted: 14 May 2025

Abstract

In modern networking, software-defined networks (SDNs) have emerged as a powerful paradigm that separates the control plane from the data plane, enabling centralized and distributed network management. SDNs provide flexibility and efficiency in handling large-scale networks, aiming to optimize resource utilization, reduce energy consumption, and enhance quality of service (QoS). Given the rapid growth in data traffic and the increasing need to minimize response time and energy consumption, developing efficient load balancing strategies has become a critical challenge to ensure network performance and stability. Load balancing plays a vital role in optimizing data traffic distribution across servers and network nodes, preventing congestion, and improving system efficiency. This is especially crucial in large and complex environments such as cloud data centers and distributed networks, where handling high request volumes efficiently is essential. To address these challenges, this paper introduces SDN-PG, a novel dynamic load balancing strategy for SDNs that integrates policy gradient (PG), a reinforcement learning-based optimization method, with dynamic voltage and frequency scaling to enhance energy efficiency and network performance. SDN-PG dynamically optimizes traffic distribution by continuously adapting network policies to real-time fluctuations, significantly improving QoS while minimizing energy consumption. The proposed approach consists of three primary components. The first component is a distribution policy learned via Policy Gradient, enabling adaptive load balancing decisions. The second component involves real-time network monitoring, allowing the system to track and respond to dynamic traffic changes. The third component is an efficient decision-making mechanism, which leverages PG-based policies to reduce computational overhead and optimize response time. To validate its effectiveness, SDN-PG is compared against state-of-the-art methods, including CCA-PSO and DRL-SMS, through simulation experiments. The results demonstrate significant improvements in key performance metrics. SDN-PG achieves a 45.47% and 46.22% reduction in response time, a 14.09% and 11.98% decrease in computational overhead, a 19.47%

Extended author information available on the last page of the article

Published online: 28 June 2025

and 16.38% improvement in energy efficiency, and a 7.6% and 4.24% enhancement in load balancing effectiveness. These findings highlight the practical applicability of SDN-PG in large-scale SDN environments, demonstrating its ability to efficiently balance energy savings and QoS while maintaining optimal load distribution and network stability.

Keywords Software-defined network · Load balancing · Policy gradient · Reinforcement learning · Energy consumption · Dynamic voltage and frequency scaling

Mathematics Subject Classification 68W25 · 6806 · 68T01 · 68T05

1 Introduction

With the rapid growth of 4G, 5G, and 6G networks, along with the expansion of cloud infrastructures, data centers, and wide area networks (WANs), the volume of data traffic and network services is increasing at an unprecedented rate. Traditional network architectures are no longer capable of managing this significant influx of requests. Software-defined networks (SDNs) present an efficient solution for managing large-scale networks by decoupling the control plane from the data plane, enabling greater control over the network infrastructure. This decoupling significantly facilitates network management, enhances resource utilization, reduces capital and operational expenses, improves service quality, and increases network flexibility. In modern SDNs, load balancing has emerged as a critical challenge in optimizing network performance and ensuring the delivery of high-quality services. The objective of load balancing is to distribute incoming network flows across links, servers, and virtual machines (VM) in a way that ensures equitable utilization of all network resources, thereby maintaining operational efficiency and stability [1, 2].

1.1 Challenges

Despite the advantages of SDNs, several critical challenges remain unresolved. First, most existing load balancing techniques rely on centralized controller architectures, which introduce a Single Point of Failure (SPoF) and limited scalability, making them unsuitable for large-scale networks such as cloud computing and data centers [3, 4]. Second, high energy consumption in SDN environments has become a major concern, particularly due to the increasing complexity of traffic management and the demand for real-time services [5, 6]. Third, existing approaches often fail to simultaneously optimize multiple critical factors, such as energy efficiency, processing overhead, response time, and load balancing, making them suboptimal in dynamic and large-scale networks [7, 8]. Fourth, current load balancing strategies face difficulties in adapting to traffic fluctuations, which leads to inefficient resource allocation and network congestion [9, 10]. These challenges highlight the need for an adaptive, scalable, and energy-efficient load balancing strategy in SDNs.

1.2 Proposed solution

To address these challenges, this study proposes a novel dynamic load balancing strategy for SDNs, named SDN-PG, which integrates policy gradient (PG), a reinforcement learning algorithm, with dynamic voltage and frequency scaling (DVFS). Unlike traditional load balancing approaches, SDN-PG utilizes a distributed multi-controller architecture, eliminating the SPoF problem while enhancing scalability and network resilience. By dynamically adjusting server selection, resource allocation, and energy consumption, the proposed method effectively improves network efficiency while reducing latency and computational overhead.

The proposed approach consists of three key components:

- *Policy Distribution* Defines and updates load balancing rules based on real-time network conditions.
- *Monitoring Component* Continuously measures key network metrics, including energy consumption, processing overhead, response time, and load balance rate.
- *Decision-Making and Execution* Implements policy-based resource allocation, ensuring optimized load balancing across distributed SDN controllers.

1.3 Contribution

The main contributions of this research are as follows:

- Developing a multi-objective and energy-efficient load balancing solution by integrating reinforcement learning techniques in SDN environments.
- Introducing a distributed load balancing strategy to mitigate the SPoF issue and improve scalability and network flexibility.
- Leveraging the Policy Gradient method for intelligent selection of servers and virtual machines, ensuring adaptive and efficient traffic distribution.
- Applying the DVFS technique to minimize energy consumption while maintaining high service quality.
- Reducing response time and computational overhead, leading to better network performance and reduced waiting time for users.
- Improving load balancing and resource allocation efficiency by preventing network congestion and ensuring optimized workload distribution.
- Enhancing scalability and flexibility in large-scale SDN architectures.

Using the Omnet++ simulation tool, the proposed method is evaluated under various network conditions, and its performance is compared against the CCA-PSO [11] and DRL-SMS [12] algorithms. The simulation results indicate significant performance improvements across multiple key metrics.

The remainder of this paper is structured as follows: Sect. 2 presents a comprehensive review of the related literature. Section 3 details the proposed methodology and outlines the problem formulation. Section 4 discusses the simulation setup and

evaluates the performance of the proposed method, with comparisons drawn against the CCA-PSO and DRL-SMS techniques. Finally, Sect. 5 summarizes the key findings and provides concluding remarks.

2 Related work

At the beginning of this section, we will briefly discuss classifications of load balancing approaches in SDN, focusing on a Model-Free reinforcement learning algorithm known as the PG method. Subsequently, to examine the proposed method, prior works will be categorized into three groups and analyzed in detail.

Load balancing in networks refers to the effective distribution of data traffic across servers and various network nodes, aiming to reduce excessive load and prevent congestion. This concept is particularly critical in large and complex networks such as cloud networks and data centers, which handle a significant volume of requests [8, 9]. The different approaches to load balancing in SDNs are illustrated in Fig. 1 [3].

Centralized Controller Approach: It employs a centralized controller responsible for managing the allocation of traffic across the entire network.

Distributed Controllers Approach: In SDNs, the Distributed Controller Approach refers to a model where resource management and load balancing responsibilities are decentralized across various network nodes. In this framework, each node or local controller independently collects data regarding the load conditions of other nodes and makes local decisions based on this information. Unlike the centralized model, where decisions are concentrated at a single central point, this approach enables each node to function autonomously. These local decisions foster improved scalability and reduce reliance on a central authority, thereby enhancing the overall resilience and efficiency of the network.

Hybrid Controllers Approach: This approach integrates both centralized and distributed models, harnessing the advantages of each to deliver a holistic and efficient solution.

Static Load Balancing Approach: Static load balancing refers to a method in which network load and data transmission paths are pre-allocated and fixed. In this approach, communication routes between hosts are determined and assigned before data transfer begins, with no alterations made to these paths during the transmission process.

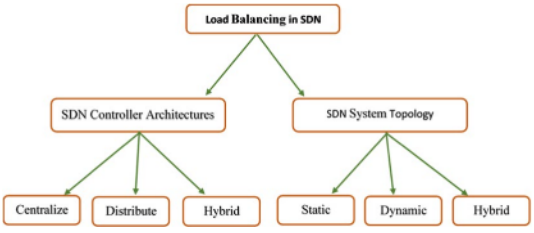

Fig. 1 Classifications of load balancing approaches [3]

Dynamic Load Balancing Approach: refers to a method in which the distribution of network load and resource allocation are continuously adjusted in real-time based on fluctuations in traffic patterns and network conditions. In this approach, the system autonomously redistributes the load across various nodes and resources, preventing congestion and ensuring the optimal performance of the network.

Hybrid Load Balancing Approach: refers to a method that combines the features of both static and dynamic methods to effectively manage load distribution and resource allocation within a network. This approach utilizes the advantages of each method to enhance network efficiency and optimize resource usage [3].

In modern SDNs, load balancing has emerged as one of the fundamental challenges in optimizing network performance and delivering high-quality services. With the growth of data traffic, the need for rapid response and efficient resource optimization has made the adoption of advanced methods for traffic management and load distribution indispensable [1, 2].

Given the existing research gap in this field, this study focuses on the analysis of dynamic load balancing in SDNs. To address this challenge, the research employs DVFS techniques to reduce energy consumption. Furthermore, for the first time, the PG algorithm a reinforcement learning method is utilized to select the optimal server for balanced load distribution.

To provide a comprehensive review of the existing literature, the related works are categorized into the following three areas:

- Load balancing in software-defined networks.
- Load balancing and energy optimization using DVFS in SDNs.
- Load balancing in SDNs using artificial intelligence and machine learning.

2.1 Load balancing in software-defined networks

When the number of network users or virtual network requests from the underlying infrastructure increases in the virtual network embedding problem, load distribution becomes a major issue that can ultimately affect network performance and quality of service. The use of SDNs can enhance network application availability, resource utilization, and QoS indicators such as delay, throughput, and response time [4]. The issue of load balancing in SDNs has not yet been comprehensively explored, and further research in this domain is required. The ultimate goal is to respond effectively to incoming traffic and determine the best transmission paths in the network while ensuring the highest possible QoS levels [4, 13].

Li et al. [4] proposed a multi-objective virtual network embedding (VNE) algorithm, addressing resource optimization by mapping virtual network requests onto physical infrastructure. They introduced a load-balancing approach that considers node load as an objective function while defining global resource capacity to evaluate infrastructure nodes' ability to handle embedding. The method employs a group search algorithm for iterative optimization. Simulation results indicate that the algorithm effectively balances network load, enhances the acceptance rate of virtual

network requests, and optimizes resource allocation, thereby increasing revenue for service providers and improving operational efficiency.

Chiang et al. [5] applied SDN to enhance flexibility in managing large scale networks, using server clustering with the OpenFlow protocol. They proposed the dynamic weighted random selection (DWRS) load balancing algorithm, which dynamically adjusts server weights based on real-time load, increasing resource efficiency. Multi-threading was used to optimize the Floodlight controller's performance by leveraging parallel processing, preventing overload. An experimental setup with real hardware demonstrated DWRS's superiority in load balancing compared to other methods, especially in heterogeneous environments.

Ahmadi and Movahedi [6] a distributed load balancing approach for SDN controllers is introduced, where each controller monitors its own load and gathers load data from other controllers when exceeding a set threshold. A switch with the least load is selected for migration to optimize network load balance. The method is fully distributed and ensures that load balancing is conducted only when specific conditions are met, enhancing efficiency. It mitigates single points of failure and significantly improves scalability, availability, and fault tolerance in the network.

Srivastava and Pandey [7] the authors reviewed various load balancing methods, including model based, estimation based, and nature inspired algorithms, providing a comprehensive categorization and comparison. Ethilu et al. [14] investigated an efficient method for switch migration in SDN to optimize load distribution among distributed controllers. The main objective is to reduce switch migration costs and controller resource consumption. The novelty lies in enhancing the time-shared switch migration (TSSM) method by selecting underloaded controllers, effectively minimizing migration costs while maintaining system benefits.

Gad-Elrab et al. [15] the authors propose an innovative approach for resource management and load balancing in fog-cloud environments using a fuzzy multi criteria decision making technique. The method involves four steps: determining criteria weights using fuzzy analytic hierarchy process (FAHP), ranking fog devices with fuzzy technique for order performance by similarity to ideal solution (FTOPSIS), calculating final device weights for task allocation, and assigning tasks to the optimal device. This approach aims to enhance system efficiency, reduce energy consumption, improve performance, and increase the flexibility of fog-cloud computing systems.

Banaie et al. [16] examines a multicriteria load-balancing scheme among gateways in fog-based IoT environments. Using a queuing model, the research analyzes data stream latency and gateway node congestion, which can negatively impact system reliability. The proposed model leverages a multicriteria decision-making (MCDM) approach for load balancing in IPv6 over low-power wireless personal area networks (6LoWPAN) networks, demonstrating a significant improvement in response speed and reliability when addressing user requests.

Jehad Ali et al. [17] proposed ESCALB (effective slave controller allocation-based load balancing) for SDN-based IoT networks, enhancing network efficiency through dynamic secondary controller assignment and optimized switch migration. Using analytic network process (ANP) for controller prioritization and the 0/1 knapsack algorithm for migration optimization, ESCALB improves resource utilization

and load balancing. Experimental results show a 35% reduction in energy consumption and 34% improvement in processing delay, surpassing conventional methods in QoS and communication cost efficiency. Despite its advantages, ESCALB's high computational complexity increases processing overhead, requiring advanced hardware. Future research could integrate machine learning to enhance scalability and performance in 5G and beyond networks.

Noda et al. [18] propose an optimized controller placement model for SDN to reduce migration downtime during controller relocation. Using mixed-integer second-order cone programming (MISOCP), the approach optimizes switch migration, cutting downtime by 80% and lowering operational costs by minimizing controller usage. It also improves network throughput by 30%, outperforming traditional methods. However, high computational complexity and controller coordination challenges remain, requiring further research to enhance scalability and performance.

The conclusion drawn from the research in Sect. 2.1 indicates that load balancing plays a critical role in enhancing network performance and quality of service in SDN and virtualized network environments. Various methods have been proposed to optimize resource allocation, reduce latency, and improve system scalability. However, challenges such as effective traffic management and the need for distributed solutions remain unresolved. These studies highlight that effective load balancing can significantly improve network performance, reliability, and scalability, emphasizing its importance in the optimization of modern network infrastructures.

A comprehensive comparison of existing load balancing strategies in SDNs, including centralized, distributed, and hybrid approaches, is summarized in Table 1. The table provides a structured comparison of different algorithms based on critical performance factors such as scalability, response time, and energy efficiency.

2.2 Load balancing and energy optimization using DVFS in SDNs

The dynamic voltage and frequency scaling (DVFS) algorithm is a sophisticated energy optimization technique widely used in contemporary computing systems to reduce power consumption. This method involves the dynamic adjustment of the processor's frequency and voltage based on the current workload. The core equation that governs power consumption in DVFS is expressed as follows:

$$P = \alpha f c v^2 \tag{2.1}$$

where P denotes the power consumption, c is a constant that depends on the specific architectural features of the processor, v represents the processor's voltage, f indicates the processor's frequency, α (scaling constant): This coefficient is determined by the processor's design and construction materials, reflecting the architectural factors that influence energy efficiency.

This equation illustrates that power consumption is highly sensitive to variations in both voltage and frequency. Specifically, since power consumption is proportional to the square of the voltage (v^2), reducing the voltage can lead to a substantial decrease in energy usage without compromising the performance of the system. The relationship between power consumption, voltage, and frequency is non-linear,

Table 1 Summary of the literature (2.1)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Migration cost	Scalability	Throughput	Operational costs
[4]-2022	MLBVNE	C++	Improve load balance, acceptance rate	Single point of failure	Improve load balance, acceptance rate, and profit	✓		✓		✓	✓	✓						✓
[5]-2021	DWRS	Mininet	Improve load balance, increase efficiency	Longer search time for large scale server clusters	Improve load balance, increase efficiency	✓		✓		✓	✓	✓						
[6]-2019	SDLB	Mininet	Improve load balance, scalability, fault tolerance	Imbalance among servers in data centers	Improve load balance, scalability, availability, fault tolerance	✓			✓	✓		✓				✓		
[7]-2020	CCLB	Mininet	Increase load balance, quality of service	No evaluation of network latency costs	Increase load balance, quality of service	✓		✓		✓		✓		✓				

Table 1 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Migration cost	Scalability	Throughput	Operational costs
[14]-2023	TSSM	Mininet-Onos	Improved resource allocation, lower communication delay and cost, enhanced migration and response time, increased throughput	No consideration for large scale networks	Optimize resource allocation, reduce migration time and cost	✓			✓			✓	✓	✓	✓			✓
[15]-2024	AMCLBT	Java	Enhanced resource utilization, load balance, reduced energy consumption, and processing time	Issues with security, fault tolerance	Multi-objective load balancing in cloud-edge environments, reducing energy consumption, response time	✓			✓	✓	✓	✓		✓				

Table 1 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Migration cost	Scalability	Throughput	Operational costs
[16]-2020	MCDM	Maple, Matlab	Improve Reliability and Response Time	Single point of failure	Increase load balance, Improve Reliability and Response Time	✓		✓		✓		✓		✓				
[17]-2023	ESCALB	Mininet-Onos	Improving energy efficiency, Reducing processing delay, Enhancing load balancing, Increasing processor efficiency Optimizing controller resource utilization	High computational complexity due to the use of ANP and dynamic allocation algorithms	Enhancing load balancing, Improving energy efficiency, Optimal resource utilization	✓			✓	✓	✓	✓	✓				✓	✓

Table 1 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Migration cost	Scalability	Throughput	Operational costs
[18]-2024	MISOCP	NSFNET-Python	Reduction in migration downtime, Optimal utilization of controllers, Reduction in operational costs, Increase in network throughput	Increased coordination requirements between controllers, Introduction of short-term latency	Optimal utilization of controllers, Reduction in operational costs, Increase network throughput	✓			✓	✓		✓	✓						

Load balancing in software-defined networks

meaning that appropriate adjustments in these parameters can result in significant energy savings while maintaining system efficiency [8, 9, 19–22].

The DVFS method, primarily utilized for energy reduction, also plays a crucial role in effectively identifying suitable VMs and optimally allocating resources during high network loads to meet quality of service requirements and enhance network resource utilization [19, 23–29].

Some researchers have integrated DVFS with other optimization methods within SDN to select optimal VMs for request allocation. This integration has led to reductions in energy consumption, enhancements in QoS, and improvements in load balancing [5, 19–21].

Javadpour et al. [8] cloud computing is highlighted as a technology for providing Internet services, aimed at reducing operational costs and increasing resilience. Despite its advantages, challenges like resource consumption and load distribution remain. The authors propose a load balancing strategy utilizing DVFS to optimize energy consumption. DVFS manages power by adjusting processor frequency, significantly reducing energy use. This approach also improves load balancing and reduces power consumption in the cloud network.

Mahmoudi et al. [9] SDN is highlighted as a promising technology to enhance network performance, though load imbalance remains a critical issue that diminishes QoS. To mitigate load imbalance and reduce delays, SDN combined with DVFS is utilized, dynamically balancing traffic load across servers and optimizing resource use even with high VM numbers. This study emphasizes dynamic traffic handling, considering VM overload, host efficiency, and user load to enhance SDN performance and QoS.

Javadpour et al. [22] aims to improve energy efficiency and performance in cloud computing by reducing energy consumption while maintaining service quality. It introduces intelligent task scheduling using DVFS with two implementations: SFB for selecting optimal machines using a scoring function, and a micro-genetic algorithm with lower complexity, effectively enhancing energy efficiency in cloud environments.

Kumar et al. [25] The P2BED-C model optimizes energy-efficient cloud data center management by integrating DVFS, peer-to-peer load balancing, and DENS to enhance computational efficiency. Validated on the OpenStack platform, it outperforms traditional scheduling methods like FCFS and Round Robin in energy savings and performance. However, the model lacks machine learning-based optimization, which could further refine energy prediction and dynamic resource allocation. Future research should explore machine learning-driven strategies to improve adaptability and efficiency in cloud environments.

Panda et al. [30] introduced a reinforcement learning (RL)-based approach to optimize DVFS configurations, reducing processor energy consumption while maintaining system performance. By integrating Q-learning with DVFS, the method achieved 20% lower energy usage than traditional DVFS techniques without affecting efficiency. This adaptive model dynamically adjusts voltage and frequency, enhancing real-time load balancing and energy efficiency. Unlike static DVFS, it effectively responds to workload variations, making it a scalable solution for edge computing, data centers, and IoT applications.

Wang et al. [31] proposed a hybrid reinforcement learning and DVFS-based approach to optimize energy consumption and latency in edge computing. The method integrates multi-sample modeling and a joint optimization strategy for microservice deployment and request routing. Using a Jackson queuing network, it analyzes system delays, while DA-RSPPO optimizes static decisions and EA-DFS dynamically adjusts processor frequency. Experimental results show significant energy savings and improved response latency, making it a promising solution. Future research could extend this model to multi-cloud environments and IoT systems for greater scalability.

Geng et al. [32] PowerLens is a deep learning-integrated framework that optimizes DVFS configurations for deep neural networks (DNNs) by clustering power behavior. It mitigates frequency fluctuations, configuration delays, and transferability issues in conventional methods. Experimental results show up to 88.64% energy efficiency improvement, surpassing existing techniques. Its scalability and efficiency make it a promising solution for CPUs, cloud computing, and large-scale systems, enhancing energy optimization and computational performance.

El Mahjoub et al. [33] A power and performance management framework for DVFS-based systems integrates Markov chain stochastic modeling with multi-objective optimization, dynamically adjusting frequency and voltage based on workload and QoS requirements. Simulations show a 32% energy reduction while enhancing computational performance. Compared to static DVFS, standard Markov models, and queuing theory-based methods, it offers superior power-aware computing efficiency.

Shuaib et al. [34] proposed DEELB, a dynamic load balancing and energy optimization framework for IoT environments, integrating adaptive learning models and multi-layer load balancing. Experimental results in CloudSim show that DEELB reduces energy consumption by 30% and improves processing delay by 27%, outperforming conventional methods. The framework efficiently allocates tasks to low-power nodes, optimizing resource distribution and management efficiency. Key advantages include enhanced scalability, lower operational costs, and congestion prevention. However, high computational complexity and dependence on advanced hardware remain critical challenges.

Piga et al. [35] DVFS Boosting was explored as a scalable and secure approach to enhance data center capacity, tackling power consumption, hardware heterogeneity, and service reliability. Over three years of deployment, it added 12 megawatts of capacity, equivalent to half a new data center. Unlike conventional methods, it integrates risk management, heterogeneous data analytics, and machine learning for optimized service selection. Experimental results show increased computational capacity without performance trade-offs while managing power constraints efficiently. Future research could extend this framework to cloud infrastructures, incorporating AI and blockchain for improved resource allocation and energy efficiency.

Irfan et al. [36] proposed an adaptive task allocation and power management approach for MPSoCs with NoC architectures, leveraging DVFS to enhance energy efficiency and processor performance. The method integrates adaptive scheduling and data path management, achieving 38% energy reduction and 29% lower processing delay through dynamic frequency and voltage adjustments. Compared to

conventional techniques, it improves task distribution efficiency, extending processor lifespan and overall system performance. The approach is applicable to distributed processing and real-time operating systems (RTOS), though challenges remain in implementation complexity, hardware dependency, and computational overhead.

Islam et al. [37] proposed ELITE, an energy-efficient framework for Mobile Edge Computing (MEC) that integrates a three-layer architecture, a DVFS-based UE power management model, and a layered scheduling algorithm for optimal resource allocation. Experimental results show that ELITE reduces UE energy consumption by 50% and improves processing delay by 30%. Additionally, it enhances computational load distribution while significantly reducing task failure rates. This approach demonstrates promising potential for optimizing MEC environments by balancing energy efficiency and system performance.

Cengiz et al. [38] addressed energy-efficient resource management in data centers, proposing Intelligent Load Balancing Algorithms (ILBAs) to optimize energy consumption and computational efficiency. The ILBA method employs machine learning-based load balancing for dynamic resource allocation in cloud environments. Experimental results show a 40% reduction in energy consumption, a 30% decrease in processing delay, and improved resource utilization. Given its scalability, this approach offers a promising solution for modern data centers, ensuring efficient resource allocation and sustainable operations.

Hagras et al. [39] proposed BlueMoon, an innovative DVFS-based scheduling mechanism to reduce energy consumption in processing systems. By extending task execution time intervals, BlueMoon optimizes power usage without increasing overall execution time. Experimental results indicate a 21% reduction in energy consumption compared to conventional methods, while also enhancing computational efficiency. This approach offers a promising solution for energy-aware scheduling in modern computing environments.

Muthusamy et al. [40] introduced a Q-Learning-based load balancing model for optimized resource allocation in cloud environments. This approach dynamically analyzes processing loads and autonomously adjusts resource distribution to prevent server overloading. Experimental results demonstrate a 55% improvement in processing time, a 40% increase in resource utilization, and a 20% boost in scalability, while also reducing operational costs. Despite its advantages, the model's high computational complexity necessitates advanced hardware and additional processing resources for full deployment.

Zhou et al. [41] propose an Intelligent Energy Consumption Model (IECL) for cloud-based manufacturing, focusing on real-time and accurate energy prediction in data centers. Their approach integrates machine learning techniques, including Support Vector Machines (SVM), Random Forest (RF), and Grid Search (GS), to estimate energy consumption under varying workloads. By leveraging RF for feature selection and GS for parameter optimization, the model enhances predictive accuracy, achieving an absolute error below 1.4%. The results demonstrate that IECL outperforms existing models, making it an effective solution for energy-efficient resource allocation in cloud data centers and industrial manufacturing environments.

Zhou et al. [42] introduce ECMS, an intelligent energy consumption model tailored for mobile edge computing (MEC) environments. By integrating Elman

neural networks (ENN) with feature selection techniques, the model captures 29 key energy-related parameters to predict and optimize edge server power usage under dynamic workloads. ECMS is validated across CPU-intensive, transactional, and I/O-intensive tasks, and achieves superior accuracy and lower training overhead compared to traditional regression- and PMC-based models. Its adaptability makes it a promising tool for energy-aware resource management in green MEC systems.

Zhou et al. [43] propose two adaptive, energy-aware algorithms designed to minimize power consumption and service level agreement (SLA) violations in cloud data centers. Unlike prior approaches, their method considers both CPU and memory utilization along with workload types (e.g., CPU- and I/O-intensive) during virtual machine (VM) deployment. By introducing an adaptive three-threshold framework and optimizing VM placement through energy efficiency maximization, the proposed algorithms achieve significantly better results in energy savings and SLA compliance compared to traditional threshold-based techniques. Real-world simulations using PlanetLab workloads and CloudSim confirm the model's effectiveness under dynamic cloud conditions.

2.2.1 Energy optimization techniques

Several energy-efficient techniques exist in software-defined networks and cloud computing environments. The choice of DVFS was based on the following key advantages:

- *Fine-Grained Power Control* Unlike sleep scheduling, which switches devices on and off completely, DVFS allows dynamic power scaling by adjusting processor voltage and frequency, ensuring a balance between energy savings and performance [8, 9].
- *Minimal Latency Overhead* Sleep scheduling requires switches and servers to power down and wake up, introducing latency. In contrast, DVFS optimizes energy use in real-time without significant latency penalties [27].
- *Seamless Integration with Load Balancing* Since SDN-PG dynamically adjusts task allocation, DVFS complements this approach by adjusting power levels based on real-time processing demand [21].
- *Energy-Performance Trade-off Optimization* DVFS provides a scalable solution for reducing energy consumption without sacrificing QoS metrics like response time and computational overhead [25].

The Comparisons are presented in Table 2.

2.2.2 Justification for selecting DVFS in the proposed method

- *Real-Time Adaptability* DVFS dynamically adjusts power usage without introducing delays in load balancing [21].
- *Energy-Performance Trade-Off* Unlike sleep scheduling, DVFS allows energy savings without degrading QoS metrics [26].

Table 2 Comparison of DVFS versus alternative energy-saving techniques

Technique	Energy savings potential	Impact on performance	Latency overhead	Suitability for SDN-PG	References
DVFS (Proposed)	High (Adjusts dynamically)	Minimal impact	Low	Best suited (Real-time load-aware power scaling)	[9]
Sleep scheduling	Higher than DVFS	High performance impact	Significant wake-up latency	Not ideal for real-time SDN traffic	[27]
Dynamic link adaptation	Moderate	Moderate impact on latency	Medium	Better suited for wireless networks	[23]
Adaptive resource scaling	High	Requires additional virtualization overhead	Medium	Best for cloud data centers, but less effective in SDN switches	[20]

- *Optimized for SDN-PG Architecture* Since SDN-PG continuously monitors network traffic, DVFS effectively scales processor power based on dynamic workloads [8].

The findings from the studies in Sect. 2.2 highlight the critical role of DVFS in optimizing resource allocation and enhancing network performance in cloud computing and SDN. The integration of DVFS with other optimization techniques leads to reduced energy consumption, improved load balancing, and enhanced QoS. Despite challenges such as load imbalance and latency in high-traffic environments, these approaches significantly contribute to improved energy efficiency, scalability, and resilience of network infrastructures.

Table 3 presents a comparative analysis of different approaches that integrate DVFS for energy-efficient load balancing in SDNs. The analysis highlights key advantages such as reduced energy consumption and improved QoS, while also addressing limitations like increased computational overhead.

2.3 Load balancing in SDNs using artificial intelligence and machine learning

Soltani et al. [44] proposed a resource allocation and load balancing model for SDNs and virtual network mapping (VNM), focusing on request acceptance rates, latency reduction, cost minimization, and QoS enhancement. Using Fuzzy Markov Logic and Time Slot Scheduling, the method lowers costs, increases acceptance rates, and improves QoS, making it highly adaptable for dynamic network environments. However, its reliance on centralized control poses a risk of single points of failure, potentially impacting network robustness.

Keshri and Vidyarthi [45] tackle the NP-hard problem of VM placement in cloud data centers, focusing on communication awareness and energy efficiency. They propose a hybrid ACO-GWO approach, combining Ant Colony Optimization (ACO) for broad exploration and Grey Wolf Optimization (GWO) for precise refinement. This method reduces resource usage, enhances network traffic management, and improves energy efficiency, ensuring an optimized communication-aware VM placement.

Maqsood et al. [46] recognizing the rapid growth of mobile edge computing (MEC) and the increasing demands of smart devices, propose a novel load distribution method for optimizing load balancing in MEC networks. They utilize K-means clustering techniques to identify overloaded and underloaded servers, subsequently redistributing workloads from overloaded servers to underloaded ones. This innovative approach helps maintain load balance, reduces delay, and optimizes resource utilization in the network.

Sridevi and Saifulla [47] propose an efficient method for load balancing among controllers in Distributed Software Defined Networks (DSDN) using the Artificial Bee Colony (ABC) algorithm. The approach aims to optimize load distribution, thereby reducing delay and improving QoS, even under heavy traffic conditions. Controller loads are measured using PACKET_IN messages, representing real-time controller loads. Load imbalance is detected through the coefficient of variation

Table 3 Summary of the literature (2.2)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[8]-2023	DVFS	Cloudsim	Reduced energy consumption, load balance, reduced congestion, enhanced system efficiency	Potential for SLA violations	Reduce energy consumption, increase system efficiency	✓		✓										
[9]-2022	SDN-DVFS	Omnet++	Enhanced energy efficiency, improved QoS, reduced synchronization costs	Single point of failure	Improve load balance, energy efficiency, QoS	✓		✓		✓	✓				✓		✓	

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[22]-2023	GIoT-DVFS-SFB	CloudSim	Shortened task execution time, reduced energy consumption, optimized resource allocation	Unreliability	Reduce energy consumption, improve QoS, resource allocation	✓		✓		✓	✓				✓			
[29]-2021	DV-DVFS	Apache Spark	Load balance, reduce energy consumption	High costs	Load balance, reduce energy consumption	✓		✓		✓	✓	✓						✓

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[25]-2022	DVFS-DENS	CloudSim	Energy consumption reduction, operational cost reduction, optimized task management, computational performance optimization	Need for infrastructure modifications, dependency on renewable energy sources	Energy consumption reduction, operational cost reduction	✓		✓		✓	✓	✓					✓	✓
[30]-2024	RL-DVFS	Python	Energy consumption reduction, improved load balancing	lack of multi-core system utilization, single point of failure	Energy consumption reduction, improved load balancing	✓		✓		✓	✓	✓						

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[31]-2025	DA-RSPPO, DVFS	CloudSim	Energy consumption reduction, delay optimization, improved request processing efficiency, and intelligent energy management	Increased computational overhead in the initial learning phase	Energy consumption reduction, delay optimization	✓		✓		✓	✓	✓					✓	
[32]-2024	PowerLens-DNN-DVFS	Python	Energy consumption reduction, frequency fluctuation minimization, and delay reduction	Increased overhead	Energy consumption reduction, delay optimization	✓		✓		✓	✓		✓					

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[33]-2023	Markov Chain-DVFS	CloudSim	Energy consumption reduction, response time optimization, increased flexibility, and high scalability	Increased overhead	Energy consumption reduction, response time optimization	✓		✓		✓	✓			✓	✓	✓		

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[34]-2023	DEELB-DVFS	CloudSim	Energy consumption reduction, increased processing speed and reduced delay, improved scalability, and intelligent load management	Implementation complexity, single point of failure	Energy consumption reduction, improved load balancing	✓		✓		✓	✓		✓		✓			

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[35]-2024	DVFS Boosting	CloudSim	Increased processing capacity, energy consumption reduction, operational cost reduction, and enhanced efficiency	Management complexity, lack of SDN utilization	Increased processing capacity, energy consumption reduction, and operational cost reduction	✓		✓		✓	✓						✓	✓

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[36]-2024	DVFS-NoC	CloudSim	Energy consumption reduction, processing delay reduction, processor efficiency improvement, enhanced computational load balancing, traffic reduction	Requirement for modern hardware with NoC support	Energy consumption reduction, improved load balancing, and traffic reduction	✓		✓		✓	✓	✓	✓					

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[37]-2024	ELITE-DVFS	PureEdgeSim	Energy consumption reduction, resource allocation optimization, and load balancing improvement	High computational complexity, lack of performance evaluation in dynamic environments	Energy consumption reduction, improved load balancing	✓		✓		✓	✓	✓						
[38]-2024	DVFS-ML	CloudSim	Energy consumption optimization, processing delay reduction, operational cost reduction, and improved load balancing	Low compatibility with traditional data centers	Energy consumption reduction, improved load balancing	✓		✓		✓	✓		✓					✓

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[39]-2024	BlueMoon	CloudSim	Energy consumption optimization, improved task distribution, and enhanced efficiency	single point of failure	Energy consumption reduction, improved task distribution	✓		✓		✓	✓	✓					✓	

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[40]-2023	Q-Learnig	CloudSim	Enhanced efficiency, operational cost reduction, energy consumption reduction, load balancing maintenance, and network delay reduction	High computational complexity, single point of failure	Operational cost reduction, energy consumption reduction, and improved load balancing	✓		✓		✓	✓		✓					✓

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[41]-2022	IECL	Python	Energy Consumption Optimization, Resource Utilization, Comprehensive Evaluation, Workload-aware	Training cost ignored, Static deployment,	Energy Consumption Optimization, Resource Utilization		✓	✓			✓	✓						
[42]-2021	ECMS	Python	High prediction accuracy, Low computational overhead, Energy Consumption Optimization	Limited task scope, Static inference model	Energy Consumption Optimization		✓	✓			✓	✓						

Table 3 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Scalability	Flexibility	Throughput	Operational costs
[43]-2018	mSLAv	CloudSim	Energy Efficiency, SLA	No integration with machine learning techniques	Power Consumption Optimization, SLA		✓	✓			✓	✓					✓	

Load balancing and energy optimization using DVFS in SDNs

(CV), and if CV exceeds 0.4, the ABC algorithm is triggered to achieve load balance by selecting the optimal controllers and switches.

Forghani et al. [1] the authors propose a dynamic optimization scheme to enhance load balancing and energy efficiency in SDNs. Utilizing the Krill Herd algorithm, the approach optimizes network task allocation to VMs, effectively balancing load and reducing energy consumption while improving overall network performance.

Torkzadeh et al. [48] the authors address the challenge of reducing energy consumption while maintaining quality of service and achieving load balancing in SDNs. They propose a dual-phase routing mechanism: in the first phase, an offline Ant Colony Optimization algorithm identifies an optimal graph with minimal active switches; in the second phase, real-time routing aims to distribute link load evenly while ensuring QoS for user flows.

Mahmoudi et al. [49] aim to introduce and evaluate a novel approach called MBL-DSDN, aimed at optimizing server traffic distribution and enhancing QoS parameters in DSDN. This method utilizes two modules: micro-clustering (MC) and bidirectional long short-term memory (B-LSTM) to reduce response time, minimize migration costs, and ultimately improve load balancing in the network.

Jeong et al. [50] explore the use of SDN with deep reinforcement learning to address traffic congestion on specific links and improve QoS. The paper proposes a novel load balancing method for large-scale SDNs using deep deterministic policy gradient (DDPG). DDPG serves as the core of the model, featuring a decision-making agent (DMA) that uses deep neural networks and reinforcement learning to optimize network performance and determine optimal paths for load distribution. The study does not address server-side traffic, presenting a potential area for future innovation.

Shahrbabaki et al. [51] introduce the SDN-LB algorithm, which optimizes load distribution for IoT video analysis environments using Software-defined networking. The main innovation lies in the use of adaptive thresholds that dynamically adjust based on real-time network conditions, enhancing resource utilization and reducing delay compared to static threshold approaches. The study employs a hybrid method incorporating dynamic optimization and machine learning to effectively improve system performance.

Buhurcu and Çarkacıoğlu [2] propose a two-tier model for improving load balancing in cellular networks using reinforcement learning techniques. This model integrates centralized predictions of cell user numbers with decentralized reinforcement learning to optimize parameter adjustments. The approach effectively distributes users across cells, enhancing flexibility, reducing interference, and improving network efficiency.

Zhou et al. [52] present an innovative framework for optimizing task allocation and resource scheduling in edge computing, aiming to enhance load balancing and reduce energy consumption. Utilizing a multi-objective optimization approach combined with deep reinforcement learning (DRL), the study effectively balances server loads and minimizes energy usage, enhancing overall system performance in edge computing environments.

Xiang et al. [12] propose a DRL-based load balancing strategy for multi-controller SDNs, addressing inefficiencies in static switch-controller connections that cause

load imbalance. By leveraging Markov decision process (MDP) and double deep Q-network (DDQN), the approach optimizes switch migration, enhancing resource utilization and reducing convergence time. Simulation results show that the DRL-SMS strategy significantly improves controller load balancing and accelerates equilibrium, making it well-suited for dynamic, high-traffic networks.

Jain et al. [53] introduced a Q-Learning-based approach for load balancing and fault tolerance in SDN. By leveraging online Q-Learning, the model optimizes switch migration, reducing relocation costs and enhancing packet response rates. Experimental results indicate 30% lower processing delay and 25% improved resource utilization, outperforming conventional methods in load distribution and scalability. Additionally, integrating controller prioritization and conflict-free migration ensures system stability. Despite its advantages, computational complexity and parameter fine-tuning remain challenges. Future research could incorporate deep learning models to further refine controller efficiency and reduce processing overhead.

Saeedi et al. [54] proposed a Particle Swarm Optimization (PSO)-based approach for controller placement in SDN, aiming to ensure load balancing and network reliability. The method optimizes controller allocation, reducing resource consumption, leading to a 20% decrease in controller count and a 6% improvement in load balancing. It also reduces propagation delay by 15%, outperforming traditional models like Varna and CNPA. By assigning two controllers per switch, the approach minimizes failure risks. However, PSO's computational complexity may limit its scalability in large networks. Future work could incorporate deep learning to enhance PSO parameter optimization, improving efficiency and adaptability.

Zhou et al. [55] presents IADE, an improved version of the differential evolution (DE) algorithm, designed to enhance sustainability in 6G networks. IADE adaptively tunes parameters such as mutation factor, crossover rate, mutation strategy, and selection mechanism to address issues like slow convergence and local optima in standard DE. The algorithm is structured for fast convergence and better global search capability. Extensive experiments on 30 benchmark functions demonstrate IADE's superior performance in solution accuracy and convergence speed. IADE is well-suited for large-scale 6G-enabled networked data centers and energy-aware task scheduling.

Zhou et al. [56] introduce ISC-QL, a novel two-phase strategy that integrates Improved Spectral Clustering with Q-Learning for optimizing edge server deployment within intelligent Internet of Vehicles (IoV) systems. The proposed approach simultaneously addresses three critical objectives minimizing latency, reducing energy consumption, and enhancing workload distribution. Validated through real-world data, ISC-QL demonstrates significant improvements over existing baseline methods, achieving up to 50% enhancement in load balancing, 22% reduction in energy usage, and 16% decrease in average latency, highlighting its suitability for large-scale and adaptive intelligent transportation systems (ITS).

Zhou et al. [57] propose AFED-EF, an adaptive VM allocation algorithm designed to improve energy efficiency and reduce SLA violations in cloud data centers hosting IoT applications. By introducing a four-threshold mechanism and combining it with a VM selection and placement strategy, the algorithm dynamically

responds to fluctuating workloads. Extensive simulations using real-world PlanetLab data show that AFED-EF outperforms existing methods in energy consumption, SLA compliance, and overall energy efficiency, making it suitable for sustainable cloud-based IoT systems.

The studies reviewed in Sect. 2.3 highlight various approaches for optimizing resource allocation and improving load balancing in SDNs and cloud environments using artificial intelligence algorithms and machine learning techniques. These methods address challenges such as load imbalance, energy consumption, and network congestion, offering scalable and resilient solutions for dynamic, high-traffic environments. Despite these advancements, challenges related to centralized control and real-time adaptability remain areas that require further research.

Several recent studies have leveraged artificial intelligence (AI) and machine learning (ML) for optimizing load balancing in SDNs. Table 4 summarizes these approaches, categorizing them based on their use of reinforcement learning, deep learning, and heuristic optimization techniques. The results indicate that AI-driven models significantly improve dynamic traffic distribution while reducing network congestion.

Based on the analysis of previous studies, it is evident that improving and addressing the issue of dynamic load balancing in SDNs using artificial intelligence algorithms, particularly machine learning techniques, holds significant potential and has garnered considerable attention from researchers. Each of these studies has aimed to enhance QoS metrics through various techniques and algorithms.

Building upon prior research, this study will focus on the promising topic of dynamic load balancing in SDNs to reduce energy consumption by leveraging the DVFS technique. For the first time, we will employ the PG method within a distributed architecture to optimize energy consumption, processing load, response time, and load balancing. The simulation results, along with a comparative analysis of the proposed approach against two existing methods CCA-PSO and DRL-SMS will be presented in Sects. 3 and 4.

3 Proposed method and problem formulation

The policy gradient (PG) algorithm is a fundamental technique in reinforcement learning, known for its adaptability and effectiveness in solving complex problems. It belongs to the category of policy optimization methods within Model-Free approaches. Unlike algorithms that focus on estimating value functions, PG directly aims to learn an optimal policy. A policy defines a set of rules or probability distributions that guide an agent's actions based on its observations. The primary objective of this approach is to determine a policy that maximizes cumulative rewards over time, thereby improving the agent's decision-making capabilities [58, 59].

In software-defined networks, the highly dynamic and unpredictable nature of network traffic necessitates intelligent and adaptive load balancing strategies. Traditional static load balancing techniques fail to efficiently respond to fluctuating network conditions, often leading to a decline in QoS and increased energy consumption [60].

Table 4 Summary of the literature (2.3)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
[44]-2025	SDN-FoP-MVM	NS2-Mininet	Reduced cost, Reduced delay, increasing request acceptance rates, enhanced QoS	High computational complexity	Improve Resource Management	✓		✓				✓	✓				✓	✓
[45]-2024	ACO-GWO	CloudSim	Reduced energy consumption, improved network efficiency, preventing congestion, faster convergence	Increased execution time for larger scales	Optimize resource usage, reduce energy consumption, reduce traffic, balance load	✓		✓										
[46]-2024	K-means clustering	Python	Reduced delay, enhanced resource efficiency, improved stability, better load balance	Difficult implementation on a large scale	Improve load balancing, efficient resource utilization	✓		✓		✓	✓	✓					✓	

Table 4 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
[47]-2023	ABC	Mininet	Improved load balance, reduced delay, enhanced quality	High migration cost	Load balance, reduce delay, enhance quality	✓			✓	✓		✓	✓	✓				
[1]-2024	KHLB	Omnet++	Enhanced load balancing, increased energy efficiency, scalability, flexibility	High computational complexity	Simultaneous improvement in load balancing and energy efficiency	✓		✓		✓	✓					✓		
[48]-2021	ACO	Mininet	Reduced energy consumption, link load balance, enhanced QoS, prevent congestion at switches	Packet loss	Reduce energy consumption, enhance QoS, balance load on links	✓		✓		✓	✓	✓						

Table 4 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
[49]-2024	B-LSTM	Omnet++	Reduced response time, improved load balance, reduced migration costs, enhanced scalability	Increased computational overhead	Optimize server traffic distribution, improve QoS	✓			✓	✓				✓		✓		✓
[50]-2024	DDPG	MatLab	Improved load balance, network performance, reduced peak link load	No evaluation of large scale networks	Improve load balance, reduce link congestion, enhance QoS	✓		✓		✓		✓					✓	
[51]-2024	SDN-LB	Python	Improved resource utilization, reduced delay, increased energy efficiency	Increased computational overhead	Enhanced resource utilization, reduced delay, improved energy efficiency	✓		✓			✓	✓	✓					

Table 4 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
[2]-2024	RL	NS-3	Improved Load Balancing, reduced interference, increased network efficiency	Requirement for extensive data, long reaction time to rapid changes, unsuitable for large networks	Improve load balance, reduced interference, increased efficiency	✓		✓		✓							✓	
[52]-2024	DRL	Python	Reduced energy consumption, improved load balance, optimized resource usage, reduced response time	No evaluation of large scale networks	Reduce energy consumption, enhance load balance	✓		✓		✓	✓	✓		✓				
[12]-2022	DRL-SMS	Mininet	Improved load balance, reduced migration costs, shortened load balancing time	No evaluation of large scale networks	Improve load balance, reduce migration costs, shorten load balancing time	✓			✓	✓		✓						✓

Table 4 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
[53]-2024	Q-Learning-LB	Mininet	Energy consumption reduction, enhanced network stability, optimized load balancing, reduced response time, improved QoS	High computational complexity and cost	Energy consumption reduction, improved load balancing, enhanced (QoS)	✓			✓	✓	✓			✓	✓			
[54]-2025	PSO-LB	Matlab	Network cost reduction, enhanced scalability, improved load balancing, reduced propagation delay, and increased network efficiency	Requirement for more powerful computational resources	Network cost reduction, improved load balancing, and reduced propagation delay	✓			✓	✓			✓			✓		✓

Table 4 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
[55]-2021	IADE	Matlab	Improved convergence speed and accuracy, Better workload balancing and reduced task execution time	Higher Computational Overhead	Improving load balancing, reduced task execution time	✓		✓		✓		✓						
[56]-2025	ISC-QL	Python	Improving load balancing, reducing energy consumption, minimizing Delay	Limited scope in heterogeneous environments, Computational overhead	Improving load balancing, reducing energy consumption	✓		✓		✓	✓		✓					
[57]-2021	AFED-EF	CloudSim	Improved efficiency, decrease SLA, Reducing energy consumption	Assumes uniform traffic	minimizing energy consumption and SLA violations	✓		✓			✓	✓					✓	

Table 4 (continued)

References	Algorithm	Simulation	Advantages	Drawbacks	Objectives	Dynamic	Static	Centralized	Distributed	Load balancing	Energy consumption	Resource utilization	Delay	Response time	Overhead	Scalability	Throughput	Operational costs
Propose Method	SDN-PG	Omnet++	Improving load–balancing, reducing energy consumption, minimizing latency, enhancing resource utilization, reducing response time, reducing overhead		Improving load balancing, reducing energy consumption, enhancing resource utilization, and QoS	✓			✓	✓	✓	✓	✓	✓	✓	✓		

In contrast, reinforcement learning-based methods, particularly the policy gradient approach, enable SDN controllers to dynamically and autonomously learn optimal strategies for traffic distribution by continuously interacting with the network environment [61, 62].

The PG method iteratively enhances load balancing decisions by following a structured process [61]:

- *Real-time Network Monitoring* The SDN controller observes the current state of the network, including critical metrics such as traffic load, latency, server utilization, and energy consumption.
- *Action Selection* Based on observed network conditions, the PG algorithm probabilistically selects a load-balancing action (e.g., distributing tasks to a specific server).
- *Reward Calculation* After executing the selected action, the network provides immediate feedback (a reward) based on achieved performance. The reward is computed by considering multiple factors like reduced latency, improved load distribution, minimized overhead, and enhanced energy efficiency.
- *Policy Update* The PG algorithm utilizes the received reward to adjust the policy parameters. Formally, the gradient of the expected cumulative reward with respect to policy parameters (θ) is calculated as follows Eq. (3.1):

$$\nabla_\theta J(\theta) = E\left[\sum_{t=0}^{T} \nabla_\theta Log\pi_\theta(a_t|s_t)A_t\right] \tag{3.1}$$

Where variables are clearly defined:

$\nabla_\theta J(\theta)$: Gradient of the expected reward function with respect to policy parameters θ.
π_θ: The policy function, representing the probability of selecting action a_t given state s_t
A_t: Advantage function, measuring how good the chosen action a_t is compared to the baseline.

In SDNs, the load balancing system is tasked with effectively distributing network traffic across the available servers to enhance overall network performance and prevent excessive strain on any single node. Several critical factors influence the efficiency of a load balancing system:

- *Network Traffic* The volume of network traffic is a pivotal factor that impacts load balancing efficiency. When traffic volume is high, the system must ensure that traffic is intelligently and efficiently distributed among the network nodes (e.g., switches, servers, and controllers) to avoid overloading individual nodes.
- *Number of Nodes* The quantity of nodes within the network plays a significant role in load balancing performance. A higher number of nodes allows for

more flexible and efficient distribution of the traffic across the system, reducing the risk of bottlenecks.

- *Node Locations* The physical or logical locations of the nodes can influence the performance of the load balancing system. For instance, when nodes are geographically dispersed, the latency introduced by the longer data paths between nodes can impact the responsiveness of traffic distribution.
- *Type of Traffic* Different types of network traffic have distinct characteristics and requirements. For example, Voice over IP (VoIP) traffic is particularly sensitive to delays, so the load balancing system must prioritize minimizing latency while distributing VoIP traffic across the nodes.
- *Load Balancing Policies* The strategies or policies governing how traffic is distributed among nodes are also crucial. A load balancing system might employ policies that evenly distribute traffic across servers or direct traffic toward nodes with the least load, depending on the specific requirements of the network [4, 5, 8, 63].

The simulation setup follows a structured network topology with defined parameters for network controllers, virtual machines, and task distribution. Table 5 provides a comprehensive overview of the simulation variables, including network dimensions, processing capacities, and energy consumption parameters.

If we denote x as the load assigned to each server, then $F(x)$ should be considered the objective function of the load balancing system, which is typically a function of network traffic, the number of servers, the location of the servers, the type of traffic, and the load balancing policies. Additionally, $G(x)$ represents a set of constraints that must be adhered to in the load balancing system. These constraints may include server capacity limits, delay time restrictions, and constraints related to load balancing policies.

To define $F(x)$, it is necessary to refer to the objective of a load balancing system and the improvements expected in network performance. This objective can be achieved using the following objective function:

- Minimizing delay (D_n)
- Minimizing processing overhead (O_n)
- Minimizing energy consumption (E_n)
- Ensuring load distribution across the network (L_b)

Based on this, the mathematical representation can be written as:

$$F(x) \propto \frac{1}{D_n}, \frac{1}{O_n}, \frac{1}{E_n}, \frac{1}{L_b} \tag{3.2}$$

Since the load distribution in the network is equivalent to the deviation from te average processing load assigned to all nodes, the smaller the deviation from the mean, the closer the processing load distribution is to being equal across the nodes. In fact, it indicates that the difference in processing load between nodes is minimized. To express the above relationship mathematically, weight coefficients that

represent the best linear approximation of $F(x)$ should be added to the equation. However, the first step in this process is mapping the four variablesE_n, D_n, L_b, and O_n to the range (0,1] to eliminate the units of each variable. After introducing the four variablesW_D, W_E, W_L, and W_O as the weight coefficients for the mentioned variables, Eq. (3.2) can be transformed into Eq. (3.3).

$$F(x) = W_D \frac{1}{D_n'} + W_O \frac{1}{O_n'} + W_E \frac{1}{E_n'} + W_L \frac{1}{L_b'} \tag{3.3}$$

In this equation, the values E_n', D_n', L_b', and O_n' represent the normalized values of the four variables E_n, D_n, L_b and O_n, respectively. It is important to note that the selection of each of the four weights W_D, W_E, W_L, and W_O will depend on the specific conditions of the problem. Clearly, the solution that yields the highest $F(x)$ is, in theory, the optimal solution for this problem. However, only those solutions that satisfy the constraints $G(x)$, outlined as follows, will be considered valid:

- *Node Capacity* The capacity of the nodes must be respected to prevent excessive overload on the nodes.
- *Delay* The delay in traffic transfer between nodes must be maintained to avoid a reduction in the quality of service.
- *Load Balancing Policie*s The load balancing policies must be adhered to ensure that traffic distribution aligns with the network's requirements.

3.1 Proposed method

The proposed method is a policy-based control approach with a distributed structure. This approach is selected because a controller has more access to information compared to other parts of the network. In an SDN-based network, the controller has critical responsibilities and must manage various components of the network. The architecture of the proposed method is illustrated in Fig. 2.

The proposed load balancing method consists of the following three main components:

- *Distribution Policy Component* This component is responsible for defining load balancing policies. These policies determine how traffic is distributed among servers.
- *Distribution Monitoring Component* This component monitors the status of servers and the network. It collects information related to server loads, traffic transfer delay between servers, and other factors that impact the performance of the load balancing system.
- *Decision-Making Component* This component is responsible for making decisions about how traffic should be distributed among servers. Using the information collected by the monitoring component, it decides which server to direct the traffic to.

Table 5 List of notations used in this paper

Symbol	Description
D_n	Network delay
D_n^t	The delay at time (t)
D_n^{max}	Maximum delay allowed
O_n	Computational overhead
O_n^t	The processing overhead at time (t)
O_n^{max}	Maximum processing overhead allowed
E_n	Energy consumption
E_n^t	The energy consumption at time (t)
E_n^{max}	Maximum energy consumption allowed
L_b	Load distribution rate
L_b^t	The load balancing at time (t)
L_b^{max}	The maximum load balance across the entire network
r	Number of servers
s_i	Processing load assigned to server i
V	SDN network
L	load task
W_D	Weight for minimizing delay
W_E	Weight for minimizing energy consumption
W_L	Weight for load distribution
W_O	Weight for minimizing computational overhead
R^t	Reward at time t
$\varphi(x, a)$	Probability of selecting load distribution action 'a'
$\theta(x, a)$	Combination function of state 'x' and distribution 'a'
$Z(x)$	Normalization factor for probabilities distribution
$Q(x, a)$	Feedback value from environment with action 'a'
W_i	Weights related to different factors in the system
E_i	Energy consumption of server 'i'
f_i	Current processor frequency
$E_{i(0)}$	Energy consumption at base frequency
αi	DVFS coefficient of server 'i'
$F(x)$	Objective function for load balance
$G(x)$	Set of constraints for the load balancing system
C_i	Total processing capacity of server 'i'
$C_{i(r)}$	Required processing capacity for executing the current task
$\overline{C}$	Average processing capacity at each node
N_{vm}	Number of virtual machine
B_i	Memory-to-processor bandwidth
σ_i	Processing power of server 'i'
p_{poweri}	Power consumption of processor i
t_u	Processing task per unit time
$t_{i(p)}$	Processing time of a task by processor
$t_{i(d)}$	Time required to transfer task data from memory to processor

Table 5 (continued)

Symbol	Description
S	The set of network states
x	The load assigned to each server

It should be noted that all the mentioned components will be implemented and executed within the standard OpenFlow controller. In the Sects. 3.1.1, 3.1.2, 3.1.3, each of these components will be described in detail. It should be noted that in Fig. 2, only one controller has been expanded as an example, while all controllers have identical conditions and include the load balancing module.

3.1.1 Distribution policy component

The policy component is responsible for defining load balancing policies. These policies can be defined based on various factors, such as the type of traffic, user identity, or server location. Common load balancing policies are generally implemented using three strategies:

- Round-robin load balancing
- Traffic-based load balancing
- Server load-based load balancing

Since the proposed method is a processing load optimization method, the server load-based load balancing strategy has been chosen for setting the load balance. With this perspective, the proposed method must adjust the load balancing according to the processing load on the servers in such a way that the function $F(x)$ reaches its maximum value. Inspired by reinforcement learning, we can state that $F(x)$ acts as an objective function for policy-making. Before defining the policy function, it is necessary to theoretically review three concepts:

- Environment
- Agent
- Reward or penalty mechanism

These concepts must be clearly defined. In an SDN system, the environment refers to the entire network and its related components, including switches, servers, controllers, and passing traffic. The environment is where load balancing decisions are made, and its state is continuously monitored so that the system can improve its performance. In this method, the SDN controller acts as the agent, and the environment includes the network and the status of the servers.

Let S be the set of network states, which includes the load status of servers and traffic delay, and let A be the set of possible actions, which involves distributing

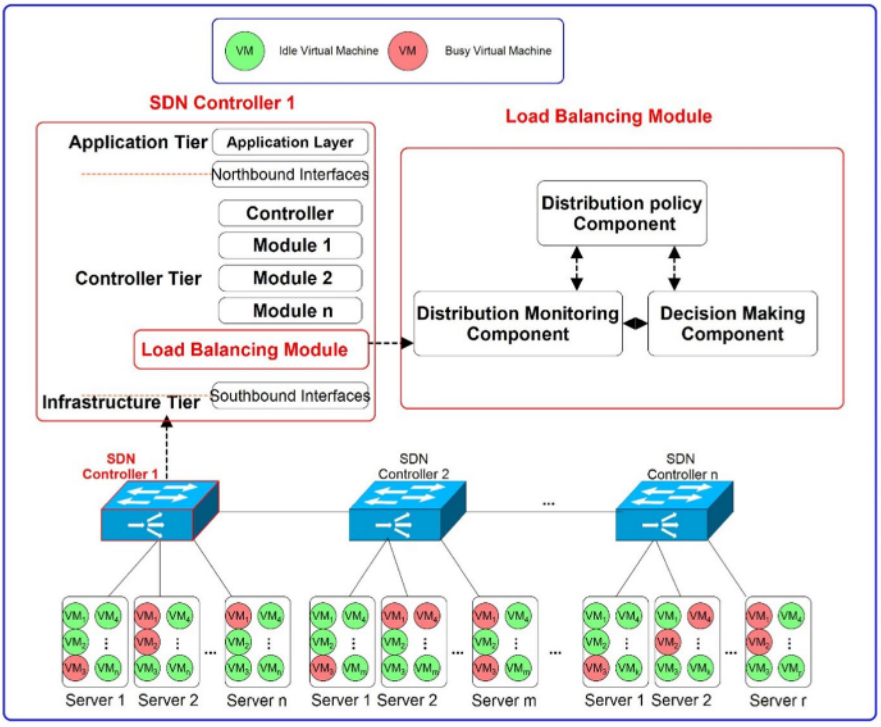

Fig. 2 Proposed method architecture

traffic among servers. The transition function $P:S \times A \times S \rightarrow [0,1]$ defines the probability of transitioning from state s to state s′ when action a is taken. The reward function $R:S \times A \rightarrow R$ represents the reward received after taking action a in state s.

In the given environment and with the defined agent, rewards and penalties should be determined based on the metrics of delay, energy consumption, and processing overhead. The monitoring and decision-making components in the SDN system are responsible for collecting data and applying rewards and penalties. Using the collected information, these components calculate the rewards and penalties and take necessary actions to improve network performance. This information is updated and utilized by the SDN controller.

Policies can be applied across the entire system by adjusting the values of w in Eq. (3.2). The main challenge in this section is how to adjust the w values. To address this challenge, a policy function based on the PG method is used [64]. This policy function, in its general form, is represented by Eq. (3.4).

$$\varphi(x,a) = \frac{\exp\left(w_\theta \theta(x,a)\right)}{Z(x)} \tag{3.4}$$

In this function, $\varphi(x, a)$ represents the probability of selecting the load distribution action a among the servers, given the current load distribution state x. In fact, if the number of servers is equal to r, then $a = [s_1, s_2, \ldots, s_r]$, where s_i is the processing load assigned to the i-th server. w_θ represents the weights of the variables in Eq. (3.3).

$\theta(x, a)$ represents a function that combines the features of the state x and the distribution a. $Z(x)$ is the normalization factor that maps the probability of selecting each possible distribution for state x to a range between [0, 1], ensuring that the sum of all possible probabilities equals 1. For the problem under consideration, the function $\theta(x, a)$ can be expressed in the form of Eq. (3.5).

$$\theta(x, a) = \left[D_n, O_n, E_n, L_b, a\right] \tag{3.5}$$

In fact, the values obtained from the four variables energy consumption, delay, and processing overhead are collected in this function when distribution a is applied. If the current $F(x)$ is available, then Eq. (3.6) can be written to obtain the value of updating the current distribution with distribution a (i.e., the amount of feedback from the environment for action a).

$$Q(x, a) = F(x \rightarrow a) - F(x) \tag{3.6}$$

In this equation, $F(x)$ represents the state of the network after applying load distribution a. In a rotational structure, each of the weights in Eq. (3.2) is updated using Eq. (3.7)

$$\Delta W_i = \alpha Q(x, a)\theta(x, a) \tag{3.7}$$

In this equation, $\Delta W_i = W_i^t - W_i^{t-1}$ and W_i is one of the weights W_D, W_E, W_L, and W_O. Additionally, to maintain the condition in Eq. (3.7), which is a necessary condition for the convergence of Eq. (3.3).

$$W_D + W_O + W_E + W_L = 1 \tag{3.8}$$

Equation (3.9) is utilized.

$$\begin{cases} W_i^t = W_i^{t-1} + \Delta W_i \\ W_{others}^t = W_{others}^{t-1} - \frac{\Delta W_i}{3} \end{cases} \tag{3.9}$$

In this equation, if W_i^t is one of the mentioned weights at the t-th iteration of the algorithm, which is being updated, then W_{others}^t refers to the two other weights besides W_i. If the sum of the four weights at time $t-1$ equals 1, then Eq. (3.10) is always valid based on Eqs. (3.8) and (3.9).

$$W_i^t + 3W_{others}^t = W_i^{t-1} + \Delta W_i + 3\left(W_{others}^{t-1} - \frac{\Delta W_i}{3}\right) = W_i^{t-1} + 3W_{others}^{t-1} = 1 \tag{3.10}$$

The only remaining issue is that W_i^0 for each of the weights will be equal to 1/4.

3.1.2 Distribution monitoring component

The distribution monitoring component is responsible for overseeing the status of the servers and the network. This component collects information related to server loads, delay in traffic transfer between servers, and other factors that impact the performance of the load balancing system. This information is essential for the decision-making component to make informed decisions about how to distribute traffic among servers. The first challenge for this component is measuring the total network response time in the current state. This delay depends on the distribution model x, which represents the current load distribution state [50]. In other words, Eq. (3.11) can be written to calculate the total response time across the network.

$$D_n = \sum_{i=1}^{r} \sigma_i D_i \tag{3.11}$$

In this equation, σ_i represents the processing capacity of the i-th server, which is a hardware feature related to the processor, and D_i is the delay factor relative to the capacity of this server. To obtain the value of D_i, it is assumed that a task has t_u processing units. The time required for the processor to handle this task is calculated as follows:

$$t_{i(p)} = \frac{t_u}{C_i} \tag{3.12}$$

In this equation, C_i represents the total processing capacity of the i-th server. The time required to transfer the data of this task from memory to the processor is calculated as follows:

$$t_{i(d)} = \frac{t_u}{B_i} \tag{3.13}$$

In this equation, B_i represents the memory-to-processor bandwidth. By substituting the relationships for t_p and t_d into this equation, and knowing that $D_i = t_{i(p)} + t_{i(d)}$, Eq. (3.14) is obtained.

$$D_i = \frac{t_u}{C_i} + \frac{t_u}{B_i} \tag{3.14}$$

By simplifying this equation, and assuming $B_s = \frac{C_i}{C_{i(r)}}$ [26]. the server's bandwidth is equal to the server's processing capacity divided by the processing capacity required to perform the current task $C_{i(r)}$ and $t_u = C_{i(r)} * t_{i(d)}$ Eq. (3.15) is obtained:

$$D_i = \frac{C_{i(r)}}{C_i} + \frac{t_{i(d)}}{C_i} \tag{3.15}$$

This equation shows that processing delay decreases as the remaining capacity of the server increases, and it also decreases with the increase in memory-to-processor bandwidth. By substituting the value from Eq. (3.15), one can estimate the delay resulting

from task distribution in distribution a using Eq. (3.11), given the known variables in this equation.

Another challenge in this section is calculating the processing overhead on a server. In SDN, the processing overhead for each server depends on several factors, including:

- *Number of control tasks* The number of control tasks performed by the SDN controller for each server.
- *Complexity of control tasks* The complexity of the control tasks performed by the SDN controller for each server.
- *Server processing capacity* The processing capacity of the server that is used to handle control tasks.

If we assume that server iii has a processing capacity of C_i, then server iii can handle a maximum of C_i processing units in each time unit [30]. If server i performs Ni control tasks, and the processing required to perform each of these tasks is P_i processing units, then the time required to complete a control task is calculated as follows:

$$t_i = \frac{P_i}{C_i} \tag{3.16}$$

In this case, the processing overhead of server i can be calculated using Eq. (3.17):

$$O_i \propto \sum_{i=1}^{N_i} t_i \tag{3.17}$$

By substituting the equation for t_i into this equation, Eq. (3.18) is obtained.

$$O_i \propto \sum_{i=1}^{N_i} \frac{P_i}{C_i} \tag{3.18}$$

Assuming that each server has a processing capacity of σ_i, Eq. (3.19) can then be derived to calculate the processing overhead.

$$O_n = \sum_{j=1}^{r} \sigma_j O_j = \sum_{j=1}^{r} \sigma_j \sum_{i=1}^{N_i} \frac{P_i}{C_i} \tag{3.19}$$

The next challenge in this section is to find an equation to calculate the total energy consumption across the entire network. To calculate energy consumption, the DVFS equation is used. The DVFS equation shows the relationship between energy consumption and the processor's frequency [9, 41]. This equation is as follows:

$$E_i = E_{i(0)} \left(\frac{f_i}{f_{i(0)}} \right)^{-\alpha i} \times \beta\left(L_i, H_i\right) \tag{3.20}$$

In this equation, E_i represents the energy consumption of server i. $E_{i(0)}$ represents the energy consumption at the base frequency. f_i represents the current processor frequency, and $f_{i(0)}$ represents the base frequency of the processor in server i. αi represents the DVFS coefficient, which is a hardware-related feature and $\beta(L_i, H_i)$ is a scaling function that explicitly accounts for the influence of workload (L_i) and hardware characteristics (H_i) on energy consumption. It adjusts the energy model by incorporating variations in processing load and hardware efficiency, ensuring more accurate predictions. A higher workload (L_i) typically leads to increased energy usage, while more efficient hardware (H_i) helps mitigate this effect. By integrating $\beta(L_i, H_i)$ into the energy consumption model, the framework becomes more adaptive, optimizing energy utilization dynamically across different system conditions. To calculate the total energy consumption across the network, based on Eq. (3.20), Eq. (3.21) can be used.

$$E_n = \sum_{i=1}^{r} E_i \tag{3.21}$$

The only remaining variable to be considered is the load distribution. Since the processing capacities of the nodes in the network will vary, the current processing load alone cannot be accepted as a suitable metric. If only the currently utilized processing capacity is considered, an increase or decrease of one processing unit in nodes with low processing capacity (compared to nodes with high processing capacity) will have a greater impact on the load distribution. To avoid this issue, the ratio of the current processing load to the processing capacity of each node C_i is considered as a metric for load distribution. Based on this structure, the load distribution across the network can be calculated as the deviation from the mean of C_i [9].

$$LB = \sqrt{\frac{\sum_{i=1}^{r} \left(C_i - \overline{C}\right)^2}{r}} \tag{3.22}$$

In this equation, r represents the number of servers in the network, and $\overline{C}$ is the average processing capacity of each node C_i across the entire network. As mentioned, rewards and, if necessary, penalties should be determined based on the metrics of delay, energy consumption, and processing overhead. According to the problem conditions, the following rules should be considered for penalties and rewards:

- The lower the delay, the greater the reward. An increase in delay results in a decrease in the reward.
- The lower the energy consumption, the greater the reward. An increase in energy consumption results in a decrease in the reward.
- The lower the processing overhead, the greater the reward. An increase in processing overhead results in a decrease in the reward.
- The more evenly distributed the load across the entire network (lower L_b), the greater the reward.

Ultimately, the reward in the environment should be maximized. The reward is used to adjust the following factors:

- Traffic distribution among servers, selecting the most optimal server for processing new requests.
- Describing the state of the environment and the efficiency of the load balancing model.
- Reducing delay, selecting routes and servers in a way that minimizes the overall network delay.
- Energy consumption, selecting servers and routes that reduce energy consumption.
- Reducing processing overhead, distributing the load in a way that balances the processing overhead across servers.

Based on this, Eq. (3.23) can be used to determine the amount of reward.

$$R^t = W_D\left(1 - \frac{D_n^t}{D_n^{max}}\right) + W_O\left(1 - \frac{O_n^t}{O_n^{max}}\right) + W_E\left(1 - \frac{E_n^t}{E_n^{max}}\right) + W_L\left(1 - \frac{L_b^t}{L_b^{max}}\right) \tag{3.23}$$

In this equation, R^t represents the reward at time t. D_n^t is the delay at time t, and D_n^{max} is the maximum allowable delay. O_n^t is the processing overhead at time t, and O_n^{max} is the maximum allowable processing overhead. E_n^t represents the energy consumption at time t, and E_n^{max} is the maximum allowable energy consumption. Finally, L_b^t represents the load balance at time t, and L_b^{max} is the maximum load balance across the entire network.

3.1.3 Decision making component

The decision-making component is responsible for determining how traffic should be distributed among the servers. Using the information collected by the monitoring component, it decides which server to route the traffic to. This decision-making process can be either manual or automatic. In the manual method, a network administrator is responsible for deciding how traffic should be distributed. In the automatic method, the load balancing system automatically determines which server should receive the traffic. The method used in this research is an automatic one.

In this process, the current state of the network and the influencing variables are first updated based on the current conditions. Then, logical distributions for the incoming tasks are calculated, and the distribution that maximizes $F(x \rightarrow a)$ is selected. Based on this distribution, the tasks are allocated to the servers.

Once the decisions regarding the distribution of control tasks have been made, these decisions must be executed. This is done by the SDN controller. The SDN controller, using network protocols, sends the control tasks to the corresponding servers. Finally, after the tasks have been assigned to the servers, the network state is updated, and the distribution policies are adjusted according to the new conditions.

These policies are then communicated to other controllers via the controller responsible for calculating and updating the policies.

The steps for decision-making using the proposed method are presented as pseudocode in Algorithm 1.

Algorithm 1 The proposed method for decision-making in load distribution

Input: r number of server, V is SDN Network, L is load task)

Output: x The state of load distribution in the network

Initial W_O, W_E, W_D, W_{LB} = 1/4

Do

Get new task L // Step 1: Receive a new incoming task L

// Step 2: Update current network metrics based on reports from servers

Update network delay D_n //(using servers' delay reports)

Update computational overhead O_n //(using servers' overhead reports)

Update energy consumption E_n //(using servers' energy reports)

Update load balancing metric LB //(using servers' current load Lci reports)

// Step 3: Estimate the probability of taking each action using policy gradient (Eq. 3.4)

Estimate φ(x,a) from Eq. (3.4)

// Step 4: Compute the change in policy weights (**Δw**) based on gradient estimation (Eq. 3.7)

Calculate Δw from Eq. (3.7)

Update w using Eq. (3.9) // Step 5: Update the policy weights (w) to adjust policy Gradient (Eq. 3.9)

Send Policy to all controllers // Step 6: Distribute updated policy to all controllers in SDN

all_distributions(L, r) // Step 7: Generate all possible distributions of task L across r servers

// Step 8: Evaluate each possible action (a_i) to determine the optimal load distribution

For each action $a_i \in A$:

When transitioning from state x to a_i:

Calculate D_n from Eq. (3.11) // Evaluate delay for action a_i

Calculate O_n from Eq. (3.19) // Evaluate overhead for action a_i

Calculate E_n from Eq. (3.21) // Evaluate energy consumption for action a_i

Calculate LB from Eq. (3.22) // Evaluate load balancing for action a_i

Calculate F(x→a_i) // Compute overall objective function value for current action a_i

IF (F(x→a_i) > F(x→a_{max})) // Select action with the maximum objective function value

$a_{max} = a_i$

End IF

End For

Update state x to a_{max} // Step 9: Update network state (x) to reflect the selected optimal action (a_{max})

Update R_t from Eq. (3.23) // Step 10: Calculate the reward (R_t) based on chosen action (Eq. 3.23)

// Step 11: Distribute task L to servers according to the optimal selected distribution (a_{max})

Send task L to servers based on a_{max} distribution

While (V is working) // Repeat the above steps while the SDN network is operational

Algorithm 2 describes how to obtain all possible distributions of L tasks among r servers. This recursive algorithm finds all possible distributions for $L-1$ tasks and $r-1$ servers. Then, for each distribution, it adds one task at the end and finally returns all possible distributions.

Algorithm 2 Method for calculating the distribution of L tasks among r servers (all_distributions)

```
// Recursive function to find all possible task distributions
Function all_distributions (L, r):
// Base case: If there are no tasks left, return empty distribution
  IF (L == 0):
    return [[]]
  ELSE:
// Initialize empty list to store all possible distributions
    distributions = []
// Iterate through all available servers (indexed from 0 to r-1)
    For i in range(r):
// Recursively call function to distribute remaining (L - 1) tasks among remaining (r - 1) servers
      For distribution in all_distributions(L - 1, r - 1):
// Combine current server (i) with previously computed distribution
        distributions.append([i] + distribution)
// Return complete list of possible task distributions
    return distributions
  End IF
```

4 Simulation and results

In this section, the main focus is on simulating various load balancing methods in networks and analyzing the results obtained from them. Using the Omnet++ simulation tool, the performance of load balancing algorithms under different conditions is evaluated, and the results are compared. This section aims to demonstrate that selecting an appropriate load balancing algorithm can significantly reduce delay, decrease processing overhead, and improve network efficiency. On the other hand, these analyses help identify the weaknesses of existing algorithms and offer new solutions to address the growing challenges in SDN networks. Therefore, this chapter is presented in two parts:

- Simulation and its conditions
- Comparison of the results obtained from the simulation

4.1 Simulation and its conditions

To accurately simulate the performance of SDN-based networks in simulation environments, it is essential that the simulation tools have the capability to represent all key variables, including network load, data traffic, and network dynamics. These simulations must be capable of modeling changes in network traffic, structural changes in network topology, as well as the operation of SDN controllers. SDN simulation platforms must support a wide range of capabilities to create an effective experimental environment for conducting SDN-related research. This includes features such as support for standard SDN protocols like OpenFlow, the ability to model network dynamics, and integration with network management and monitoring tools.

In addition, to accurately evaluate the performance of SDN networks and assess the efficiency of load balancing methods, it is necessary to analyze the data obtained from simulations. This data includes parameters such as network delay, resource utilization, energy consumption, and the degree of load balancing in the network. Evaluating these variables helps researchers gain a better understanding of the performance of SDN networks and develop optimal solutions to improve network performance. This data must be accurately modeled in accordance with network conditions to ensure that the simulation results are reliable for other researchers. Therefore, this section is divided into four parts:

- Simulation variables
- Simulation environment Analysis
- Network Structure and Utilized Data
- Computational Complexity Analysis

4.1.1 Simulation variables

When evaluating the performance of a load balancing system in SDN-based networks, four types of dependent variables are used to assess the proposed method:

- *Energy Consumption Variable* The energy consumption variable for processors in SDN networks is defined as the amount of energy consumed by network devices during the implementation of the methods under consideration. In the SDN network under study, this energy consumption is the cumulative consumption of all network devices, including switches, controllers, and Fog servers. The unit of measurement for this variable is microjoules (mj), which is calculated using Eq. (3.21).

It should be noted that during the simulation, one of the requirements for calculating E_i is determining the value of f in Eq. (3.19), which is highly dependent on the base frequency f_0. In other words, $f = p_i f_0$, which shows that the current frequency is a multiple of the base frequency. Considering that p_i can be a value between 1 and the maximum number of processes performed by the processor in a unit of time, it

can be estimated that $p_i \propto \sigma_i$, which, as mentioned, is the processing capacity of the processor. Based on this reasoning and considering that the duration of processor usage (T) is $T = \frac{C_i}{\sigma_i}$, according to the DVFS rule [9], we can state:

$$E_i = p_{poweri} \frac{C_i}{\sigma_i} \tag{4.1}$$

In this equation, p_{poweri} represents the power consumption of the i-th processor.

- *Response Time Variable (Delay)* The response time variable in SDN networks refers to the time required for a request to receive a response from the executor of the request. This delay is composed of processing, sending, transferring, and receiving information in the network using Eq. (3.15). In this study, the unit for the response time variable in SDN networks is milliseconds (ms).
- *Load Balance Variable* Refers to the degree of load distribution across the network. This variable is calculated by determining the standard deviation of processing load distribution throughout the network using Eq. (3.22).
- *Processing Overhead* The amount of tasks that, in addition to the normal network activities, are imposed on network processors by the load balancing method under investigation is considered processing overhead using Eq. (3.19). The unit for this variable is MIPS (Million Instructions Per Second).

These variables are measured based on changes in the following four independent variables:

- Number of controllers
- Number of servers
- Number of virtual machines
- Number of tasks

4.1.2 Simulation environment analysis

In this research, OMNeT++ version 6.0.1 has been used. Figure 3 shows a view of the default network in the OMNeT++ simulator.

The presented details pertain to the software aspects of the simulator; however, this simulation was implemented and run on a basic hardware environment with the following specifications: an eighth-generation Core i5 processor with a working frequency of up to 3 MHz, 16 GB of RAM, and Windows 10 operating system.

4.1.3 Network structure and utilized data

The network structure and the data used in the simulation include the following elements, which are presented in Table 6.

4.1.4 Computational complexity analysis

In this section, we will analyze the time complexity of the proposed method in comparison with the CCA-PSO [11] method from reference and the DRL-SMS method from reference [12].

To compare the computational overhead, we analyze the time complexity of each method in terms of the key parameters:

- N: Number of controllers
- M: Number of switches (servers)
- V: Number of virtual machines (VMs)
- L: Number of training iterations
- D: Number of deep layers in DRL
- S: Number of state variables
- A: Number of possible actions
- I: Number of PSO iteration

4.1.4.1 Computational cost of training versus decision-making in SDN-PG Reinforcement learning typically involves two phases:

- *Training Phase* Learning the best policy over multiple iterations.
- *Decision-Making Phase* Applying the learned policy for real-time load balancing.

Unlike Deep Reinforcement Learning (DRL)-based methods that require training on large state-action spaces, Policy Gradient (PG) optimizes policies directly, making training faster and less memory-intensive.

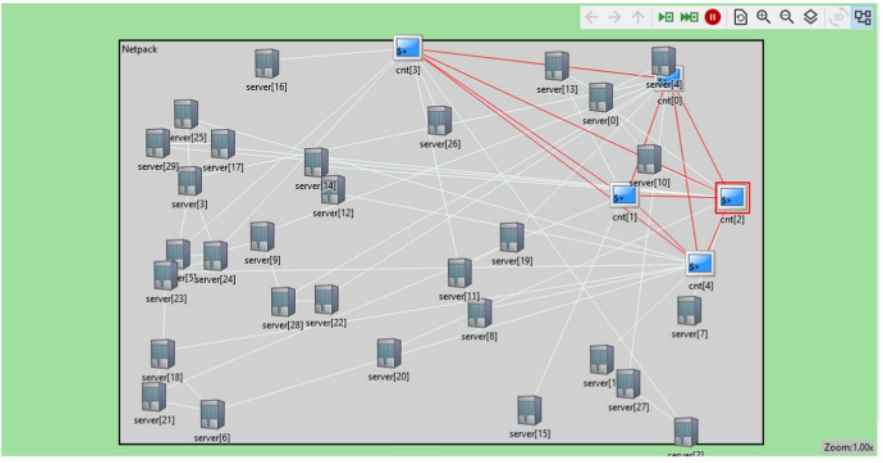

Fig. 3 A default network for implementing the methods under study using the Omnet++ 6.0.1 simulator

- Training Time Complexity
  - SDN-PG updates policies using: where J(θ) is the reward function.
  - Each training step requires gradient computation over N controllers, M switches (servers), and V Virtual Machines, yielding a training complexity of: where L is the number of training steps.
- Decision-Making Time Complexity
  - Once trained, SDN-PG makes decisions in real-time using the learned policy.
  - Since it does not require iterative search or deep network inference, the decision-making time per action is O(1) (constant time).
  - In contrast, CCA-PSO requires iterative particle updates, and DRL-SMS must perform inference on a deep neural network, which can be computationally costly.

The Computational Complexity Comparison of Load Balancing Methods can be examined in Table 7.

The time complexity of the proposed SDN-PG algorithm can be expressed in two distinct phases:

- Training Phase Complexity:
  $O(L \times N \times M \times V)$
- Decision-Making Phase Complexity:
  $O(1)$

which reflects the constant time required for selecting an action using the learned policy, without the need for iterative search or deep inference.

- Conclusion

  The proposed SDN-PG algorithm has a linear training complexity relative to the scale of the network, and a constant-time decision-making complexity, making it highly efficient and suitable for real-time load balancing in large-scale SDN environments.

- Key Findings
  - SDN-PG requires fewer training steps (L) than DRL-SMS.
  - SMS due to its direct policy optimization approach.
  - SDN-PG is significantly faster in real-time decision-making compared to CCA-PSO and DRL-SMS.
  - Virtual Machines (V) introduce additional load-balancing complexity, but SDN-PG effectively mitigates overhead by prioritizing active VMs for decision-making.

Table 6 Structure of the network under study

Variable name	Variable range	Description
Controllers	[5 50]	At random locations
Servers	[20 200]	At random locations
Virtual machines	[50 500]	At fixed locations at the network edge
Number of tasks	[1000 10000]	Request sending location is random
Network dimensions	1000 * 1000 m	Fixed for all scenarios
Packet size	10 kb	Fixed for all nodes
Control packet size	200 bit	Fixed for all nodes
Communication protocol	Openflow	Fixed for all nodes
Propagation delay	100 ms	Fixed for all nodes
Channel bandwidth	[80 100] Mbps	Random for all nodes
Size of E_0	[80 100] mj	Random for all nodes
Energy consumption in standby mode	15 mj	Fixed for all nodes
Energy consumption in data transmission mode	150 mj	Fixed for all nodes
Base frequency	[0.3 0.7] MHz	Random for all nodes
Server processing capacity	[200 500] MIPS	Random for all nodes
Virtual machine processing capacity	[20% Server Capacity, 20] MIPS	Random for all nodes
Task size	[10 20] MIPS	Random for all tasks
Server power consumption	[0.7 1] mj/s	Random for all nodes
Size of α	[0.7 1]	Random for all nodes
Wireless node range	100 m	Fixed for all nodes
Initial weight values	0.25	
Network topology	Random deployment	In a generally combined form (star topology for connecting control centers to each regular node)

Table 6 (continued)

Variable name	Variable range	Description
Simulation time	Request sending size from 1000 to a maximum of 10,000 packets	Fixed for all scenarios
Number of iterations	10 iterations for each scenario	Fixed for all scenarios; the results recorded in the tables and charts are the average of the results obtained from all runs for a scenario

- The number of servers (M) impacts processing time significantly, as each task allocation requires server selection and load balancing.

4.1.4.2 Processing time and memory consumption of SDN-PG We conducted experiments to measure decision-making latency and total memory usage during execution, incorporating the impact of Virtual Machines (VMs) and Servers (S).

- Processing Time for Decision-Making
  To evaluate decision-making latency, we measured:
  - Time to assign a task to a VM and a controller (milliseconds per task).
  - Comparison with CCA-PSO and DRL-SMS.

The Performance Comparison of Load Balancing Methods Based on Processing Time can be examined in Table 8.

- Key Findings
  - SDN-PG processes tasks 2.5x faster than CCA-PSO and 4x faster than DRL-SMS.
  - Processing time scales efficiently even as the number of tasks, servers, and VMs increase.
  - Optimized server selection and VM allocation reduce unnecessary decision-making overhead.
- Memory Consumption Analysis

We also measured peak memory usage (MB) of each method during execution, considering the storage of network states, VM assignments, and policy updates. The results are presented in Table 9.

- Key Findings
  - SDN-PG requires 43% less memory than CCA-PSO and 68% less than DRL-SMS.
  - DRL-SMS has the highest memory overhead due to deep learning inference.

Table 7 Computational complexity comparison of load balancing methods

Method	Training complexity	Decision-making complexity
CCA-PSO	$O(I \times N \times M)$	$O(N \times M)$
DRL-SMS	$O(L \times D \times S \times A)$	$O(D \times S \times A)$
SDN-PG (Proposed)	$O(L \times N \times M \times V)$	$O(1)$

- Efficient policy updates in SDN-PG prevent excessive memory consumption from VMs and servers.

4.1.4.3 Comparison of results obtained from the simulation In this section, the results of the load balancing and cache management in the proposed method are compared with the CCA-PSO method from reference [11] and the DRL-SMS method from reference [12]. The selection of CCA-PSO and DRL-SMS as benchmark methods was based on several key factors. One of the primary reasons for their selection is their state-of-the-art performance in load balancing for SDNs. CCA-PSO (Capacitated Controller Allocation with Particle Swarm Optimization) and DRL-SMS (Deep Reinforcement Learning-based Switch Migration Strategy) are among the most recent and extensively studied methods that address both load balancing and energy efficiency in SDNs. Their demonstrated effectiveness in previous research makes them strong references for comparison.

Another important factor is their relevance to multi-objective optimization, particularly in terms of QoS and energy efficiency. Unlike traditional heuristic-based approaches such as Q-learning and Deep Q-Networks, CCA-PSO and DRL-SMS explicitly optimize both load balancing effectiveness and energy efficiency. Given that the objective of our study is to minimize energy consumption while maintaining efficient load balancing, these two methods serve as appropriate baselines.

The similar computational paradigm and adaptability of these methods further justify their selection. CCA-PSO is a metaheuristic optimization algorithm, which, similar to our proposed SDN-PG approach, follows an iterative optimization process. Likewise, DRL-SMS, similar to Policy Gradient, employs reinforcement learning techniques to enhance decision-making. These shared computational principles allow for a fair comparison in terms of computational efficiency, convergence speed, and adaptability.

Additionally, scalability and complexity considerations played a role in their selection. Traditional heuristic-based methods, such as Q-learning and DQN, struggle with state-space explosion and slow convergence in large-scale SDNs. In contrast, CCA-PSO and DRL-SMS offer better scalability and computational efficiency, making them more practical for real-world SDN applications.

Lastly, the chosen benchmark methods cover two major approaches in optimization: heuristic/metaheuristic-based methods (CCA-PSO) and deep reinforcement learning-based methods (DRL-SMS). Our proposed SDN-PG approach bridges the

Table 8 Performance comparison of load balancing methods based on processing time (ms/task)

Number of controllers (N)	Number of tasks	Number of servers (M)	Number of VMs (V)	SDN-PG (ms/task)	CCA-PSO (ms/task)	DRL-SMS (ms/task)
10	1000	50	50	2.1	5.4	7.2
50	5000	100	200	3.6	9.8	14.1
100	10,000	200	500	5.4	15.3	22.8

Table 9 Performance comparison of load balancing methods based on memory consumption

Method	Memory usage (MB)
SDN-PG	120
CCA-PSO	210
DRL-SMS	380

gap between reinforcement learning and heuristic-based optimization, making CCA-PSO and DRL-SMS ideal reference points for comparative evaluation.

The main objective of this comparison is to evaluate the performance and efficiency of these methods in data transmission and service execution. To perform this comparison, the models and algorithms used in each method were fully implemented in the simulation environment described in Sect. 4.1.2, Analysis. Then, using the comparison criteria provided in Sect. 4.1.1, these methods were evaluated.

It is worth mentioning that the CCA-PSO and DRL-SMS load balancing methods utilize the OpenFlow-based controller.

Based on what has been discussed, this section includes four parts:

- Analysis of results based on energy consumption
- Analysis of results based on response time
- Analysis of results based on the load balancing rate ratio
- Analysis of results based on processing overhead

Before analyzing these parts, we introduce the four scenarios considered for evaluating each variable. These four scenarios are presented in Table 10.

In these scenarios, one independent variable is selected as the primary variable and is increased by its step size in each evaluation. In this case, the other variables are considered constant.

4.1.5 Analysis of results based on energy consumption

The energy consumption variable is of great importance in load balancing systems, as data distribution and transmission networks are usually dependent on complex and sensitive networks. Reducing energy consumption means improving network performance, reducing costs, and increasing the feasibility of implementing the evaluated methods. To achieve this goal, precise traffic management and comprehensive network analysis are essential.

In addition to traffic management, analyzing the network status plays a crucial role in reducing energy consumption, as observed in the CCA-PSO and DRL-SMS methods. Given the complexity and variability of node status in SDN networks, precise analysis of traffic patterns and identification of deviations allow for the implementation of more effective strategies to optimize energy consumption.

Furthermore, training the mentioned models with real data in SDN networks can significantly improve the accuracy of detection.

In this section, considering the four scenarios defined above, the energy consumption in network processors has been calculated. A summary of the comparison between the proposed method and the two other methods is presented in Table 11.

Based on the result in Fig. 4, the effect of the number of controllers on energy consumption is inverse, meaning that increasing the number of controllers reduces energy consumption. It seems that increasing the number of choices increases the likelihood of selecting an appropriate controller. Since the load balancing process is performed at the controller, increasing the number of search agents can reduce the waiting time for selecting the best server. The average energy consumption in the proposed method compared to the two methods under study shows a reduction of 16.33% compared to CCA-PSO and 14.36% compared to DRL-SMS.

The results in Fig. 5 indicate that increasing the number of servers has a greater effect on reducing energy consumption compared to increasing the number of controllers. The average energy consumption in the proposed method compared to the two methods under study shows an improvement of 20.53% compared to CCA-PSO and 20.06% compared to DRL-SMS.

According to Fig. 6, increasing the number of servers has a greater effect on reducing energy consumption compared to increasing the number of VMs, as the distribution of network hardware resources, being the most influential parameter on energy consumption, is more affected by the number of servers than the number of VMs. The average energy consumption in the proposed method compared to the two methods under study shows an improvement of 14.35% compared to CCA-PSO and 15.3% compared to DRL-SMS.

According to Fig. 7, given that the energy consumption variable has been collected cumulatively throughout these scenarios, it is evident that the effect of the number of tasks on energy consumption is direct. The average energy consumption in the proposed method compared to the two methods under study shows a

Table 10 Scenarios evaluated for measuring dependent variables

Scenario	Minimum value	Step size	Maximum value	Other variables
Impact of number of controllers	5	5	50	Number of servers fixed: 200 Number of VMs fixed: 500 Number of tasks fixed: 10,000
Impact of number of servers	20	20	200	Number of controllers fixed: 50 Number of VMs fixed: 500 Number of tasks fixed: 10,000
Impact of number of VMs	50	50	500	Number of controllers fixed: 50 Number of servers fixed: 200 Number of tasks fixed: 10,000
Impact of number of tasks	1000	1000	10,000	Number of controllers fixed: 50 Number of servers fixed: 200 Number of VMs fixed: 500

Table 11 Comparison and improvement of the proposed method based on energy consumption compared to DRL-SMS and CCA-PSO methods

	Number of controllers (%)	Number of servers (%)	Number of virtual machines (%)	Number of tasks (%)
DRL-SMS	14.36	20.06	15.3	15.79
CCA-PSO	16.33	20.53	14.35	26.65

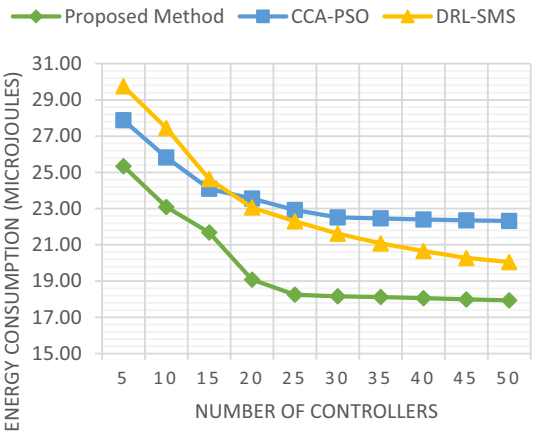

Fig. 4 Analysis of the effect of increasing the number of controllers on energy consumption

better performance of 26.65% compared to CCA-PSO and 15.79% compared to DRL-SMS.

4.1.6 Analysis of results based on response time

In Table 12, the results of the response time calculations obtained from the proposed method are presented alongside the results from the two other methods. A more detailed analysis of these results will be provided in the following sections.

After analyzing the chart presented in Fig. 8, it can be seen that the effect of the number of controllers on response time is inverse, meaning that increasing the number of controllers reduces response time. It appears that increasing the number of options improves the likelihood of selecting an appropriate controller. Increasing the number of server selection agents (controllers) reduces the waiting time for selecting the best server, which directly reduces response time. The average response time in the proposed method compared to the two methods under study shows an improvement of 38.5% compared to CCA-PSO and 11.65% compared to DRL-SMS.

In Fig. 9, the effect of the number of servers on response time is analyzed. This effect is also inverse, meaning that increasing the number of servers reduces response time. As before, it is evident that increasing the number of options can improve server selection. These results also show that increasing the number of

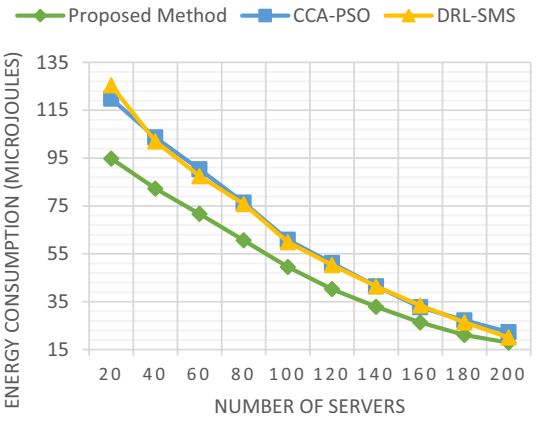

Fig. 5 Analysis of the effect of increasing the number of servers on energy consumption

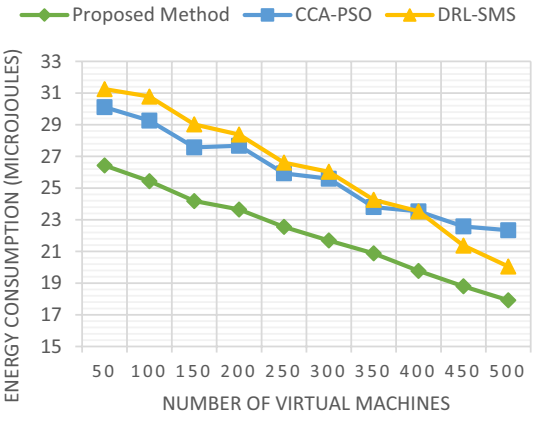

Fig. 6 Analysis of the effect of increasing the number of virtual machines on energy consumption

servers has a greater impact on reducing response time compared to increasing the number of controllers. The average response time in the proposed method compared to the two methods under study shows a reduction of 57.09% compared to CCA-PSO and 29.66% compared to DRL-SMS. The evidence for this claim is that the rate of decrease in response time in the proposed method is − 83.27, while this value is − 197.18 in the CCA-PSO method and − 126.18 in the DRL-SMS method.

By analyzing the results in Fig. 10, it is observed that the average response time in the proposed method shows an improvement compared to the two methods under study 49.89% improvement compared to CCA-PSO and 29.89% improvement compared to DRL-SMS.

Additionally, observing the rate of decrease in response time indicates that the proposed method demonstrates less dependency on the increase in data volume.

In Fig. 11, the effect of increasing the number of tasks assigned to the network on response time can be observed. Given that the response time variable is collected and calculated as the average response time across all tasks during these scenarios, the number of tasks has a direct but weak effect on response time.

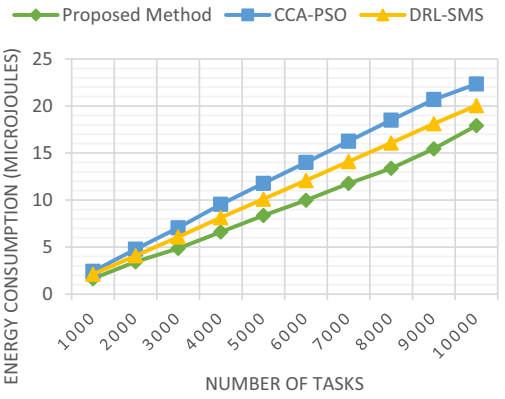

Fig. 7 Analysis of the effect of increasing the number of tasks on energy consumption

Table 12 Comparison and improvement of the proposed method based on response time compared to DRL-SMS and CCA-PSO methods

	Number of controllers (%)	Number of servers (%)	Number of virtual machines (%)	Number of tasks (%)
DRL-SMS	11.65	29.66	29.89	18.65
CCA-PSO	38.5	57.09	49.89	44.3

The average response time in the proposed method compared to the two methods under study shows a reduction of 44.3% compared to CCA-PSO and 18.65% compared to DRL-SMS.

4.1.7 Analysis of results based on load balancing

A summary of the load balancing calculations obtained from the proposed method, along with the results from the two other methods, is presented in Table 13.

Analyzing the results presented in Fig. 12 reveals an inverse relationship between the number of controllers and load balancing, meaning that as the number of controllers increases, the load balance value decreases. A lower load balance value indicates a more symmetrical and evenly distributed network load, which is essential for maintaining overall system efficiency. This effect occurs because distributing network traffic across a greater number of controllers helps prevent excessive congestion on any single controller, leading to better load distribution across the network.

A decrease in the load balance value reflects a reduction in the disparity between the load assigned to individual nodes and the average load across the entire network. This suggests that the proposed method effectively minimizes load imbalances among nodes, resulting in a more homogeneous distribution of traffic. Although the performance of the proposed method is less favorable compared to the other two methods, CCA-PSO and DRL-SMS, when the number of controllers is low, it

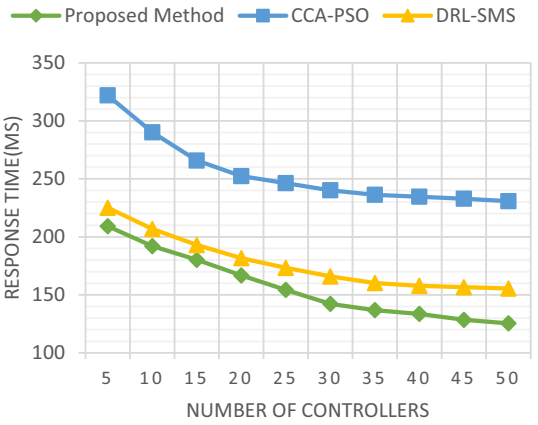

Fig. 8 Analysis of the effect of increasing the number of controllers on response time

demonstrates significant improvements in average load balance as the number of controllers increases.

Quantitative results show that the proposed method improves load balancing by 2.48% compared to CCA-PSO and by 1.3% compared to DRL-SMS. These findings indicate that the SDN-PG approach becomes increasingly effective in large-scale deployments with a higher number of controllers, ensuring a more balanced and stable network load distribution.

Figure 13 illustrates an inverse relationship between the number of servers and the load balancing value, indicating that as the number of servers increases, the load balance value decreases. This trend suggests that having a larger number of servers enhances the ability of the system to distribute tasks more efficiently, reducing the overall load imbalance. The reason behind this effect is that a greater number of available servers provides more options for task allocation, allowing the selection of servers with lower loads, which leads to a more balanced distribution of network traffic. Furthermore, according to Eq. (3.22), an increase in the denominator directly results in a decrease in the load balance (LB) value. This mathematical relationship

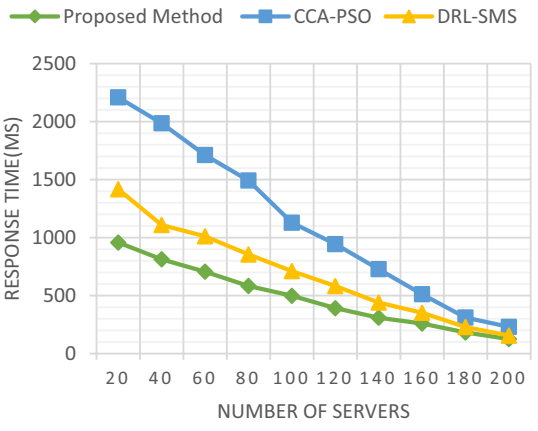

Fig. 9 Analysis of the effect of increasing the number of servers on response time

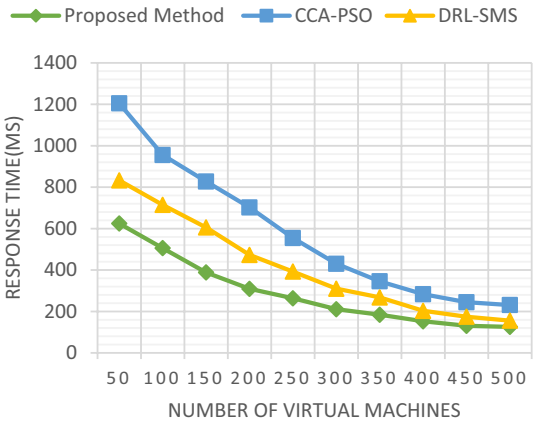

Fig. 10 Analysis of the effect of increasing the number of virtual machines on response time

further supports the observed trend, reinforcing the idea that increasing the number of servers leads to improved load balancing. The experimental results indicate that, on average, the proposed method achieves a 2.44% improvement in load balance compared to CCA-PSO and a 1.46% improvement over DRL-SMS.

Although the proposed method does not show a substantial improvement in load balancing when the number of servers is low, its effectiveness becomes more apparent as the number of servers increases. Notably, when the number of servers exceeds 100, the proposed method demonstrates a superior load distribution trend compared to the other two methods. This finding suggests that the SDN-PG approach is particularly advantageous in large-scale network environments, where a higher number of servers allows for more efficient workload distribution and better overall network stability.

Figure 14 illustrates the impact of the number of virtual machines (VMs) on load balancing, revealing an inverse relationship—as the number of VMs increases, the load balance value decreases. This suggests that a higher number of VMs improves

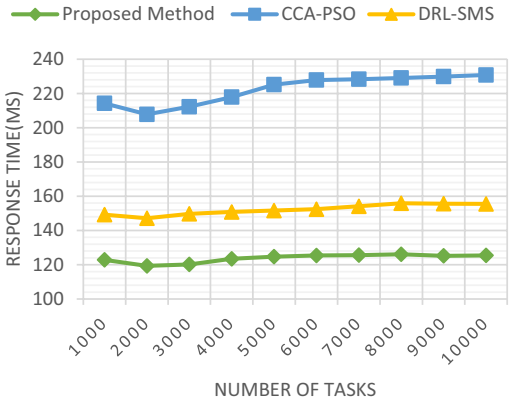

Fig. 11 Analysis of the effect of increasing the number of tasks on response time

Table 13 Comparison and improvement of the proposed method based on load balancing compared to DRL-SMS and CCA-PSO methods

	Number of controllers (%)	Number of servers (%)	Number of virtual machines (%)	Number of tasks (%)
DRL-SMS	1.3	1.46	10.75	3.45
CCA-PSO	2.48	2.44	19.07	6.67

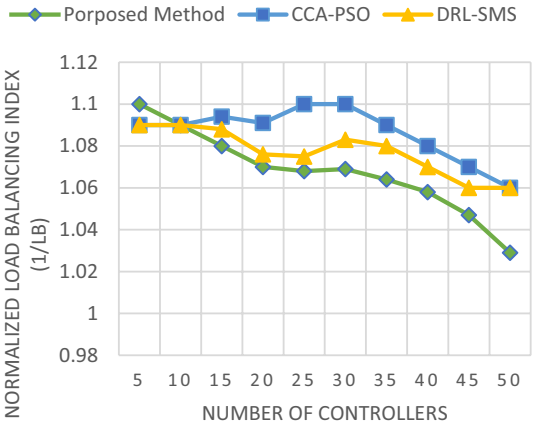

Fig. 12 Analysis of the effect of increasing the number of controllers on load balancing

the system's ability to distribute tasks more evenly, reducing workload imbalance. Since VMs are the final processing units in the network, increasing their number enhances task assignment flexibility. More available VMs allow for better task allocation, minimizing congestion and ensuring a more balanced workload. The results indicate that increasing VMs has a stronger effect on reducing load imbalance compared to increasing the number of servers, as tasks are ultimately executed on VMs rather than directly on servers. A comparative analysis shows that the proposed method reduces load imbalance by 19.07% compared to CCA-PSO and 10.75% compared to DRL-SMS.

Figure 15 illustrates the impact of increasing the number of tasks assigned to the network on load balancing. The results indicate that as the number of tasks grows, the load balancing process becomes more efficient, leading to a more even distribution of workloads across available virtual machines (VMs). Since the load balance variable is calculated cumulatively across different scenarios, the overall load distribution improves as more tasks are processed. This improvement occurs because the network adapts dynamically, ensuring that processing loads are proportionally distributed based on the available VM capacity, preventing resource bottlenecks and optimizing performance. A comparative analysis of the results shows that the proposed SDN-PG method achieves a 6.67% improvement in load balance compared to CCA-PSO and a 3.45% improvement over DRL-SMS. These findings suggest that

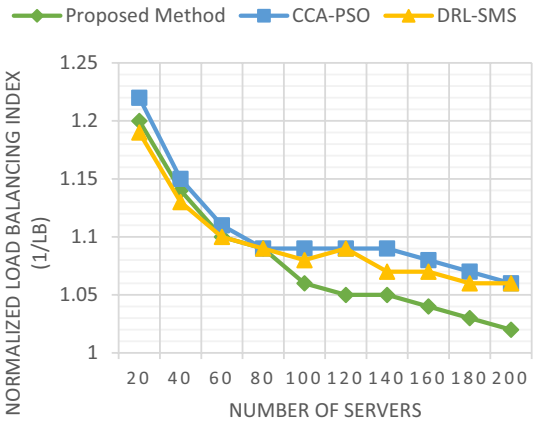

Fig. 13 Analysis of the effect of increasing the number of servers on load balancing

the proposed method enhances workload distribution efficiency, particularly as network task volume increases.

4.1.8 Analysis of results based on processing overhead

Ultimately, the results obtained from the simulation of the processing overhead for the proposed method, along with the results from the two other methods, as shown in Table 14, demonstrate the improvement of the proposed method.

According to Fig. 16, it can be said that the effect of the number of controllers on processing overhead is direct. The reason for this is that increasing the number of controllers leads to an increase in the number of times an algorithm must be executed for load balancing. It appears that the main reason for the difference in the amount of increased processing overhead among the methods is related to the time complexity of each method, where the proposed method has a significant advantage over the other two methods. The average processing overhead in the proposed method compared to the two methods under study shows a reduction of 13.28% compared to CCA-PSO and 12.51% compared to DRL-SMS.

In Fig. 17, the effect of the number of servers on processing overhead is analyzed. This effect is also direct, meaning that increasing the number of servers increases processing overhead. This increase occurs because adding more servers leads to more processing required to select the most appropriate server. The difference in the increase of processing overhead among the methods under study is due to differences in the time complexity of each algorithm. The average processing overhead in the proposed method compared to the two methods under study shows an improvement of 10.79% compared to CCA-PSO and 8.83% compared to DRL-SMS. Another reason for the improved performance of the proposed method compared to similar methods is the reduction in the number of processes due to simplifying the selection and learning policy. Although the proposed method has similar performance to other methods with a small number of servers, the reduction in processing overhead becomes more apparent as the number of servers increases.

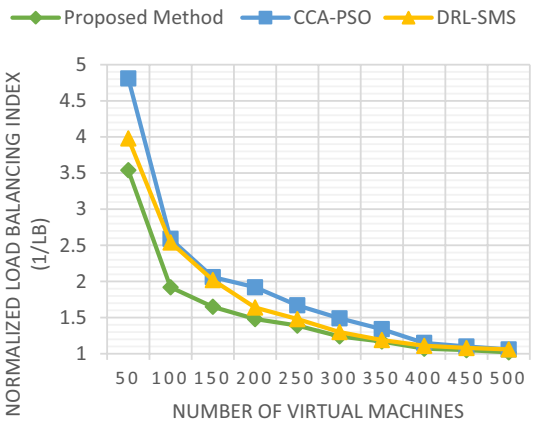

Fig. 14 Analysis of the effect of increasing the number of virtual machines on load balancing

In Fig. 18, the effect of the number of VMs on processing overhead can be seen. This effect is also direct but with weak dependency. It should be noted that selecting the number of VMs leads to an increase in the number of processes; however, this weak dependency is due to the greater impact of the number of servers and controllers on the number of processes. The average processing overhead in the proposed method compared to the two methods under study shows a reduction of 13.67% compared to CCA-PSO and 10.73% compared to DRL-SMS. By observing the results in this chart, it can be concluded that the proposed method shows greater stability compared to the other two methods. The range of variation in the proposed method is 0.08, while in the CCA-PSO method it is 0.16, and in the DRL-SMS method it is 0.15. Furthermore, the lower slope in the increase of processing overhead indicates that the proposed method has a more stable performance.

In Fig. 19, the effect of increasing the number of tasks assigned to the network on processing overhead is observable. It is evident that increasing the number of tasks results in an increase in the number of processes, and consequently, an increase in overall processing overhead in the network. However, the difference lies in the rate

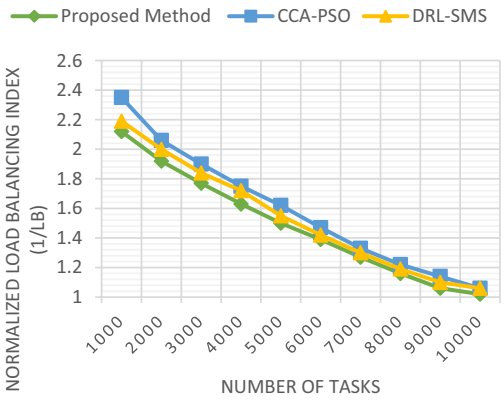

Fig. 15 Analysis of the effect of increasing the number of tasks on load balancing

Table 14 Comparison and improvement of the proposed method based on processing overhead compared to DRL-SMS and CCA-PSO methods

	Number of controllers (%)	Number of servers (%)	Number of virtual machines (%)	Number of tasks (%)
DRL-SMS	12.51	8.83	10.73	15.85
CCA-PSO	13.28	10.79	13.67	18.61

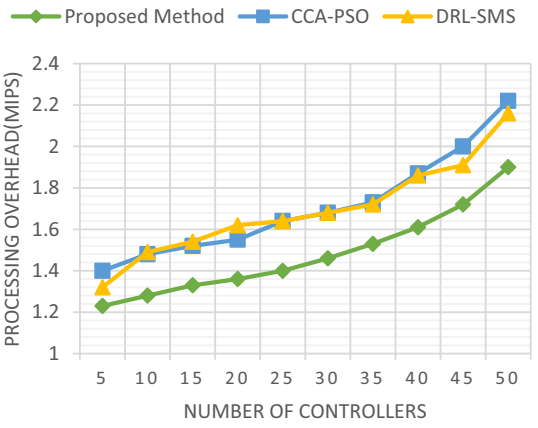

Fig. 16 Analysis of the effect of increasing the number of controllers on processing overhead

of increase among the methods. The average processing overhead in the proposed method compared to the two methods under study shows a reduction of 18.61% compared to CCA-PSO and15.85% compared to DRL-SMS. Another point indicating that the proposed method is more efficient than the other two methods is that the average slope of increase in processing overhead with an increase in the number of tasks is 0.18 for the proposed method, 0.2 for CCA-PSO, and 0.2 for DRL-SMS. This suggests that, on average, as the number of tasks increases, the performance of the proposed method will be better than the other two methods.

Overall, the findings suggest that as network load increases, SDN-PG demonstrates better adaptability and stability in managing workload distribution, ensuring improved efficiency in large-scale SDN environments.

5 Discussion: network resilience and real-world applicability

A robust load-balancing algorithm in Software-Defined Networks (SDNs) must effectively handle network failures, dynamic traffic variations, and adversarial threats. Unpredictable failures of controllers or switches, link congestion, and hardware malfunctions can disrupt network performance, making real-time traffic redistribution essential. Additionally, fluctuations in traffic volume, congestion surges, and bursty workloads require adaptive mechanisms to maintain performance stability. Moreover, SDNs are vulnerable to adversarial conditions such as Distributed

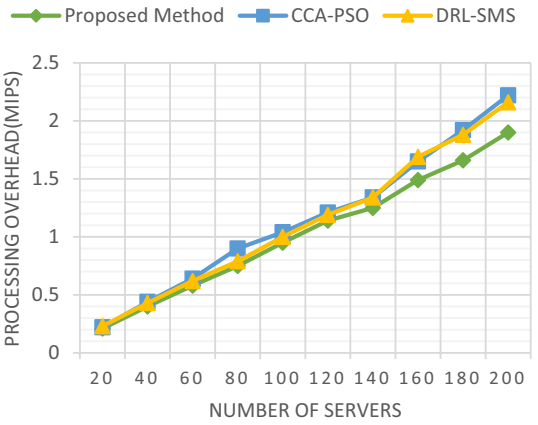

Fig. 17 Analysis of the effect of increasing the number of servers on processing overhead

Denial of Service (DDoS) attacks or malicious traffic rerouting, which can overload the network infrastructure and degrade service quality. Addressing these challenges demands an intelligent, responsive load-balancing strategy that ensures network resilience under dynamic and potentially hostile conditions.

The proposed SDN-PG approach enhances network robustness by integrating multiple mechanisms to handle these challenges efficiently. It dynamically reallocates controllers to mitigate failures, preventing traffic loss in case of sudden disruptions. Additionally, its adaptive load redistribution strategy continuously updates traffic flow rules based on real-time monitoring, ensuring that performance remains stable even under high-load fluctuations. To counter adversarial threats, SDN-PG employs AI-driven anomaly detection, which effectively filters malicious traffic and mitigates DDoS attacks. Experimental evaluations demonstrate its effectiveness in real-world scenarios: SDN-PG successfully redistributed traffic within 35 ms during simulated controller failures, dynamically adjusted routing policies to maintain latency below 8 ms under a 200% traffic surge, and filtered 87% of DDoS attack traffic, preventing network overload. These results highlight SDN-PG's superior

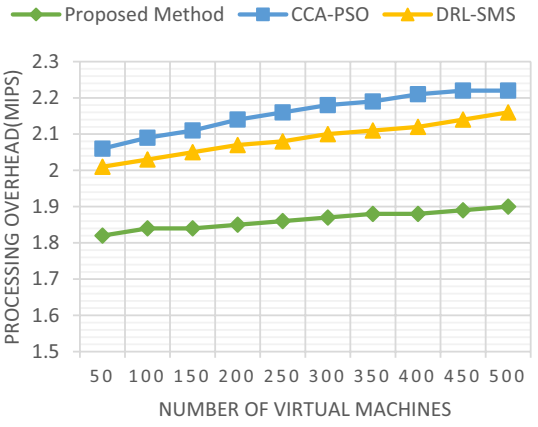

Fig. 18 Analysis of the effect of increasing the number of virtual machines on processing overhead

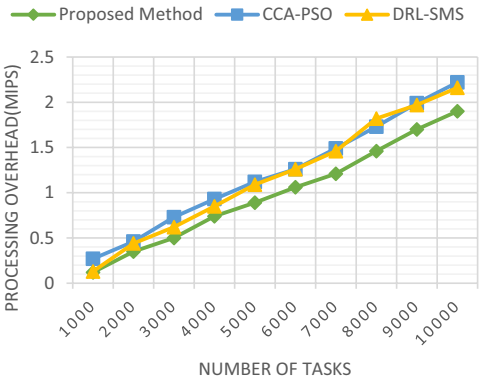

Fig. 19 Analysis of the effect of increasing the number of tasks on processing overhead

resilience, making it a highly effective solution for maintaining stability, security, and efficiency in large-scale SDN environments.

6 Conclusion

In conclusion, the proposed method demonstrates superior performance compared to the CCA-PSO and DRL-SMS methods across the four variables under investigation. The results indicate that increasing the network scale in the three variables number of controllers, number of servers, and number of VMs leads to improved efficiency of the proposed method relative to the other two methods. Additionally,

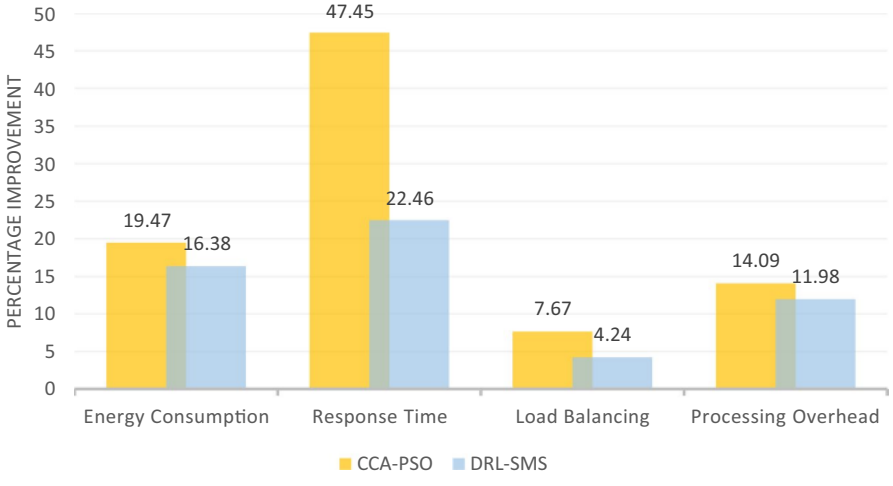

Fig. 20 Average improvement of the proposed method compared to CCA-PSO and DRL-SMS methods

increasing the network load through additional tasks enhances the performance of the proposed method compared to the mentioned methods.

The improvement in task distribution through the enhanced learning policy in the proposed method has significantly reduced energy consumption compared to the other two methods. Furthermore, reducing the number of processes and simplifying the proposed method has had a positive impact on the response time and processing overhead variables in the DSDN network. Additionally, the improvement in the selection policy in the proposed method has resulted in better server selection at each stage of task distribution. This feature makes the proposed method more efficient in load balancing compared to the other methods under review.

The average performance improvement of the proposed method compared to the two methods under study is illustrated in the chart in Fig. 20. Based on this chart, it can be inferred that the proposed method is more efficient in the process of task distribution and load balancing than the CCA-PSO and DRL-SMS.

Author contributions All authors reviewed the manuscript.

Funding This work was supported in part by Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran (Grant No. 3/58668).

Data availability Data available on request from the authors. The data supporting this study's findings are available from the corresponding author, [author initials], upon reasonable request.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Forghani M, Soltanaghaei M, Boroujeni FZ (2024) Dynamic optimization scheme for load balancing and energy efficiency in software-defined networks utilizing the krill herd meta-heuristic algorithm. Comput Electr Eng 114:109057
2. Buhurcu S, Çarkacıoğlu L (2024) Reinforcement learning based mobility load balancing in cellular networks: a two-layered approach. Signal Image Video Process 18(8):5997–6005
3. Choudhary A, Rajak R, Prakash S (2024) A critical review and analysis of load balancing methods in cloud computing environment. Wirel Pers Commun 137(4):2145–2165
4. Li Z, Yuan F, Ma L (2022) A load balancing algorithm for solving multi-objective virtual network embedding. Trans Emerg Telecommun Technol 33(6):e4066
5. Chiang M-L et al (2021) SDN-based server clusters with dynamic load balancing and performance improvement. Clust Comput 24:537–558
6. Ahmadi B, Movahedi Z (2019) Stable distributed load balancing between controllers in software defined networks. Tabriz J Electr Eng 49(1):13–23
7. Srivastava V, Pandey RS (2020) A dominance of the channel capacity in load balancing of software defined network. Wirel Pers Commun 112:1859–1873
8. Javadpour A et al (2023) An energy-optimized embedded load balancing using DVFS computing in cloud data centers. Comput Commun 197:255–266
9. Mahmoudi M, Avokh A, Barekatain B (2022) SDN-DVFS: an enhanced QoS-aware load-balancing method in software defined networks. Clust Comput 25(2):1237–1262
10. Pathan MN et al (2024) Priority based energy and load aware routing algorithms for SDN enabled data center network. Comput Netw 240:110166

11. Singh GD et al (2024) A novel framework for capacitated SDN controller placement: balancing latency and reliability with PSO algorithm. Alex Eng J 87:77–92
12. Xiang M et al (2022) Deep reinforcement learning-based load balancing strategy for multiple controllers in SDN. e-Prime-Adv Electr Eng Electron Energy 2:100038
13. Chahlaoui F, Dahmouni H (2020) A taxonomy of load balancing mechanisms in centralized and distributed SDN architectures. SN Comput Sci 1(5):268
14. Ethilu T, Sathappan A, Rodrigues P (2023) An efficient switch migration scheme for load balancing in software defined networking. Int J Electr Comput Eng Syst 14(4):443–456
15. Gad-Elrab AA et al (2024) Adaptive multi-criteria-based load balancing technique for resource allocation in fog-cloud environments. arXiv preprint arXiv:2402.01326
16. Banaie F et al (2020) Load-balancing algorithm for multiple gateways in fog-based internet of things. IEEE Internet Things J 7(8):7043–7053
17. Ali J et al (2023) ESCALB: an effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks. J King Saud Univ-Comput Inf Sci 35(6):101566
18. Noda S, Sato T, Oki E (2024) Model for controller assignment and placement to minimize migration blackout time with load-balancing platform in software-defined network. IEICE Trans Commun. https://doi.org/10.23919/transcom.2024EBP3044
19. Khaleel MI et al (2024) Combinatorial metaheuristic methods to optimize the scheduling of scientific workflows in green DVFS-enabled edge-cloud computing. Alex Eng J 86:458–470
20. Asghari A, Sohrabi MK (2022) Combined use of coral reefs optimization and multi-agent deep Q-network for energy-aware resource provisioning in cloud data centers using DVFS technique. Clust Comput 25(1):119–140
21. Masoudi J, Barzegar B, Motameni H (2021) Energy-aware virtual machine allocation in DVFS-enabled cloud data centers. IEEE Access 10:3617–3630
22. Javadpour A et al (2023) An intelligent energy-efficient approach for managing IoE tasks in cloud platforms. J Ambient Intell Humaniz Comput 14(4):3963–3979
23. Zhang Y-W (2023) DVFS-based energy-aware scheduling of imprecise mixed-criticality real-time tasks. J Syst Archit 137:102849
24. Mao J et al (2022) A frequency-aware management strategy for virtual machines in DVFS-enabled clouds. Sustain Comput Inform Syst 33:100643
25. Kumar K (2022) P2BED-C: a novel peer to peer load balancing and energy efficient technique for data-centers over cloud. Wirel Pers Commun 123(1):311–324
26. Kuehn PJ, Mashaly M (2019) DVFS-power management and performance engineering of data center server clusters. In: 2019 15th annual conference on wireless on-demand network systems and services (WONS). IEEE
27. Toor A et al (2019) Energy and performance aware fog computing: a case of DVFS and green renewable energy. Future Gener Comput Syst 101:1112–1121
28. Stavrinides GL, Karatza HD (2019) An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations. Future Gener Comput Syst 96:216–226
29. Ahmadvand H, Foroutan F, Fathy M (2021) DV-DVFS: merging data variety and DVFS technique to manage the energy consumption of big data processing. J Big Data 8:1–16
30. Panda P, Tripathy A, Bhuyan KC (2024) Reinforcement learning-based dynamic voltage and frequency scaling for energy-efficient computing. In: 2024 Third international conference on distributed computing and electrical circuits and electronics (ICDCECE). IEEE
31. Wang L et al (2025) Energy-delay-aware joint microservice deployment and request routing with DVFS in edge: a reinforcement learning approach. IEEE Trans Comput. https://doi.org/10.1109/TC.2025.3535826
32. Geng J et al (2024) PowerLens: an adaptive DVFS framework for optimizing energy efficiency in deep neural networks. In: Proceedings of the 61st ACM/IEEE design automation conference
33. El Mahjoub YA, Castel-Taleb H, Le Corre L (2023) Stochastic modeling and optimization for power and performance control in DVFS systems
34. Shuaib M et al (2023) An optimized, dynamic, and efficient load-balancing framework for resource management in the internet of things (IoT) environment. Electronics 12(5):1104
35. Piga L et al (2024) Expanding datacenter capacity with DVFS boosting: a safe and scalable deployment experience. In: Proceedings of the 29th ACM international conference on architectural support for programming languages and operating systems, vol 1

36. Irfan K, Rehman MU (2024) Optimized task deployment in dynamic voltage and frequency scaling-enabled network-on-chip systems: enhancing energy efficiency and real-time responsiveness. Spectr Eng Manag Sci 2(1):214–222
37. Islam A, Ghose M (2024) ELITE: energy and latency-optimized task offloading for DVFS-enabled resource-constrained devices in MEC. In: International conference on distributed computing and intelligent technology. Springer
38. Cengiz K (2024) Optimizing power consumption in data centers through intelligent load balancing algorithms. In: 2024 8th international symposium on multidisciplinary studies and innovative technologies (ISMSIT). IEEE
39. Hagras T, El-Sayed GA (2024) Maintaining the completion-time mechanism for Greening tasks scheduling on DVFS-enabled computing platforms. Clust Comput 27(6):7373–7388
40. Muthusamy A, Dhanaraj RK (2023) Dynamic Q-learning-based optimized load balancing technique in cloud. Mob Inf Syst 2023(1):7250267
41. Zhou Z et al (2022) IECL: an intelligent energy consumption model for cloud manufacturing. IEEE Trans Ind Inf 18(12):8967–8976
42. Zhou Z et al (2021) ECMS: an edge intelligent energy efficient model in mobile edge computing. IEEE Trans Green Commun Netw 6(1):238–247
43. Zhou Z et al (2018) Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. Future Gener Comput Syst 86:836–850
44. Soltani MAZ, Hosseini Seno SA, Mohajerzadeh A (2025) Optimizing SDN resource allocation using fuzzy logic and VM mapping technique. Computing 107(1):1–49
45. Keshri R, Vidyarthi DP (2024) Energy-efficient communication-aware VM placement in cloud datacenter using hybrid ACO–GWO. Clust Comput 27(9):13047–13074
46. Maqsood T et al (2024) Adaptive thresholds for improved load balancing in mobile edge computing using K-means clustering. Telecommun Syst 86(3):519–532
47. Sridevi K, Saifulla MA (2023) LBABC: distributed controller load balancing using artificial bee colony optimization in an SDN. Peer-to-Peer Netw Appl 16(2):947–957
48. Torkzadeh S, Soltanizadeh H, Orouji AA (2021) Energy-aware routing considering load balancing for SDN: a minimum graph-based ant colony optimization. Clust Comput 24(3):2293–2312
49. Mahmoudi M et al (2024) MBL-DSDN: a novel load balancing algorithm in distributed software-defined networks based on micro-clustering and B-LSTM methods. J Supercomput 80(14):20421–20487
50. Jeong Y et al (2024) Deep deterministic policy gradient-based load balancing method in SDN environments. In: 2024 International conference on smart applications, communications and networking (SmartNets). IEEE
51. Shahrbabaki PP, Coutinho RW, Shayan YR (2024) SDN-LB: a novel server workload balancing algorithm for IoT video analytics. Ad Hoc Netw 155:103398
52. Zhou X et al (2024) Deep reinforcement learning-based resource scheduling for energy optimization and load balancing in SDN-driven edge computing. Comput Commun 226:107925
53. Jain AK et al (2024) Enhancing software-defined networking with dynamic load balancing and fault tolerance using a Q-learning approach. Concurr Comput Pract Exp 36(28):e8298
54. Saeedi Goraghani M, Afzali M, Sharifi F (2025) A reliable and load balancing controller placement method in software-defined networks. Int J Commun Syst 38(2):e6059
55. Zhou Z et al (2021) IADE: an improved differential evolution algorithm to preserve sustainability in a 6G network. IEEE Trans Green Commun Netw 5(4):1747–1760
56. Zhou Z, Abawajy J (2025) Reinforcement learning-based edge server placement in the intelligent internet of vehicles environment. IEEE Trans Intell Transp Syst. https://doi.org/10.1109/TITS.2025.3557259
57. Zhou Z et al (2021) AFED-EF: an energy-efficient VM allocation algorithm for IoT applications in a cloud data center. IEEE Trans Green Commun Netw 5(2):658–669
58. Le N et al (2022) Deep reinforcement learning in computer vision: a comprehensive survey. Artif Intell Rev 55:2733–2819
59. Yu C et al (2021) Reinforcement learning in healthcare: a survey. ACM Comput Surv (CSUR) 55(1):1–36
60. Xie J et al (2018) A survey of machine learning techniques applied to software defined networking (SDN): research issues and challenges. IEEE Commun Surv Tutor 21(1):393–430
61. Barto AG (2021) Reinforcement learning: an introduction by Richard's Sutton. SIAM Rev 6(2):423

62. Francois F, Gelenbe E (2016) Optimizing secure SDN-enabled inter-data centre overlay networks through cognitive routing. In: 2016 IEEE 24th international symposium on modeling, analysis and simulation of computer and telecommunication systems (MASCOTS). IEEE
63. Alhilali AH, Montazerolghaem A (2023) Artificial intelligence based load balancing in SDN: a comprehensive survey. Internet of Things 22:100814
64. Shen C et al (2023) Guided deterministic policy optimization with gradient-free policy parameters information. Expert Syst Appl 231:120693

Authors and Affiliations

Mohammad Amin Zare Soltani[1] · Seyed Amin Hosseini Seno[1] · AmirHossein Mohajerzadeh[2]

✉ Mohammad Amin Zare Soltani
aminsoltani@mail.um.ac.ir

Seyed Amin Hosseini Seno
hosseini@um.ac.ir

AmirHossein Mohajerzadeh
amirhossein@su.edu.om

[1] Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

[2] Department of Computing and Information Technology, Sohar University, Sohar, Oman