Cluster Computing (2025)28:660
https://doi.org/10.1007/s10586-025-05331-y

Controller placement in software-defined networks using reinforcement learning and metaheuristics

Mohammad Sadegh Sirjani[1] · Ali Maleki[2] · Amir Pakmehr[3] · Maedeh Abedini Bagha[4,5] · Ali Ghaffari[5,6,7] · Ali Asghar Pour Haji Kazem[6]

Received: 16 November 2024 / Revised: 12 March 2025 / Accepted: 25 April 2025

Abstract
Software-defined networking (SDN) has revolutionized network management by enabling dynamic control and optimization of network resources. A key challenge in SDN deployment is the strategic placement and assignment of controllers, which significantly affects network performance in terms of energy consumption, latency, and load balancing. This research addresses the controller placement problem by proposing a two-step method that combines enhanced reinforcement learning with an improved metaheuristic algorithm, termed Bedbug-GLA. In the first step, an irregular cellular learning automata model is developed to determine the optimal number of controllers required. In the second step, the Bedbug metaheuristic algorithm is employed to efficiently assign controllers to switches. Simulation results demonstrate that Bedbug-GLA achieves up to an 18% improvement in maximum controller load, a 69% reduction in congested controller overload, and a 20% decrease in energy consumption compared to state-of-the-art metaheuristic approaches, as evaluated on standard network topologies derived from real-world datasets.

Keywords Bedbug metaheuristic algorithm · Controller placement problem · Energy consumption · Software-defined network

1 Introduction

The software-defined networking (SDN) is a new promising technology that has provoked interest in academic and industrial research [1]. SDNs and conventional networks differ in the way they operate. SDNs decouple the data plane, which handles the forwarding of network traffic, from the control plane, which manages network policies and configurations [2]. All control functions are performed by a specific entity named the controller [3]. Controllers are the brain of SDNs, and the software-based controller logically centralizes the network intelligence [4]. Controllers are the operating system of SDNs. They oversee network management by offering services that enable user applications to communicate with the network's hardware and connect with other devices within the network [5]. While controllers maintain an overview of the entire network with the help of information gathered on network status, they also set the rules of the flow table of switches [6]. The network switches, those known as nodes in SDNs [7], are solely responsible for forwarding the data, whereas

✉ Maedeh Abedini Bagha
maide.abedini@gmail.com

1 Department of Computer Science, University of Texas at San Antonio, San Antonio, TX, USA

2 Department of Computer Engineering, Shiraz BranchDepartment of Computer Engineering, Shi. C., Islamic Azad University, Shiraz, Iran

3 Department of Computer and Information Technology Engineering, Qa. C., Islamic Azad University, Qazvin, Iran

4 Roshdiyeh Higher Education Institute, Tabriz, East Azerbaijan Province, Iran

5 Department of Computer Engineering, Ta.C., Islamic Azad University, Tabriz, Iran

6 Department of Software Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Türkiye

7 Department of Computer Science, Khazar University, Baku, Azerbaijan

Published online: 03 September 2025

controllers specify the routing of network packets among the switches [8]. The data plane switches forward flows based on the switch flow table [9]. As soon as a new flow arrives that is unknown to the switch, the switch sends a flow set-up request to the controller. The controller by receiving flow set-up requests from switches, responds by sending rules to be installed in the switch's flow table [10].

From an architecture perspective, the control plane may be composed of either one controller, in which case it is called the single-controller architecture, or multiple controllers, in which case it is called the multi-controller architecture. The single-controller architecture suffers the single failure point, so the multi-controller architecture is usually recommended. The multi-controller architecture is itself divided into flat architecture and hierarchical architecture. In the former, all controllers are placed in the same layer, but in the latter, the controllers are placed in different layers [11].

The placement and quantity of controllers in static networks are predetermined and remain constant. However, in dynamic networks, controllers must adjust rapidly to changes in network traffic, with new controllers being activated as network load increases and some controllers being deactivated as load decreases [12]. The placement of controllers is critical as it can affect the network's performance, reliability, availability, and ability to handle traffic spikes. The Controller Placement Problem (CPP) refers to determining the appropriate number of controllers (NC) to deploy in the network, identifying the optimal locations for their installation, and assigning switches to these controllers in order to meet specific performance requirements such as delay, energy usage [13], and load balancing [14].

Every switch in the SDN must be controlled by at least one controller. Assigning appropriate switches to a specific controller is called controller assignment [15], which is an important issue in the CPP. The optimal solution for CPP depends on various factors [16], such as propagation delay [17], latency [18], load balancing [19], bandwidth, energy usage [20], security issues [2], and data transfer rate. These factors are often conflicting in nature, including reliability, load balancing, latency, and energy efficiency [21]. Finding an appropriate trade-off among these metrics is essential for effective controller placement, as a single optimal or random placement may not be feasible [22].

Due to environmental and economic concerns, grid energy usage and heat and carbon dioxide produced for energy production cannot be ignored. Hence, it is crucial to design an energy-aware SDN topology [23].

The significance of delay in controller placement within SDNs cannot be ignored, as it directly impacts the network's responsiveness, efficiency, and overall performance. A well-optimized controller placement strategy plays a pivotal role in minimizing communication delays between network components, ensuring timely decision-making, and enhancing the network's reliability. By strategically situating controllers closer to critical network nodes, delays can be significantly reduced, leading to improved network agility and robustness [24].

Another important parameter in SDNs is load balancing [25]. Load balancing strategies become important when the network is congested. If switches are assigned to the closest controller, congestion will occur and some controllers will be overloaded while others are underutilized. An overload controller reduces responsiveness and degrades performance as flows experience an unexpected delay. The consequences of overloading the controllers and creating an imbalance in the utilization of network resources include network congestion and packet loss [26].

The CPP in SDNs is a critical optimization challenge that aims to minimize propagation latency and ensure efficient network performance. Initially introduced to address latency [24], the CPP has evolved to incorporate multiple objectives, reflecting the complexity of modern network deployments. This problem is known to be NP-hard, and its complexity increases significantly when multiple objectives are involved, demanding substantial computational resources to find efficient solutions [27].

Recent trends in SDN emphasize the need for efficient and scalable networks, which are challenged by factors such as increasing latency, load balancing issues, and energy consumption [28]. The shift towards multi-objective optimization is crucial as it addresses the diverse requirements of modern SDNs, offering a more holistic approach to solving the CPP [29]. However, comprehensive solutions that simultaneously consider all critical parameters remain scarce, underscoring the need for innovative multi-objective approaches that can effectively navigate the trade-offs between competing objectives in SDN deployments.

This paper addresses the multi-objective CPP using reinforcement learning and metaheuristic algorithms, which are well-suited for complex optimization problems. A hybrid approach is proposed, integrating the Bedbug Meta-Heuristic Algorithm (BMHA) [30], Genetic Algorithm, and reinforcement algorithm, known as Bedbug-GLA. Bedbug-GLA employs Irregular Cellular Learning Automata (ICLA) [31], enhanced by integrating a reinforcement signal derived from the Genetic Algorithm (GA), resulting in Developed ICLA (DCLA), which is used to determine the necessary number of controllers. Additionally, Bedbug-GLA utilizes the Bedbug Metaheuristic Algorithm (BMHA) and further refines it into Developed BMHA (DBMHA) by incorporating genetic operators to enhance both the speed and accuracy of the algorithm. The DBMHA is then applied for controller assignment in SDN. Theoretically, Bedbug-GLA offers superior adaptability

and flexibility compared to traditional methods, such as sub modularity-based approaches, which are effective in IoT networks but may lack adaptability in broader scenarios. Unlike ILP-based dynamic placement methods, which can be computationally intensive, Bedbug-GLA provides a lightweight and adaptive solution. The proposed method stands out due to its comprehensive approach, addressing three critical parameters: delay, energy consumption, and load balancing. The advancements made in ICLA and BMHA significantly enhance the relevance of this study. Furthermore, Bedbug-GLA is versatile and can be implemented across various multi-controller SDN networks, offering a flexible and efficient framework for optimizing controller placement. The significant contributions of the paper are as follows:

- Bedbug-GLA develops ICLA by GA and presents new developed ICLA. ICLA is the reinforcement learning algorithm that consists of a grid or network of interconnected cells (automata). Each cell possesses a set of possible actions it can take based on its current state and external stimuli. When a cell interacts with its environment, it receives feedback based on its actions. This feedback guides the cell in adapting its behavior by adjusting its state or action selection probabilities. However, the learning time and adaptation of ICLA within specific applications and problem domains are crucial factors. DCLA employs the GA to accelerate the learning process.
- Bedbug-GLA Uses the proposed DCLA to find appropriate NC. Bedbug-GLA employs DCLA to identify the most suitable NCs tailored to the network's size, structure, traffic patterns, and controller workloads, aiming to enhance the Quality of service (QoS) in SDN.
- Bedbug-GLA enhances the BMHA to accelerate the convergence rate and prevent local optima entrapment. While metaheuristic algorithms are proficient at quickly identifying near-optimal solutions, they often face challenges such as getting stuck in local optima and experiencing premature convergence. To address these issues, Bedbug-GLA introduces DBMHA, a hybrid approach that combines BMHA with genetic operators to enhance convergence speed and avoid local optima. Additionally, DBMHA incorporates chaotic theory for generating the initial population, which improves population diversity.
- Bedbug-GLA employs the proposed DBMHA for controller assignment, designed to enhance energy efficiency, manage controller loads, and minimize propagation latency.
- Experimental simulation is applied on three different real-world topologies to prove Bedbug-GLA.

The rest of the paper is written as follows: An overview of the works related to CPP is presented in Sect. 2. Section 3 is dedicated to the modeling of system and formulation of the problem, as well as an explanation of the Bedbug-GLA method proposed for solving the CPP. Section 4 evaluates Bedbug-GLA. The final section is devoted to conclusions and suggestions for future works.

2 Related works

The CPP in SDNs has been addressed through both single-objective and multi-objective optimization approaches [32]. Single-objective methods typically utilize algorithms to minimize or maximize one objective, often focusing on latency reduction using metaheuristic techniques. In contrast, multi-objective approaches employ algorithms that consider two or more objectives, such as latency, reliability, load balancing, and energy consumption. These methods may also incorporate machine learning or metaheuristic techniques. The evolution of CPP from focusing solely on latency minimization to incorporating multiple objectives reflects the growing complexity of modern network deployments, where efficiency, reliability, and scalability are increasingly important.

2.1 Single-objective approaches

The CPP has been normalized as a single-objective optimization problem in some research. Heller et al. [24], introduced CPP in 2012. They studied it as a K-median or K-center problem and their aim was propagation latency. They evaluated all potential locations for the controller placement with a comprehensive approach. In [28], CPP was investigated to reduce energy usage. The authors modeled it as a binary integer program (BIP). The network energy usage was reduced due to the delay of the control paths and the load of the controllers in it. Also, a GA was designed to find a suboptimal solution for an extensive network. However, they didn't find NC in the findings. Wang et al. [33], developed an optimized K-means algorithm for CPP to decrease SDN latency. Liu et al. [34], presented a form of PSO algorithm for CPP while considering the latency and load balance of the controllers. Li et al. [35], designed a different method based on PSO by dynamic parameters. They improved the switch to controller average transmission latency. Authors in [36], provided guidelines for network operators based on the SRGM framework to manage the NC and increase service reliability and quality in SDN. Ateya et al. [32], developed a dynamic optimization algorithm based on the Chaotic Salp Swarm Algorithm (CSSA). The CSSA dynamically evaluates the optimum NC and the optimum controller

assignments, which minimizes the deployment cost and latency. Authors in [37], show that distributing the tasks between homogeneous cores in controllers can reduce the frequency of operations and the overall energy usage. In [38], a clustering strategy was used for the fair allocation of switches. The authors used the analytical network process (ANP) and a multi-criterion decision-making (MCDM) plan for network clustering and CA. Their goal was to improve end-to-end delay and controller-to-controller latency. But, they did not consider load balancing. Babbar and Rani [39], introduced a capacity-based switch-splitting technique to partition the network into sub-networks, each managed by a strategically positioned controller. This method employs a personalized and adaptive learning approach alongside density-based splitting to effectively segment the network. Each sub-network is assigned a controller, optimizing both the placement and the NCs needed for efficient network management. However, this approach also increases complexity. Also, none of these works didn't consider energy usage parameters in solving the CPP.

Single-objective approaches to the CPP focus on optimizing specific metrics, such as latency or energy efficiency. While methods like K-median, binary integer programming, K-means, and PSO have achieved targeted improvements, they often neglect critical factors like load balancing, network capacity, or energy usage. These limitations highlight the narrow focus of single-objective methods, making them less suitable for addressing the complex demands of dynamic network environments.

2.2 Multi-objectives approaches

The approaches discussed above consider a single criterion for solving the CPP. However, multiple parameters must be considered to solve the CPP in the real world, and some works solve the problem by multi-objective algorithms.

In [40], the authors proposed an analytical model considering propagation latency. They used two models, Single Data Ownership (SDO) and Multiple Data Ownership (MDO), to demonstrate compatibility. They proposed two models for CPP that would reduce the response time in SDO and MDO models. They also presented an evolutionary algorithm to find Pareto's optimal locations in propagation latency. Still, they did not consider energy usage and NC. Ramya et al. [41], pursued three goals: identifying the required NC, locating the controllers optimally, and ensuring the network's reliability. They determined the NC using a graph theory approach. The Tabu-Pareto Integrated Search (PITS) algorithm was used to identify the optimal position of the controllers. A heuristic approach and controller migration managed link failure and controller load imbalance. Killi et al. [42], proposed a scalable algorithm in large-scale networks to improve the controller's load balancing. To increase the load balance of the controllers, this algorithm uses the fractional distribution of switches among controllers in the form of poly-stable matching. The remaining switches are assigned to the closest controllers, taking into account the switches' latency and the controllers' load. The algorithm also migrates some switches from their controllers to reduce propagation latency whereas they do not care about energy usage. Kazemian and Mirabi [43], employed Ant Lion Optimizer (ALO) to develop an effective solution. They took into account propagation latency and link reliability by utilizing disjoint paths between switches and controllers. Additionally, they identified the appropriate NC and demonstrated strong performance in terms of latency; however, they did not address the energy parameter. Guo et al. [15], formulated and solved the CPP for low-Earth orbit (LEO) satellite networks using static placement with the dynamic assignment (SPDA) method. The method has two steps. It first combines SDN controllers in some fixed satellites by formulating a mixed integer programming (MIP). Then, it defines dynamic controller assignment to improve the latency and load of the controllers. However, it does not consider energy usage. Naseri et al. [44], propose a solution to the CPP by considering both setup costs and latency of control packets. They develop several models, including Cost Model, Cost & Controller-to-Switch Latency Model (CCSLM), Cost & Controller-to-Controller Latency Model (CCCLM), Cost & Controller-to-Controller Hops Model (CCCHM), and Cost & Controller-to-Switch Hops Model (CCSHM), to balance parameters like controller number, latency, and hops. The solution uses a binary linear programming model, solved with CPLEX or MATLAB, employing exact optimization techniques. This multi-objective approach optimizes both cost and performance metrics simultaneously. However, solving NP-Hard problems like CPP exactly can be computationally challenging. To enhance delay and load balancing in SDNs, Abedini Bagha et al. [25] first partitioned the network using the Fuzzy C-Means (FCM) clustering algorithm. They then identified optimal locations for controllers through the Seagull Optimization Algorithm (SOA). In another study [23], they proposed ELA-RCP to improve energy efficiency, load balancing, and reliability in SDNs. They determined the appropriate NCs using cellular learning automata and subsequently identified optimal controller locations through an improved version of SOA. Additionally, they monitored the load on the controllers and enhanced network load balancing and reliability using a heuristic algorithm. However, this approach did not lead to any improvements in delay. Hemagowri et al. [45], developed a new method called Demming Regressive Multi-objectives Dragonfly Optimized Controller

Placement (DRMDOCP) to determine the optimal NC in different network topologies, minimize average latency, and increase the reliability and throughput of SDN by solving CPP. Li et al. [46] developed a policy for controller placement and synchronization aimed at minimizing network costs while enhancing overall network performance through deep reinforcement learning (DRL). Their approach seeks to balance multiple objectives, including reducing network expenses, improving synchronization among distributed controllers, and ensuring efficient communication with network devices. However, the study did not determine the optimal NCs. Additionally, DRL may present challenges such as high complexity and computational costs, difficulties in generalizing to new environments, scalability limitations due to resource constraints, and extensive data requirements for training. Singh et al. [11] addressed the Challenge of the CPP, taking into account both communication latency and reliable data transmission. Their main objective was to reduce the overall average latency of the network by implementing a robust network model capable of functioning seamlessly even in the event of failure of n-1 controllers out of n deployed. Employing PSO, they successfully optimized controller allocation, thereby reducing network latency and improving switch assignments within this dynamic system. Through experiments on the Internet2 OS3E architecture, it was observed that three controllers maintained high reliability and low latency performance even under controller failures. The results demonstrate the effectiveness of the proposed methodology, paving the way for exploring additional multi-objective optimization techniques to address reliability concerns within the CPP more effectively in future research. But it didn't determine the NC. However, none of these works don't consider energy parameter in problem solving. Khojand et al. [12], proposed GEWO for the Controller Placement Problem (CPP), combining game theory with a hybrid optimization algorithm that integrates the Golden Eagle Optimization (GEO) and the Grey Wolf Optimizer (GWO) to optimize controller placement for efficient network management. GEWO improves load balancing, end-to-end delay, and energy consumption. However, it has added complexity from game theory, potential limitations in scalability for large networks, and a dependency on algorithm parameters that require tuning for optimal performance.

Multi-objective approaches for solving the CPP address the need for balancing multiple performance metrics, such as latency, energy consumption, and load balancing, to achieve optimal network management. Various studies have proposed innovative algorithms, including evolutionary techniques and hybrid optimization methods, to enhance controller placement while considering trade-offs among conflicting objectives. However, many of these approaches often neglect critical parameters like energy usage or face challenges related to algorithm complexity and computational demands. Overall, while multi-objective methods provide a more holistic view of network optimization, they must also address these limitations to improve their applicability in real-world scenarios.

Table 1 provides a comprehensive comparison of methodologies, evaluation parameters, and the advantages and disadvantages of each method. As shown in Table 1, research on the CPP in SDN can be broadly categorized into single-objective and multi-objective approaches. Single-objective methods focus on optimizing specific metrics like latency or energy, offering efficient solutions for particular network conditions but potentially neglecting other critical performance aspects. In contrast, multi-objective approaches, such as GEWO, DRMDOCP, and CCSHM, aim to balance multiple conflicting objectives simultaneously, including network delay, energy, load balancing, and cost. These methods provide a more comprehensive optimization by considering trade-offs between different performance metrics, offering robust and adaptable solutions for complex network environments. However, they often require more complex algorithms and computational resources due to the increased dimensionality of the optimization problem. The choice between single-objective and multi-objective methods depends on the specific network requirements and constraints.

Notably, most previous studies have overlooked the importance of incorporating Number of controllers (NC) into their problem-solving approaches and have often neglected the energy parameter. Despite these advancements, comprehensive solutions that simultaneously consider all critical parameters remain scarce, highlighting the need for innovative multi-objective approaches that can effectively navigate the trade-offs between competing objectives in SDN deployments.

This work introduces Bedbug-GLA as a novel approach to solving CPP. Bedbug-GLA calculates the proper number of controllers using a new proposed DCLA and assigns controllers to switches using a new proposed DBMHA, while considering load balancing, propagation latency, and energy usage parameters.

3 System modeling and proposed method

3.1 System modeling

To address the problem mathematically, the CPP is modeled as a graph $G = (V, E)$, where V denotes the set of switches and E represents the links between these switches. Specifically, V is defined as the set of switches $V = \{s_1, s_2, \ldots, s_n\}$, with $|V| = n$ indicating the

Table 1 A summary of studies

References	Object	Type of method	Method	Performance Metrics	Advantages	Weakness/Gaps
[24]	Single-objective	Machine learning	K-median	-Latency	- Improve latency	- Not specifying the NC - High memory and time consumed -Not caring about energy usage -Consider only one object (latency)
[28]	Single-objective	Metaheuristic	GA	- Energy	-Improve energy usage	- Not specifying the NC -Consider only one object (Energy)
[33]	Single-objective	Machine learning	CNPA	-Latency	- Improve latency	-Not caring about energy usage -Consider only one object (latency)
[34]	Single-objective	Metaheuristic	NCPSO	-Latency	- Optimize Controller's load	-Not caring about energy usage -Consider only one object (Load)
[32]	Single-objective	Metaheuristic	CSSA	-Latency	-Improve latency -Improve deployment cost	- Not specifying the NC -Not caring about energy usage -Consider only one object (latency)
[38]	Single-objective	Machine learning	Clustering MCDM	-Latency	- Improve inter-controller latency - Improve end-to-end delay	- Not specifying the NC -Not caring about energy usage -Not caring about load balancing -Consider only one object (latency)
[39]	Single-objective	Machine learning and heuristic	PUAL-DBSCP	-Latency	-Improve latency -Determine the NC	-Not caring about energy usage -Increased complexity -Consider only one object (latency)
[49]	Multi-objective	Metaheuristic	POCO	-Latency, -Reliability, -Load balancing	- Improve latency - Considered to be failure cases	- Not specifying the NC - No implementation in large and real network -Not caring about energy usage
[40]	Multi-objective	Metaheuristic	Exa-Place	-Latency, -Reliability, -Response time	- Improve latency	-Choosing the master controller of a switch as the closet controller -Not caring about energy usage
[41]	Multi-objective	Metaheuristic	graph-theory & PITS	-Latency, -Reliability, -Load balancing	- Determine the NC -Determine the optimal location of controllers	-Not caring about energy usage
[42]	Multi-objective	Machine learning	Poly-stable matching	-Latency, -Load balancing	- Improve load balancing - Improve latency -Determine the NC -Improve scalability	-Not caring about energy usage
[43]	Multi-objective	Metaheuristic	Multi-objective antlion algorithm	-Latency - Reliability	-Determine the NC - Assign switch to controllers - Improve latency	-Not caring about energy usage

Table 1 (continued)

References	Object	Type of method	Method	Performance Metrics	Advantages	Weakness/Gaps
[15]	Multi-objective	Heuristic	SPDA	-Load balancing, -Latency	- Improve propagation latency - Improve load balancing	- Not specifying the NC -Not caring about energy usage
[44]	Multi-objective	Exact optimization techniques	CCSHM	-Cost -Latency	-Determine the NC -improve latency -improve cost	-High computational complexity
[25]	Multi-objective	Metaheuristic	SOA	-Load balancing, -Delay	- Improve end-to-end delay - Improve load balancing	- Not specifying the NC -Not caring about energy usage
[23]	Multi-objective	Metaheuristic-machine learning	ELA-RCP	-Reliability, -Energy -Load balancing,	- Improve load balancing -Determine the NC -Improve reliability - Improve energy consumption	- No improvement in delay
[45]	Multi-objective	Metaheuristic	DRMDOCP	-Latency, -Throughput, -Load balancing, -Packet drop rate	- Improve load balancing - Improve latency -Determine the NC -Improve fault tolerance - Improve throughput	-Not caring about energy usage
[46]	Multi-objective	Machine learning	DRL	-Delay -Energy -Cost	-Improve Controller utulization - Improve latency -Improve energy consumption	- High complexity - Scalability limitations - Extensive data requirements
[11]	Multi-objective	Metaheuristic	PSO	-Latency - Reliability	- Improve latency - Assign switch to controllers -Improve reliability	- Not specifying the NC -Not caring about energy usage
[12]	Multi-objective	Metaheuristic-machine learning	GEWO	-Delay -Energy -Load balancing,	- Determine the NC - Assign switch to controllers -Improve energy consumption - Improve load balancing	-High complexity -Scalability

total number of switches. Additionally, C is the set of controllers, expressed as $C = \{c_1, c_2, \ldots, c_m\}$, where m represents the number of controllers such that $|C| = m$.

Based on the given assumptions, the quantity of controllers does not exceed the quantity of switches ($m \leq n$). Each controller is capable of handling requests up to its specified capacity. Additionally, the total number of requests managed by each controller at any given time must remain within its processing limits. A request cannot be split among different controllers for processing [10]. In this model, each switch can be linked to only a single controller. Dynamic models are used, with all network traffic being dynamic and unpredictable [47].

$$y_{s,c} = \begin{cases} 1 \; if \; node \; s \in V \; is \; controlled \; by \; the \; controller \; c \in C \\ 0 \; Otherwise \end{cases} \tag{1}$$

$$\sum_{c=1}^{c} y_{s,c} = 1 \tag{2}$$

$$L_c \leq U_c \forall c \in C \tag{3}$$

Equation (1) shows whether or not node $s \in V$ is controlled by controller $c \in C$. Equation (2) mandates that each switch is assigned to precisely one controller. $y_{s,c}$ has a binary value and when switch s is assigned to controller c, it will be equal to 1; otherwise, 0. Equation (3) shows that the load of each controller (Lc) cannot exceed its processing capacity, denoted by Uc for controller c.

Considering that the end-to-end delay, the energy usage, and the controllers' loads are the parameters of this research, equations are defined to measure them.

3.1.1 End-to-end delay

In networking contexts, the end-to-end delay refers to the average time it takes for data to travel from a specific source to a specific destination. In SDN, this delay is composed of several components.

The transmission delay is the time it takes to transmit data over a link, which depends on the size of the data packet and the bandwidth of the link. The propagation latency is the time it takes for a signal to travel through a medium, such as a fiber optic cable or wireless link, determined by the distance between the source and destination and the speed of signal propagation in the medium.

Additionally, processing delay occurs when packets are processed by network devices like routers or switches. In SDN, this includes the time taken by the controller to process flow rules and update switch configurations. Queuing delay is the time packets spend waiting in buffers at network devices before being transmitted, which depends on network congestion and buffer size.

The mathematical formulation of the end-to-end delay, as shown in Eq. (4), includes four components: transmission delay (d_{trans}), propagation latency (d_{prop}), processing delay (d_{proc}), and queuing delay (d_{que}).

$$d_{end2end} = d_{prop} + d_{trans} + d_{proc} + d_{que} \tag{4}$$

3.1.2 Propagation latency

There are two types of control paths: Switch-to-controller path and inter-controller path. Latency in SDNs is a key criterion that affects the performance of the systems. When there are many controllers in the SDN, the frequent exchange of packets between controllers, switches, and controllers affects the end-to-end delay, as well as QoS. The propagation latency is the sum of inter-controller latency and switch-to-controller latency. In networking, propagation latency is defined as the time it takes to transmit data from one node to another, which can be calculated by Eq. (5).

$$d_{prop} = C2C_{avg} + S2C_{avg} \tag{5}$$

where $d\,(s,\,c)$ is the shortest possible route from switch s to controller c in the network. The shortest possible route between switches and controllers is calculated using the haversine distance [48]. Equation (6) shows the average propagation latency of the controller set [10, 24]:

$$d_{prop}avg(C) = \frac{1}{n}\sum_{s} \min_{c \in C} d(s,c) \times y_{s,c} \tag{6}$$

The average inter-controller latency in the SDN can be determined by Eq. (7) [49]:

$$C2C_{Avg} = \frac{1}{m(m-1)} \sum_{c_i.c_j \in C} d\left(c_i.c_j\right) \tag{7}$$

As before said, $d\left(c_i.c_j\right)$ is the shortest possible route from controller c_i to c_j.

The calculation of the latency of switch-to-controller propagation is necessary. Equation (8) presents the average latency of switch-to-controller propagation in SDNs [48].

$$S2C_{Avg} = \frac{1}{|S|}\sum_{s}\sum_{c} d(s,c) \times y_{s,c} \tag{8}$$

In this paper, the controllers' queues are modeled as an *M/G/1* queue system. Whenever a new flow arrives at a switch, the switch sends a flow adjustment request message to its SDN controller. Investigations carried out in the field of network traffic [50] have shown that the flow arrival process in packet-switching networks follows the Poisson distribution [51].

Controller c is responsible for S active switches. If the flow arrival process from switch s ($s \in S$) is a Poisson flow with parameter λ_s independent of other switches and λ_c is the rate of incoming messages to controller c, then the sum of incoming packet messages from this controller's assigned switches is λ_c with Poisson distribution in Eq. (9).

$$\lambda_c = \sum_{s \in S} \lambda_s \times y_{s,c} \tag{9}$$

μ_c is the service rate and ρ_c is the traffic intensity of messages received in controller c, which are calculated by Eq. (10).

$$\rho_c = \left(\frac{\lambda_c}{\mu_c}\right) \times y_{s,c} \tag{10}$$

The processing time of the input flows in the controller corresponds to the normal distribution with the position parameter μ_c/l and the scale parameter σ_c. The average queue length of the messages received in the controller can be calculated from Eq. (11) where Ql_c shows the queue length of controller c. Also, the average waiting time of requests in the controller is calculated by Eq. (12) [52].

$$Ql_c = \rho c + \frac{\rho_c^2 + \lambda_c^2 \sigma_c^2}{2(1-\rho_c)} \tag{11}$$

$$d_{queu} = \frac{Qlc}{\lambda_c} = \frac{1}{\mu_c} + \frac{\rho_c^2 + \lambda_c^2 \sigma_c^2}{2\lambda_c(1-\rho_c)} \tag{12}$$

3.1.3 Controllers load

The load on a controller in a SDN is primarily driven by the handling of PACKET_IN messages, which are sent by switches when a packet does not match any entry in the switch's flow table. This typically occurs with new flows, prompting the controller to determine the forwarding path and install new flow rules, allowing future packets to be handled locally. The frequency and volume of these messages significantly impact the controller's workload, affecting network performance and scalability. The load is quantified using Eq. (13), which calculates the load on controller c, denoted as L_c, as the sum of loads generated by all connected switches s, represented by l_s [49]. The average load across all controllers is then found by summing their individual loads and dividing by the total number of controllers, as shown in Eq. (14) [23].

$$L_c = \sum_{s=1}^{n} y_{s.c} \times l_s \tag{13}$$

$$L_{\text{Avg}} = \frac{1}{|C|} \sum_{c=1}^{m} L_c \tag{14}$$

3.1.4 Energy usage

Energy usage on the controller level in a SDN can be broken down into three primary components: the controller's base energy, the dynamic load of the controller energy, and the links and transmission energy.

The controller's base energy refers to the constant energy consumed by the controller when it is operational, regardless of its workload. This includes the power required to maintain basic functions such as running the operating system, keeping the hardware components active, and maintaining network connectivity. The base energy consumption is influenced by the type of hardware used, such as server specifications and processor type, the operating system efficiency, and any background processes running on the controller.

The dynamic load of the controller energy accounts for the energy consumed by the controller due to its workload, such as processing PACKET_IN messages, managing flow tables, and executing control logic. The dynamic load varies based on network activity, traffic volume, and the complexity of control decisions. Key factors influencing this load include the number of connected switches, the frequency of PACKET_IN messages, the complexity of routing decisions, and the efficiency of the controller's software and algorithms.

The links and transmission energy component includes the energy used for communication between the controller and other network elements, such as switches and other controllers. This encompasses both the transmission of control messages and the reception of network status updates. The energy consumption here depends on the communication protocols used, the distance between network components, the bandwidth requirements, and the efficiency of the network interfaces.

The energy usage at the controller level of the SDN is formulated as Eq. (15):

$$E_c = e_{base} + (e_{dyn} \times L_c) + \sum_{s \in S} \frac{d(s,c)}{bw_{s,c}} \times e_{trans} \tag{15}$$

which comprises controller c, the average load of controller L_c, and the transmission energy usage in links, and bw refers to the bandwidth of the link. Coefficients e_{base}, e_{dyn}, and e_{trans} are constants. Equation (16) shows the network's average energy usage at the controller level [23].

$$E_{Avg} = \frac{1}{m} \sum_{c \in C} E_c \tag{16}$$

3.2 Bedbug-GLA

Bedbug-GLA solves the CPP in two steps as shown in Fig. 1. In the first step, Bedbug-GLA utilized the developed ICLA to determine the necessary NC, taking into account traffic and network size, in order to enhance latency and reduce energy usage in the SDN. After determining the NC, in the second step, Bedbug-GLA developed the Bedbug metaheuristic algorithm and utilized it to assign controllers to switches, with the aim of improving load balancing and reducing energy usage and latency.

Fig. 1 Two steps of Bedbug-GLA

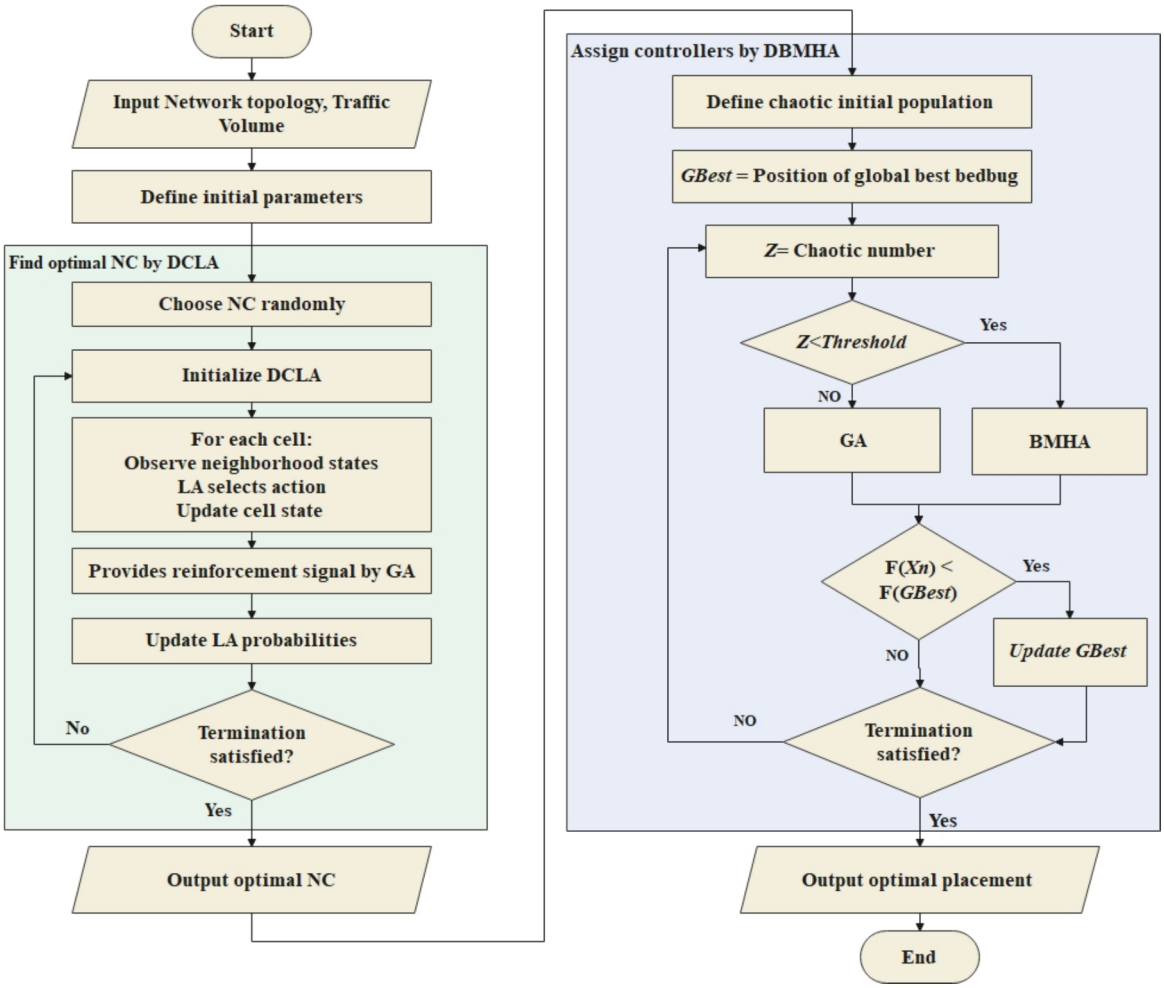


Fig. 2 General flowchart of Bedbug-GLA

Figure 2 illustrates the overall flowchart of Bedbug-GLA for solving the CPP. As shown in the figure, Bedbug-GLA first acquires the network topology and traffic volume, with the traffic volume being dynamically calculated by the controllers. It then randomly defines the number of controllers and assigns them. Subsequently, using the proposed DCLA, Bedbug-GLA calculates the optimal NC by considering the network topology and controller load. In continuation, Bedbug-GLA assigns controllers to switches using the proposed DBMHA. The DBMHA considers latency, load balancing, and energy usage of controllers in its fitness function, thereby solving the controller placement problem effectively.

Bedbug-GLA offers several theoretical advantages over traditional methods for solving the CPP in SDN. These advantages stem from its innovative combination of ICLA and the BMHA, refined into DBMHA.

Bedbug-GLA utilizes ICLA, which provides decentralized learning and adaptability, making it particularly suitable for dynamic network environments. Bedbug-GLA dynamically adjusts the number of controllers based on network load, ensuring optimal performance under varying conditions. The Bedbug metaheuristic algorithm offers flexibility in controller assignment, allowing for efficient placement across different network topologies and conditions.

The use of ICLA enables parallel and distributed processing, significantly reducing computation time compared to centralized methods. This efficiency is crucial for handling large-scale networks. DBMHA enhances convergence speed and avoids local optima by incorporating genetic operators, further improving computational efficiency.

Bedbug-GLA is robust due to its ability to handle distributed and evolving networks effectively. It optimizes resource allocation and adapts to changing network conditions over time, ensuring consistent performance. The use of chaotic maps and genetic operators in DBMHA maintains population diversity, preventing premature

convergence and ensuring robust exploration of the solution space.

Compared to ILP-based dynamic placement methods, which can be computationally intensive and less adaptable to dynamic changes, Bedbug-GLA provides a lightweight and adaptive solution. It also outperforms other metaheuristic approaches in load balancing, congested controllers, and energy efficiency, as demonstrated in simulation results using real-world network topologies.

3.2.1 Finding number of controllers

Bedbug-GLA developed and utilized irregular cellular learning automata (ICLA) [31] to determine the required NCs in the initial phase. The features and benefits of decentralized learning, adaptability, and parallel processing inherent in ICLA made it an attractive choice for this task. ICLA operates in a decentralized manner, allowing individual units or cells to learn and adapt based on their local interactions. This characteristic enables ICLA to effectively manage dynamic and distributed environments.

ICLA can seamlessly adapt to changes in network structure, traffic patterns, and other environmental factors, making it particularly suitable for scenarios where adaptability is essential. Its parallel and distributed nature facilitates efficient and simultaneous learning, thereby enabling the system to handle large-scale networks effectively.

When identifying the appropriate number of network controllers within a network, ICLA demonstrates significant potential to adapt to dynamic environments, learn from feedback, and optimize system configurations. Its strengths lie in its ability to manage distributed and evolving networks, optimize resource allocation, and adjust to changing network conditions over time. This approach proves especially beneficial in situations where the network size and structure may change dynamically or when continuous optimization of network controllers is required based on varying traffic patterns and loads.

However, it is important to note that the time complexity of ICLA is influenced by several factors: the number of cells, the number of actions available to each learning automaton, and the number of iterations needed for convergence. In large networks, calculating the reinforcement signal in ICLA can become computationally expensive due to the necessity for accurate updates of action probabilities based on environmental feedback. This limitation makes ICLA less efficient for large-scale or complex problems.

To address these challenges, Developed Cellular Learning Automata (DCLA) is proposed in this research as an enhanced version of ICLA, designed to improve its efficiency and scalability. A detailed discussion of both ICLA and DCLA is provided in the following sections.

3.2.1.1 Irregular cellular learning automata (ICLA)

ICLAs are mathematical models designed for complex dynamic systems that comprise numerous simple components with learning abilities. These components work together to generate intricate behavioral patterns [53].

An ICLA is a specific type of cellular learning automata (CLA) that features irregular structures or configurations in its grid or lattice. Unlike traditional Cellular Automata, which typically operate on a regular grid (like a square or hexagonal lattice), ICLA allows for variations in the arrangement and connectivity of cells. This irregularity can be in the form of varying cell shapes, sizes, or connection patterns. ICLA are characterized by their non-uniform structures, allowing for variations in cell shapes, sizes, and connectivity within the grid or lattice. Each cell employs learning algorithms to adapt its behavior based on local interactions with neighboring cells, enabling dynamic responses to environmental changes. Despite the irregular arrangement, these cells maintain local interactions that can vary significantly due to their layout. ICLAs are particularly useful in applications involving complex systems, optimization problems, and adaptive networks, where traditional regular grid models may fall short, thus providing a flexible framework for exploring intricate behaviors and adaptations in distributed systems [54]. The ICLA have a structure represented as $A = (G(V,E), \Phi, A, N, F)$, where $G(V,E)$ is a network of nodes and edges, and Φ is a finite set of states. A is a collection of learning automata assigned to the cells of the cellular automata. N is a limited subset of V, referred to as the neighborhood vector, which according to Eq. (17), $\overline{m}$ represents the number of neighboring cells and $\overline{x_i} \in V$ indicates the specific cell in question.

$$N = \{\overline{x_1}, \overline{x_2}, \overline{x_m}\} \tag{17}$$

$$\{u + \overline{x}_i | i = 1, 2, ..., \overline{m}\} \tag{18}$$

The neighborhood vector determines the relative position of neighboring cells from any specific cell u in the network G. Equation (18) specifies that the neighbors of a specific cell u form a set of cells. The neighborhood function $\overline{N}(u)$ maps a cell u to its set of neighbors, as defined in Eq. (19).

$$\overline{N}(u) = \{u + \overline{x}_1 . u + \overline{x}_2 . . . . u + \overline{x}_m\} \tag{19}$$

F: $\Phi_j \rightarrow \beta$ is the local rule of the cellular learning automata at each vertex j, where the set of states of all neighbors of j is represented by Eq. (20). $\beta \in \{0, 1\}$ is the response value set from the environment, where 0 and 1 correspond to reward and penalty, respectively. β computes the reinforcement signal for the learning automata based on the actions chosen by the neighboring cellular learning automata.

$$\Phi_j = \{\Phi_i | (i.j) \in \mathrm{N}\} + \{\Phi_j\} \tag{20}$$

Each cell of the ICLA represents a learning automaton. For each learning automata $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, it defines a set of actions for the learning automaton. $p = (p_1, p_2, \ldots, p_r)$ is the action probability vector, where p_i is the probability of selecting action α_i, satisfying $\sum_{i=1}^{r} pi = 1$, and T adjusts the action probability vector according to the response from the environment. The solution vector is formed by the states of the cells, i.e., the current actions selected by all cells. Therefore, in each iteration cycle t, the solution vector can be written by Eq. (21).

$$S(t) = (\alpha_1(t), \alpha_2(t), \ldots, \alpha_n(t)) \tag{21}$$

where $\alpha_i(t)$ is the action chosen by the cellular learning automata i in cycle t, and the solution vector is updated along with the evolution of the cellular learning automata.

In ICLA, a set of potential reinforcement signals (β) is defined based on the actions taken by successor LAs. ICLA selects an action from a finite set of options based on the probability distribution of actions and implements it in the environment. The environment provides feedback, which is used to update the probabilities of actions using a linear formula [55]. If the feedback is positive ($\beta = 0$), Eq. (22) is used to update the probabilities. However, if the feedback is negative ($\beta = 1$), Eq. (23) is used with reward and penalty parameters.

$$p_j(n+1) = \begin{cases} p_j(n) + a\big(1 - p_j(n)\big) if j = i \\ (1-a)p_j(n) if j \neq i \end{cases} \tag{22}$$

$$p_j(n+1) = \begin{cases} (1-\beta)p_j(n) if j = i \\ \dfrac{\beta}{r-1} + (1-b)p_j(n) if j \neq i \end{cases} \tag{23}$$

3.2.1.2 Developed ICLA (DCLA) Although ICLA offers several benefits, it is hindered by high computational complexity when applied to large-scale networks. To address this limitation, this study proposes DCLA. In DCLA, GA accelerates the calculation of reinforcement signals by employing heuristics and optimizing parameters such as α and β. This approach reduces the computational overhead associated with precise calculations in ICLA, leading to faster convergence. In the proposed DCLA, β values are dynamically set by the GA, which further enhances convergence speed. In contrast to conventional ICLA, where reinforcement signal calculations require accurate and time-consuming computations, DCLA leverages the GA and heuristics to compute these values more efficiently. This results in faster learning and improved adaptability in dynamic environments.

Algorithm 1 Pseudocode of DCLA

```
Initialize the DCLA environment
Repeat
    For each cell Ci in the DCLA do in parallel
        For each Learning Automata LAij in Ci do
// Action selection and genome update
            Select action using the neighbors and probability vector Pi
            old_genome = Ci.genome
            new_genome = generateNewGenome (old_genome)
// Evaluate fitness
            If fitness(new_genome) > fitness(old_genome) Then
                Ci.genome = new_genome
// Select neighboring cells for mating
            neighbors = selectNeighbors(Ci)
// Calculate reinforcement signal using GA
            reinforcement_signal = calculateReinforcementSignal(Ci.genome, neighbors)
// Update action probabilities
            LAij.updateProbabilities (action, reinforcement_signal)
// Evaluate cell genome
            Ci.evaluateGenome (neighbors)
        End For
    End For
Until (a predetermined criterion is met)
Return optimized cell states
```

DCLA process is outlined in Algorithm 1, with each C_i cell undergoing a specific function. LA_{ij} selects an action based on its probability vector P_i, with the chosen actions becoming the old LA genome. A new genome is then generated through crossover and mutation, and its fitness is evaluated. If the new genome is superior to the current one, it replaces it in the cell. Neighboring cells are chosen for mating based on genome fitness, and reinforcement signals

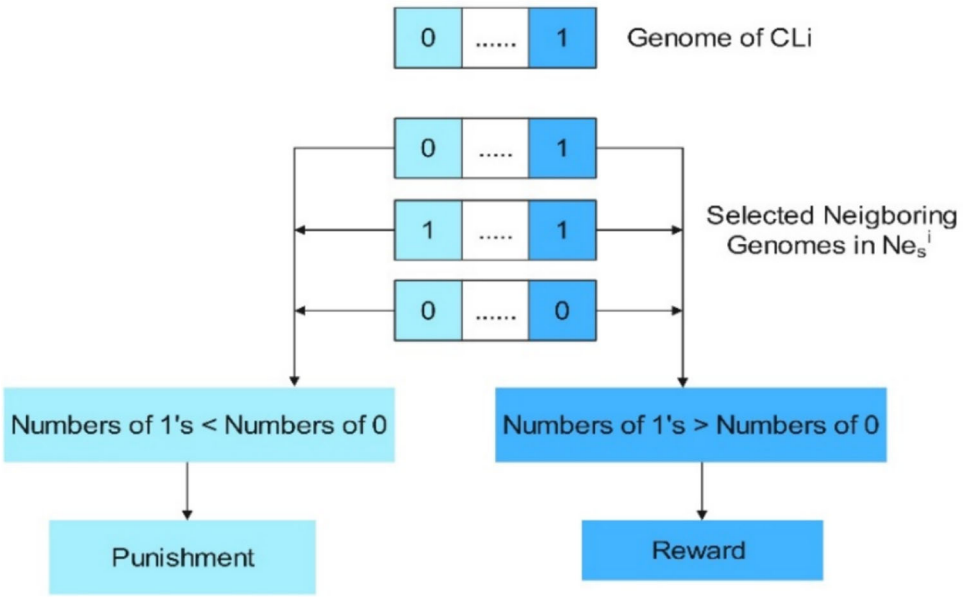


Fig. 3 The reinforcement signal calculation

are regulated through the crossover process and mutation. The reinforcement signal calculated by genetic algorithm based on proposed function in Algorithm 2. Its calculation for each LA_{ij} as shown in Figure 3. LA_{ij} adjusts its action probability vector based on the reinforcement signal and chosen action, while the CL_i cell genome evaluates the experiences of neighboring genomes. This process continues until a termination condition is reached and GA accelerates the calculation of the reinforcement signal and learning in the DCLA.

3.2.1.3 Using DCLA to find NC In order to meet QoS requirements in SDN, the ideal number of network switches would be equivalent to the NCs. However, this approach is not practical in terms of efficiency and cost-effectiveness. To determine the optimal NC, the scale of the network and traffic are measured. Bedbug-GLA proposes a solution to finding the optimal NC based on the load and scale of the SDN by DCLA. When the SDN load increases, the NC must also increase to handle the added load and avoid overloading the control layer. Conversely, if

Algorithm 2 Pseudocode of reinforcement signal calculation by GA

```
Function: calculateReinforcementSignal(genome, neighbors)
  count_ones = 0
  count_zeros = 0
    For each neighbor in neighbors
        For each gene in neighbor.genome
                If gene == 1 then
                        count_ones += 1
                Else
                        count_zeros += 1
        End For
    End For
   If count_ones > count_zeros
        Return REWARD
   Else
        Return PUNISHMENT
```

the SDN load decreases, some controllers can be put into sleep mode or disconnected to reduce energy usage and improve efficiency. DCLA calculates the optimal NC by taking into account factors such as the number and location of switches, links between them, bandwidth, controller processing capacity, cost, and budget allocated for purchasing controllers by providers.

DCLA is used by Bedbug-GLA to determine the optimal NCs based on the size and structure of the network, traffic volume, and controller loads. DCLA uses a cell-based approach where each cell represents a switch and has an LA. The initial NC is chosen randomly, and DCLA adjusts it over time by having each LA choose an action based on its action probability vector. The solution vector $S(t) = ((\alpha_1(t), \alpha_2(t), \ldots, \alpha_n(t))$ is formed by the actions of all LAs at each period t. DCLA method has the action set $a = \{a_1, a_2, a_3\}$, where a_1 is the action of increasing NC, a_2 is the action of decreasing NC, and a_3 is the action of fixing NC, and the action probability vector $p_j \varepsilon P$ is initially set to 0.33 to ensure fairness in action selection and in each iteration it updated.

3.2.2 Controller assignment

Bedbug-GLA uses the new BMHA-based metaheuristic for controller assignment in the second step. Metaheuristics are fast in finding solutions close to the optimum, but they sometimes suffer from getting stuck in the local optimum and premature convergence. Bedbug-GLA proposes DBMHA by combining BMHA, genetic operators and chaotic map to speed up the convergence and avoid local optimum.

3.2.2.1 Bedbug meta-heuristic algorithm (BMHA) The standard BMHA [30] is a relatively recent metaheuristic optimization algorithm inspired by the behavior of bedbugs in search of food.

As shown in Algorithm 3, the BMHA begins with the initialization of a population of bedbugs, randomly distributed within the search space. Each bedbug X_i (where $i = 1, 2, \ldots, N$, and N is the total number of bedbugs) evaluates its position using a fitness function $f(X_i)$, which quantifies the quality of the solution corresponding to the position of each bedbug. The algorithm proceeds through several iterations. In exploration stage bedbugs move randomly to explore the search space. Position update by Eq. (24). Where α is a control parameter, rand is a random number between 0 and 1, ub and lb are the upper and lower bounds of the search space.

$$X_i(t+1) = X_i(t) + \alpha.rand.(ub - lb) \tag{24}$$

In the exploitation stage bedbugs move towards the best solutions found so far and position update by Eq. (25). Where β is a control parameter, $LBest$ is the position of the local best bedbug.

$$X_i(t+1) = X_i(t) + \beta.rand.(LBest - X_i(t)) \tag{25}$$

Equation (26) illustrates how a bedbug's subsequent position can be determined by considering its current location, the target location, and the positions of all other bedbugs in the vicinity. The first component of this equation emphasizes the relationship between the current bedbug's position and those of its peers, allowing for the establishment of effective search agent locations surrounding the target.

A notable feature of the BMHA is that it utilizes three distinct vectors to represent both speed and position for each bedbug. This differentiates BMHA from other algorithms, enhancing its ability to navigate complex environ-

Algorithm 3 Pseudocode of BMHA

```
Initialize the parameters and population;
Set initial parameters and generate random values within specified ranges
While Iter < Max_Iter do
        For each X in population do
          For each V in velocities do
            Set LBest = previous position
            Propagate GBest and SBest using Eqs. (26) and (27)
            Update X to new X′ using Eq. (28)
            Update all variables
            Calculate fitness by Eq. (29)
            Update GBest
          End For
        End For
        Iter = Iter + 1
End While
Return GBest;
```

ments. The update mechanism for a search agent's position is based on several factors: its current position, the best-known condition across the population, the positions of all other search agents, and the estimated location of the target.

In this context, w_i denotes the weight of each bedbug i, which reflects the quality of the solution associated with that specific bedbug within the population X. The coefficients c_1, c_2, and c_3 are referred to as learning coefficients, where c_1 and c_3 represent individual learning coefficients and c_2 serves as an aggregative coefficient. Typically, these coefficients are non-negative and do not exceed a value of 2. Furthermore, $Rand_1$, $Rand_2$, and $Rand_3$ are random vectors corresponding to the length of the position vector, with values uniformly distributed between 0 and 1. At each step, each bedbug moves towards its optimal position using three distinct strategies relative to its current location. The bedbug returns to a position where it previously found prey, about half a meter away from the previous location. This position is remembered and referred to as the best nostalgic value for the bedbug, denoted as *LBest*. Additionally, the bedbug moves towards the best position ever discovered by the entire bedbug population, guided by pheromone trails left by other insects, which attract them to aggregate. This is known as *GBest* and calculated by Eq. (26). Lastly, the bedbug is drawn to the warmth emitted by the prey's body, guiding its movement towards this heat source, referred to as *SBest* and calculated by Eq. (27). These strategies help the bedbugs navigate efficiently in their search for prey.

$$GBest = \sum\nolimits_{J=1\&J\neq i}^{N} S(d_{ij})\widehat{d_{ij}} \tag{26}$$

$$SBest = tan\frac{E}{2} \times \left(g - \widehat{e}_g \times D\right) \tag{27}$$

The distance between bedbugs i and j is represented by d, while S denotes a function that quantifies the strength of social interactions among bedbugs. The expression $\widehat{d}_{ij} = (X_j - X_i)/d_{ij}$ defines a vector pointing from bedbug i to bedbug j.

Additionally, the intensity of heat transfers and carbon dioxide emissions from the human body can be modeled using angle E, which indicates the orientation of the target point relative to a given bedbug. The constant g represents heat transfer properties, while $\widehat{e}_g$ quantifies the percentage reduction in heat over distances of one centimeter, specifically set at 0.185. This comprehensive approach allows for a nuanced understanding of how bedbugs navigate toward their targets while accounting for environmental influences and inter-agent dynamics.

$$\begin{aligned} X_i(t+1) = {} & X_i(t) + (w_i \times V_i(t)) + (C_1 \times Rand_1(t) \\ & \times (LBest_i - X_i(t))) + (C_2 \times Rand_2(t) \\ & \times (GBest_i - X_i(t))) \\ & + (C_3 \times Rand_3(t) \times (SBest_i - X_i(t))) \end{aligned} \tag{28}$$

After updating their positions, each bedbug re-evaluates its fitness using the same fitness function $f(X_i)$.

$$F(X) = \sum_{i=1}^{N} f(X_i) \tag{29}$$

where $B_{\beta est}$ represents the optimal solution identified by the algorithm. The BMHA thus combines exploration and exploitation effectively, leveraging social interactions among bedbugs to enhance the search for optimal solutions in complex problem spaces.

3.2.2.2 Developed bedbug meta-heuristic algorithm (DBMHA) The BMHA offers significant advantages for controller assignment in SDN. Its flexibility allows it to adapt to varying network conditions and topologies, ensuring optimal controller placement. The algorithm is efficient, quickly converging to near-optimal solutions, which reduces computation time. It also excels at avoiding local optima by effectively exploring the search space to find better global solutions. Additionally, its simplicity and understandability make it accessible and practical for real-world applications, ensuring robust and efficient management of SDN controllers.

Algorithm 4 Pseudo-code of DBMHA

```
Initialize the chaotic population using Eq. (30)
Set initial parameters and generate random values within specified ranges
Set the threshold to 0.5
While Iter < Max_Iter do
        For each X in population do
          For each V in velocities do
            Set LBest = previous position
            Calculate Z by Eq.(30)
            If Z<threshold Then
                Propagate GBest and SBest using Eqs. (26) and (27)
                Update X to new X′ using Eq. (28)
                Update all variables
                Calculate fitness by Eq. (31)
                Update GBest
            Else
                Crossover (LBest, GBest) → Xn, Xn+1
                Mutation (Xn)
                Mutation (Xn+1)
                Calculate fitness Xn, Xn+1by Eq. (31)
                Update GBest
            End If
          End For
        End For
        Iter = Iter + 1
End While
Return GBest;
```

However, BMHA like many optimization algorithms, has its own set of limitations. The BMHA may converge too quickly to a local optimum, especially in complex landscapes with many local minima. This can limit its ability to explore the search space thoroughly. Without explicit mechanisms to maintain population diversity, the algorithm may lose valuable genetic material, resulting in a homogeneous population that can hinder exploration.

Figure 4 presents the flowchart of the DBMHA algorithm. The sections highlighted in red represent the additions made to the standard BMHA. As illustrated in Algorithm 4, DBMHA incorporates a chaotic map to enhance the diversity of the initial population while utilizing genetic operators to escape local optima and improve convergence speed. DBHA start with defining parameters and defining the initial chaotic population. then the fitness of all bedbugs is calculated using the proposed fitness function [Eq. (29)]. Based on experimental results, a threshold value of 0.5 is established. To maintain the algorithm's efficiency, a chaotic number is generated using the logistic function in each iteration and stored in variable *Z*. This value *Z* is then compared to the threshold; if *Z* exceeds the threshold, a new bedbug is generated using genetic operators. Specifically, crossover and mutation are applied to the Global-best bedbug (*GBest*) and the second-best bedbug (*LBest*), which are selected as elite individuals. Otherwise, the algorithm proceeds with the standard BMHA. In the continue, if the new bedbug is better than the previous best, the best bedbug (*GBest*) is updated. This process continues until a terminal condition is met and the algorithm converges.

The logistic map serves as a classic example of a simple mathematical model that demonstrates chaotic behavior. The equation [Eq. (30)] represents the logistic map used to generate chaotic values, where x_n denotes the current value (ranging from 0 to 1). In this chaotic map, x_n is calculated randomly; however, DBMHA uses the system time as the seed for the random function to produce varied outputs. *r* is a parameter typically set between 3.57 and 4 to ensure chaos.

$$x_{n+1} = r \cdot x_n \cdot (1 - x_n) \tag{30}$$

To assign the controllers, the solutions' length is defined as the number of switches and each element represents a coordinate switch. Figure 5 shows an example with 10 switches and 3 controllers. There are 10 switches in the SDN, so the solution length is 10. Index of each element indicates the corresponding switch. The values of elements are the IDs of the controllers selected to assign switches. The assignment in this example is as follows: controller 1

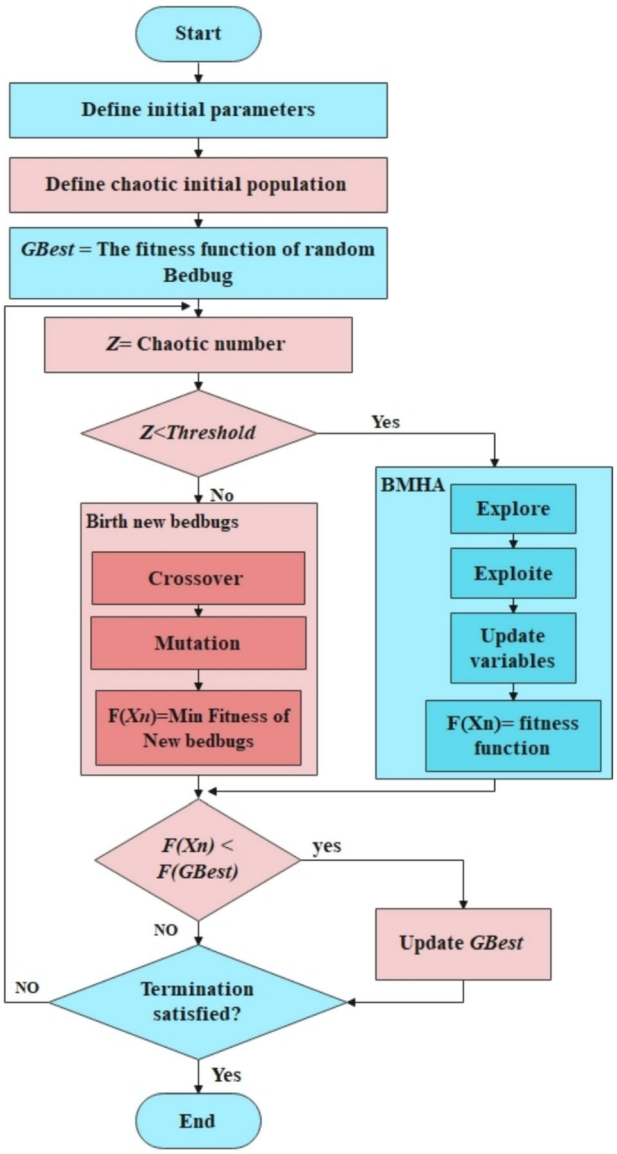


Fig. 4 The flowchart of DBMHA (Color figure online)

manages the 1st, 3rd, 7th, and 9th switches, controller 2 manages the 4th, 5th, and 8th switches, and controller 3 manages the 2nd, 6th, and 10th switches.

LBest and *GBest* bedbugs are selected as elite individuals to apply GA operators on them. Single-point crossover involves selecting a random point on both bedbugs. This chosen point splits the parent cells into segments, and crossover takes place by swapping these segments to create new offspring [56]. An illustration of this crossover process between two bedbugs in Fig. 6.

The variety of the samples increases by mutation operator. It also prevents the local optima. There are different types of mutations. However, as shown in Fig. 7, the mutation is applied to newly borne bedbugs and is done by randomly selecting and changing an element of a bedbug.

1	3	1	2	2	3	1	2	1	3	(X_1)

Fig. 5 An example of solutions

One element of a bedbug is randomly selected for mutation. Then, the random number $[1, |C|]$ is selected and replaced by the selected element. $|C| = m$ is the controller count in the SDN. In this example, the selected element is switch 5 and the random number for the controller is 3, so the value of the 5th element is changed to 3.

As shown in Algorithm 4, After the birth of new bedbugs, their fitness is calculated. If the fitness of the new bedbugs is smaller than the fitness of best bedbug, they are added to the bedbugs 'population as new solutions and the *GBest* is updated.

This paper aims to address energy usage, controller load, and propagation latency parameters. The fitness function for bedbug X_i is defined using Eqs. (31) and (32), with each metric normalized by its corresponding maximum value. The maximum values are determined by the network operator and reflect the physical characteristics of the network devices. The weighted sum method is used to determine the degree of importance of each metric, with adjustable weighting factors *w1, w2*, and *w3*. These factors were determined through empirical testing and experimentation, as presented in Table 2.

$$\mathrm{f(Xi)} = w1 \times \left(\frac{dprop_{Avg}}{dprop_{Max}}\right) + w2 \times \left(\frac{L_{Avg}}{L_{Max}}\right) + w3 \times \left(\frac{E_{Avg}}{E_{Max}}\right) \tag{31}$$

$$\mathrm{Min\ f(X_i)} \tag{32}$$

3.3 Computational complexity

Calculating algorithm complexity is essential for ensuring performance, efficiency, and scalability in software systems. Therefore, this section analyzes the computational complexity of the Bedbug-GLA algorithm, which consists of two main steps.

The first step of Bedbug-GLA utilizes ICLA to determine the required number of controllers. This step involves several key operations. Initialization of each cell's action probability vector results in a complexity of $O(n)$, where n is the number of cells (switches). The action selection and learning process, repeated over time until convergence, adds a factor of t iterations, leading to a complexity of $O(n \times t)$. Additionally, crossover and mutation operations contribute to the overall complexity but are typically proportional to n. Assuming t is significant, the overall complexity of this step is dominated by $O(n \times t)$.

The second step involves using a modified Bedbug metaheuristic algorithm enhanced with genetic operators (DBMHA) for controller assignment. The complexity of this step can be analyzed by considering the operations involved in DBMHA. Initialization has a complexity of $O(P)$, where P is the population size. Chaotic map generation contributes a complexity of $O(t)$, while genetic operations (crossover and mutation) contribute a complexity of $O(n \times t \times P)$. The fitness calculation, which involves evaluating a weighted sum of metrics for each bedbug, dominates the complexity with $O(n \times m \times P \times t)$, where m is the number of controllers. Overall, the complexity of this step is generally $O(n \times m \times P \times t)$, reflecting the interplay between the number of switches, controllers, population size, and iterations. The overall complexity of Bedbug-GLA is primarily influenced by the second step, making it $O(n \times m \times P \times t)$.

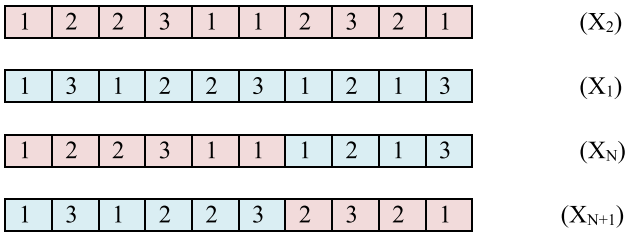


Fig. 6 The crossover on bedbugs (Color figure online)

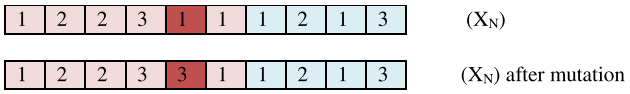


Fig. 7 An example of a mutation (Color figure online)

Table 2 The weights of parameters

Coefficient	Value
w1	0.59
w2	0.4
w3	0.01

4 Performance evaluation

To evaluate the performance of Bedbug-GLA, simulations were done and Bedbug-GLA is compared with BMHA [30] to show the effect of improvement. It was also compared with other state-of-the-art metaheuristic algorithms in CPP including PSO [11], GEWO [12], and ALO [43]. PSO is one of the most common and popular swarm intelligence techniques. The GEWO approach combines a hybrid metaheuristic with game theory. The ALO strikes a great balance between exploration and exploitation, demonstrating good performance. For each method evaluates key parameters such as controller loads, congested controllers, end-to-end delay, and energy consumption.

4.1 Simulation setup

The methods and algorithms were implemented using MATLAB on a computer with an Intel Core i7 processor and 16 GB RAM. The algorithms ran for 30 iterations. The standard network topology from the ITZ [56, 57] is used, along with a range of network topologies from various providers. This repository features over 200 network topologies from Internet Service Providers (ISPs) at the Point of Presence level. Each ISP's network graph is provided, with every node (referred to as a switch) marked with its geographical coordinates. The selection included the Internet2 OS3E network topology [57], a widely adopted configuration for research purposes [11, 38, 42], consisting of 34 nodes interconnected by 42 links. Additionally, the IRIS topology, encompassing 51 nodes and 64 links, represents a common network prevalent in Tennessee, Kentucky, Alabama, Virginia, and Georgia. Furthermore, the Colt Telecom topology [58], spanning globally with 153 nodes and 197 links, known for its extensive coverage and complexity, was also chosen. These topologies were specifically selected for evaluating and testing the methods, with detailed specifications provided in Table 3. Figure 8 visually showcases the utilization of real-world topologies. The initial population was set to 30. The traffic was generated randomly, defined between different origins and destinations with a varying number of packets in the range of 0 to 500. Table 4 shows the simulation parameters and constants and their values.

Table 3 The details of the SDNs

	Network	Type	Geo extent	Geolocation	Layer	No. switch	No. links
1	Internet2 OS3E (InternetMCI)	COM	Country	USA	IP	34	42
2	IRIS	COM	Region	Tennessee, USA	IP	51	64
3	Colt	COM	Continent	Europe	IP	153	177

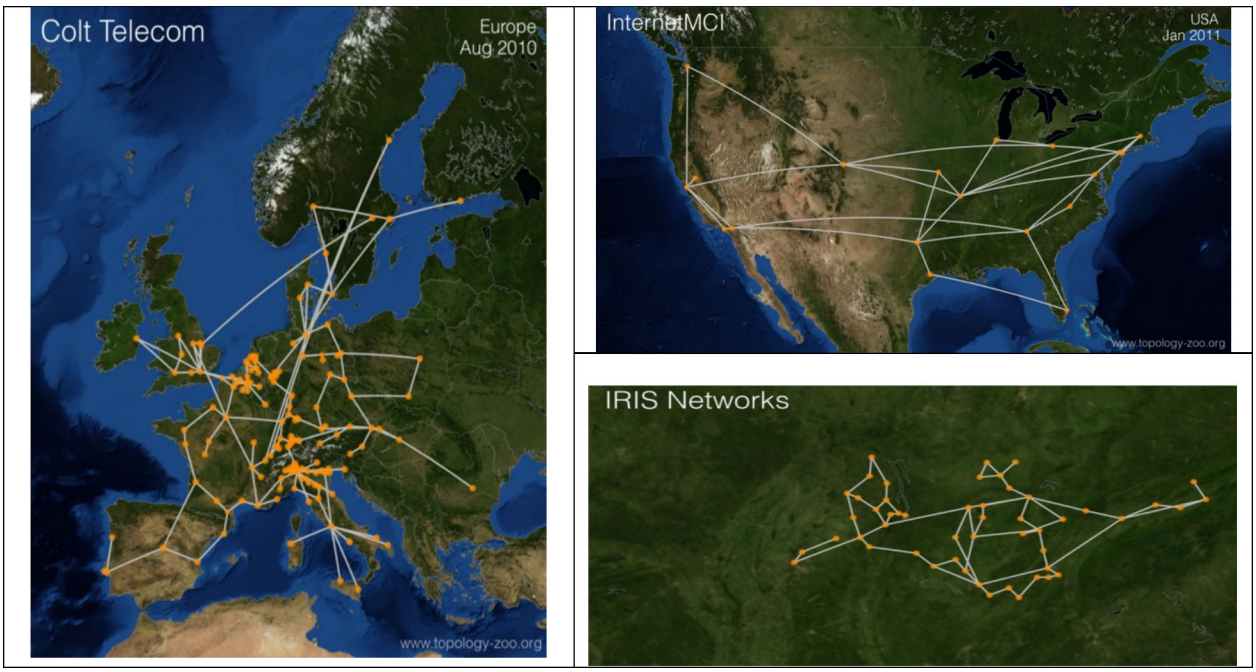


Fig. 8 The Internet2 OS3E, IRIS and Colt network topologies [59]

Table 4 The simulation parameters and values

Parameter/constants	value
Initial population	30
Run time	30 iterations
Traffic	Random (0, 500)
e_{base}	0.1
e_{dyn}	0.001
e_{trans}	1
l_i	1
α, β (DCLA)	0.01
$\widehat{e}_g$ *(BMHA)*	0.185
MinVar (BMHA)	0
MaxVar (BMHA)	1
Number of search agents (ALO)	40
r_1, r_2 (PSO, GEWO)	A random number in [0,1]
c_1' (PSO)	1.7
c_2' (PSO)	2
Inertial weight (PSO)	0.75

4.2 Weights' sensitivity analysis

The fitness function of DBMHA integrates multiple performance metrics, including energy consumption, latency, and load balancing, with weights assigned to prioritize energy efficiency and latency. To assess the robustness and flexibility of DBMHA, a comprehensive sensitivity analysis was conducted. This involved varying each weight by ± 20% and examining the impact on energy consumption and latency across different numbers of controllers. The analysis aimed to identify which weights exert the most significant influence on system performance and to explore potential trade-offs between energy efficiency and latency.

The sensitivity analysis revealed that adjusting the weights leads to notable changes in both energy consumption and latency. For example, increasing the weight on energy efficiency generally results in reduced energy consumption but may slightly increase latency. Conversely, prioritizing latency leads to faster response times at the expense of higher energy usage. These findings underscore the flexibility of the approach and highlight the importance of carefully selecting weights based on specific operational requirements. By exploring these trade-offs, system performance can be optimized to meet diverse needs, whether prioritizing energy efficiency, minimizing latency, or achieving a balanced compromise between these competing objectives.

To evaluate the impact of weighting factors in the DBMHA fitness function, experiments were performed by adjusting the values of *w1*, *w2*, and *w3* for parameters such as energy consumption and latency. The results are summarized in Table 5, which presents energy consumption values for different combinations of these coefficients in the Internet2 OS3E topology. The algorithm achieves optimal performance, highlighted in grey, when the weighting factors are set to $w1 = 0.59$, $w2 = 0.40$,

Table 5 Sensitivity analysis of weights on energy consumption and latency (Color table online)

Scenario	w1	w2	w3	Energy Consumption			Latency		
				NC=3	NC=4	NC=5	NC=3	NC=4	NC=5
Original	**0.59**	**0.4**	**0.01**	**1801 J**	**1902 J**	**1962 J**	**4.27 ms**	**3.89 ms**	**3.65 ms**
w1 -20%	0.472	0.43	0.085	1750 J	1850 J	1910 J	4.35 ms	3.95 ms	3.70 ms
w1 +20%	0.708	0.27	0.005	1850 J	1950 J	2010 J	4.10 ms	3.80 ms	3.55 ms
w2 -20%	0.63	0.32	0.05	1820 J	1920 J	1980 J	4.20 ms	3.85 ms	3.60 ms
w2 +20%	0.55	0.48	0.005	1780 J	1880 J	1940 J	4.30 ms	3.90 ms	3.65 ms
w3 -20%	0.60	0.39	0.008	1810 J	1910 J	1970 J	4.25 ms	3.88 ms	3.62 ms
w3 +20%	0.58	0.41	0.012	1790 J	1890 J	1950 J	4.28 ms	3.86 ms	3.63 ms

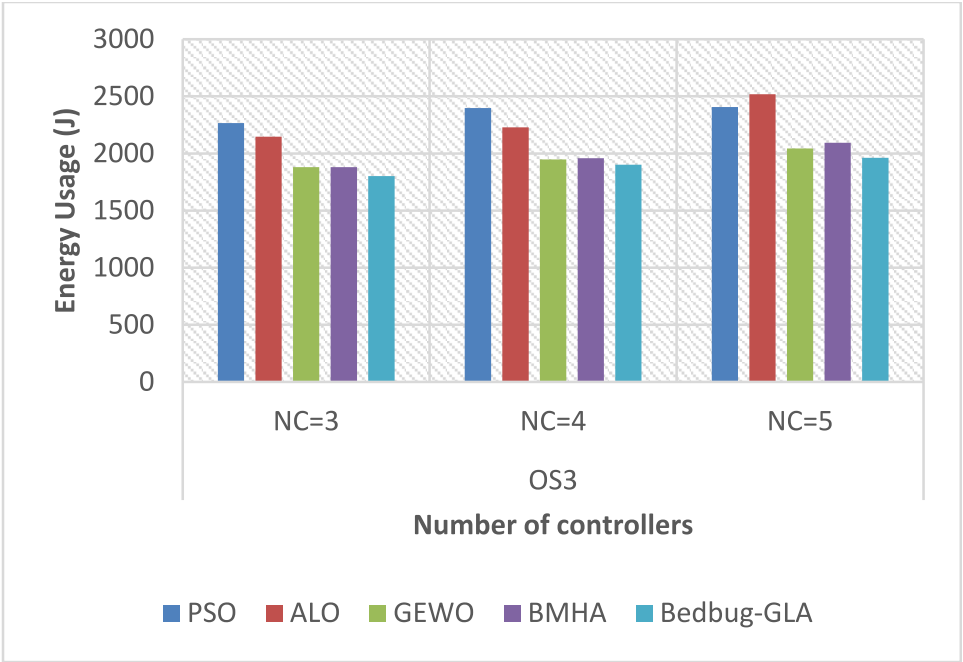


Fig. 9 The energy usage in the Internet2 OS3E topology (Color figure online)

and $w3 = 0.01$. This optimal configuration demonstrates the effectiveness of DBMHA in balancing competing performance metrics.

4.3 Energy usage

Reducing energy usage is a key objective of this study. Figures 9, 10, and 11 illustrate the average energy usage in Joules for the Internet2 OS3E, IRIS, and Colt topologies with different NCs, respectively. These figures demonstrate that Bedbug-GLA enhances the energy efficiency of the controllers within the network. Energy consumption in the control plane is influenced by the NCs, their load, and the distance packets must travel. Bedbug-GLA exhibits energy usage of less than 2000 J in the Internet2 OS3E topology, under 2400 J in the IRIS topology, and below 7000 J in the Colt topology, all of which are lower than that of the compared algorithms. Unlike PSO and ALO, BMHA, GEWO, and Bedbug-GLA incorporate energy consumption considerations into their controller placement strategies, leading to lower energy usage. However, Bedbug-GLA distinguishes itself by achieving superior performance and reduced energy consumption. This is due to its innovative approach in the second step, where the fitness function integrates DBMHA to optimize both energy consumption and load balancing. As a result, Bedbug-GLA efficiently allocates controllers to switches, significantly reducing the network's overall energy consumption. Moreover, by ensuring balanced traffic distribution across the network, Bedbug-GLA prevents resource overutilization, which in turn contributes to its low energy usage profile.

The Internet2 OS3E topology with different controllers is selected to calculate the improvement percentage in energy usage parameters for Bedbug-GLA. Table 6 shows the mean energy usage in 100 runs for different topologies with 3, 4 and 5 NC. Bedbug-GLA improves energy usage

up to about 20% compared to PSO and by about 18% compared to ALO, and about 4% compared to GEWO, and BMHA.

4.4 Load of controllers

To evaluate the load on the controllers, both the average and maximum loads were considered for each method. In this simulation, it was assumed that the route calculations for a single flow from switch i required l_i units of load on the controller, with $l_i = 1$. Figures 12, 13, and 14 display the average loads on controllers in the Internet2 OS3E, IRIS, and Colt topologies with 3, 4, and 5 controllers, respectively. These topologies were subjected to varying traffic conditions, with the number of flows generated randomly within the range of 0 to 500. As shown in the figures, the average controller load for Bedbug-GLA across different topologies and controller counts is lower than that of other methods, indicating effective load distribution. In contrast, PSO exhibits the highest load on its controllers due to being trapped in local optima. Regarding controller load performance, BMHA fails to reach maximum flow capacity. PSO, ALO, and BMHA generally demonstrate poor performance, often resulting in high maximum controller loads due to inadequate load balancing. With an equal NCs, Bedbug-GLA consistently outperforms other methods, including BMHA. This superior performance is attributed to Bedbug-GLA's enhancements over the original BMHA through strategic modifications. The integration of mutation and crossover operators into BMHA improves the exploration–exploitation trade-off, while generating the initial population using chaotic maps enhances population diversity. This ultimately increases the likelihood of quickly and accurately identifying the optimal solution.

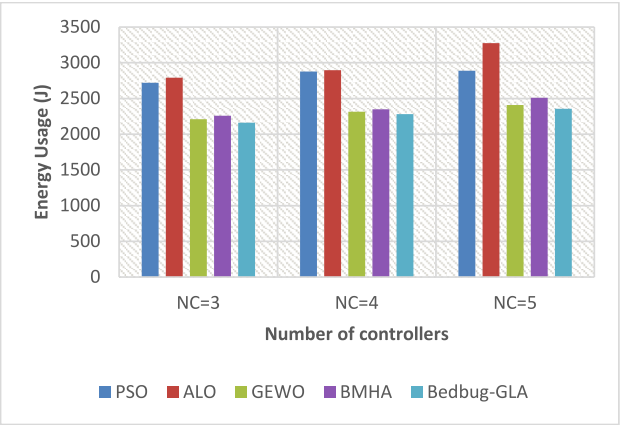


Fig. 10 The energy usage in the IRIS topology (Color figure online)

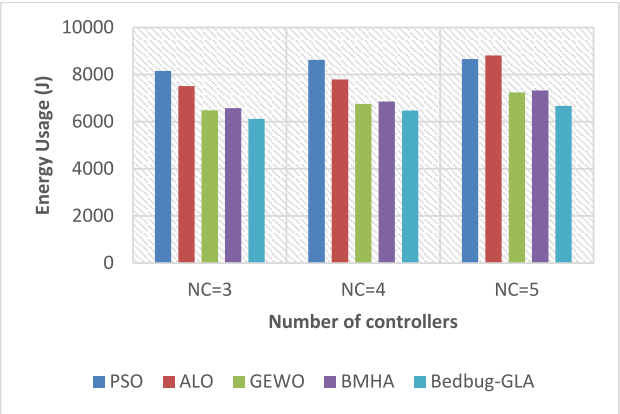


Fig. 11 The energy usage in the Colt topology (Color figure online)

Figures 15, 16, and 17 illustrate the maximum load on controllers in the Internet2 OS3E, IRIS, and Colt topologies with 3, 4, and 5 controllers, respectively. It is clear that the maximum loads of controllers using the Bedbug-GLA method are lower compared to other methods, demonstrating effective load balancing. Since GEWO, BMHA, and Bedbug-GLA consider controller load in their fitness functions, their maximum controller loads are similar. In contrast, PSO and ALO exhibit higher and more varied maximum loads due to their inferior load balancing capabilities. Similar to the average controller loads, PSO shows higher maximum loads compared to other methods, which is attributed to its poor load distribution. Through efficient load distribution among controllers, Bedbug-GLA successfully reduces the maximum load on controllers, showcasing its superior performance in load balancing.

Table 7 shows the average of maximum loads of controllers in 100 runs for Internet2 OS3E topology in different NCs, average of them and improvement percentage of Bedbug-GLA to each method. As shown in Table 7,

Table 6 The energy usage analysis of Bedbug-GLA in comparison with other methods (Color table online)

Method	NC=3	NC=4	NC=5	AVG	Percent improvement (%)
PSO	2266.05	2396.92	2407.24	2356.74	19.85
ALO	2146.42	2228.37	2518.13	2297.64	17.79
GEWO	1880.38	1948.24	2043.90	1957.51	3.51
BMHA	1880.65	1958.09	2092.74	1977.16	4.47
Bedbug-GLA	1801.70	1902.39	1962.49	1888.86	0

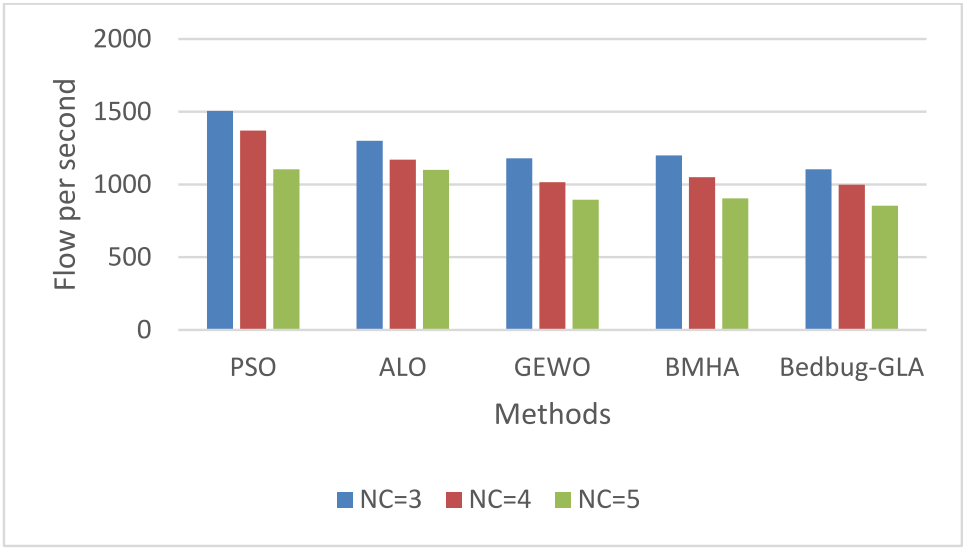


Fig. 12 The average load of controllers in the Internet2 OS3E topology (Color figure online)

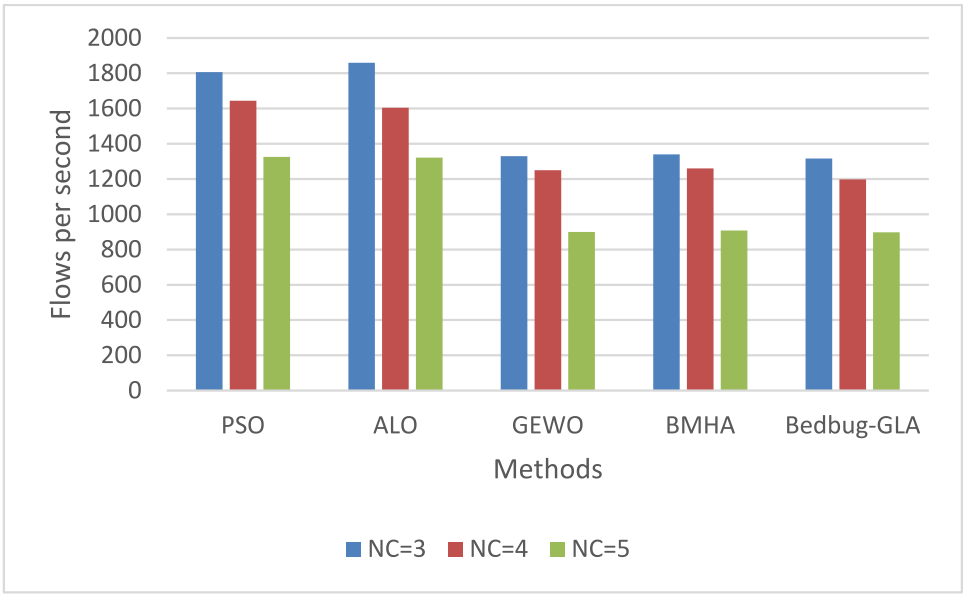


Fig. 13 The average load of controllers in the IRIS topology (Color figure online)

Bedbug-GLA entails 18% improvement to PSO, 10% improvement to ALO, ≈ 4% improvement to GEWO and BMHA.

4.5 Adaptation to traffic variability

The Bedbug-GLA algorithm is designed to optimize SDN controller placement, ensuring efficient load balancing across the network. However, real-world networks often face sudden traffic spikes and long-term changes in traffic patterns, which can significantly impact network performance. This section explores how Bedbug-GLA adapts to these dynamic conditions.

Sudden traffic spikes can overwhelm network resources, leading to congestion and increased latency. The Bedbug-GLA algorithm incorporates mechanisms to dynamically adjust controller loads during such spikes. By adjusting the NC based on the real-time traffic conditions and adjusting controller assignment accordingly, Bedbug-GLA ensures that no single controller becomes overwhelmed, maintaining optimal network performance even under stress.

To evaluate the performance of the methods under spike traffic conditions, a traffic pattern is applied to the network as shown in Fig. 18. The chart shows a baseline traffic volume of 100 flows from time 1 to 4 s. From time 5 to 10 s, the traffic volume increases to a peak of 500 flows, representing a traffic spike. After the peak, the traffic gradually decreases back to the baseline level by time 15 s.

Analyzing the results from the Tables 8, 9 and 10 for the Internet2 OS3E, IRIS, and Colt topologies provides

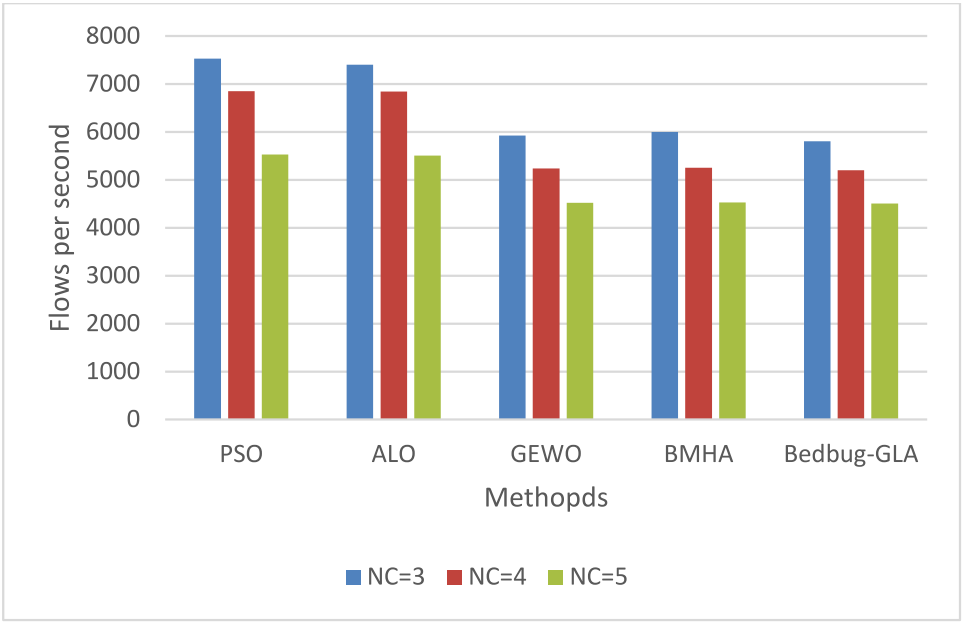


Fig. 14 The average load of controllers in the Colt topology (Color figure online)

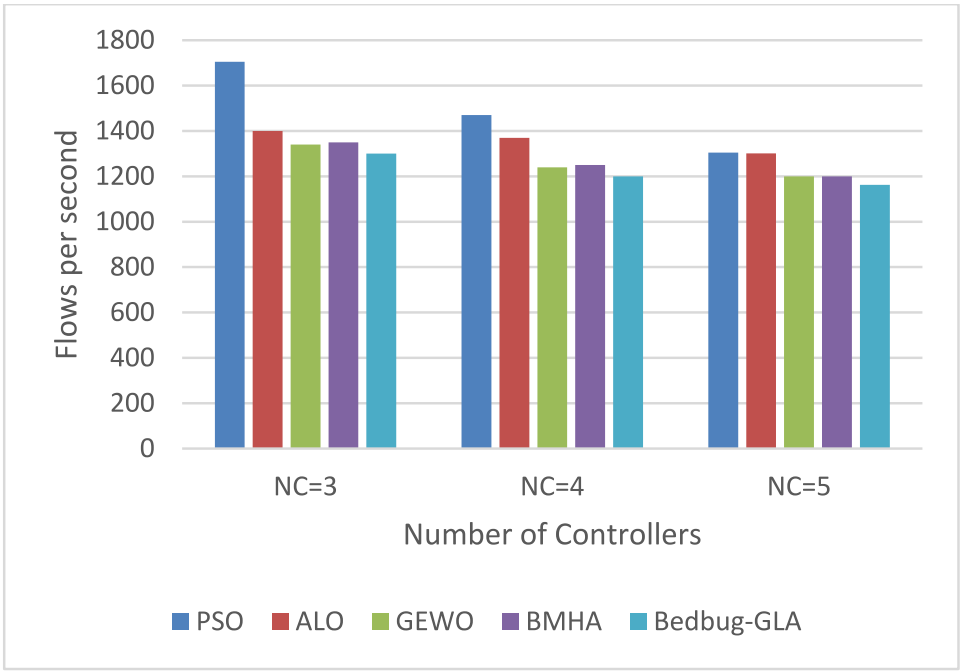


Fig. 15 The maximum load of controllers in the Internet2 OS3E topology (Color figure online)

valuable insights into how different algorithms adapt to traffic variability. Across all three topologies, PSO and ALO consistently demonstrate superior latency performance, which is crucial for applications requiring real-time communication. However, this comes at the cost of higher energy consumption and less effective load balancing compared to Bedbug-GLA and BMHA. The latter algorithms excel in energy efficiency and load balancing, making them more suitable for environments where resource optimization is paramount.

In the Internet2 OS3E topology, Bedbug-GLA shows a load balancing index of 0.95 with five controllers, indicating excellent load distribution. In contrast, PSO and ALO have lower load balancing indices but achieve better latency. This trend is consistent across the IRIS and Colt topologies, where Bedbug-GLA maintains superior load balancing while PSO and ALO excel in latency. Notably, the larger Colt topology highlights the scalability challenges faced by all algorithms, with Bedbug-GLA and

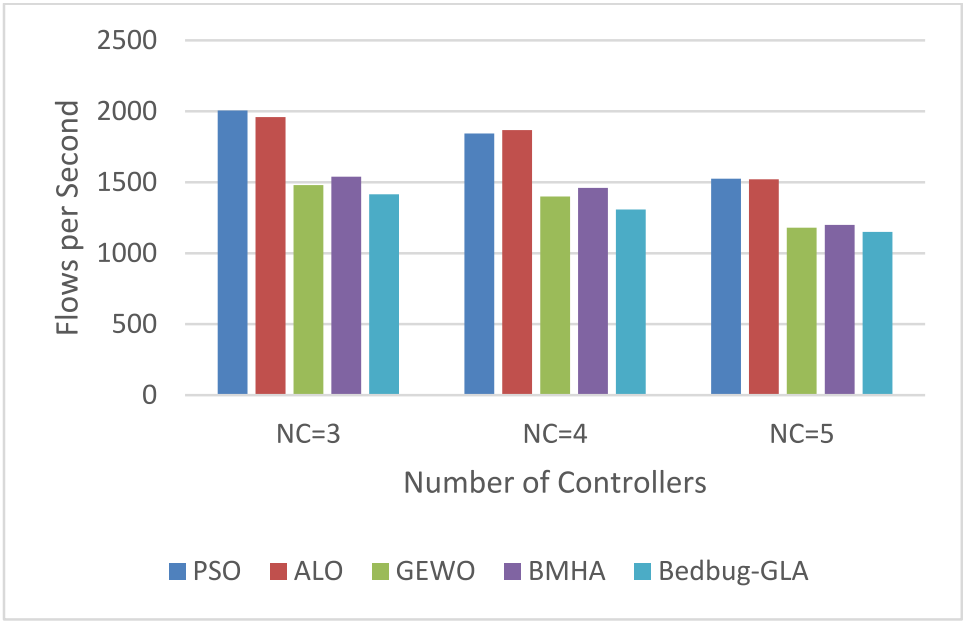


Fig. 16 The maximum load of controllers in the IRIS topology (Color figure online)

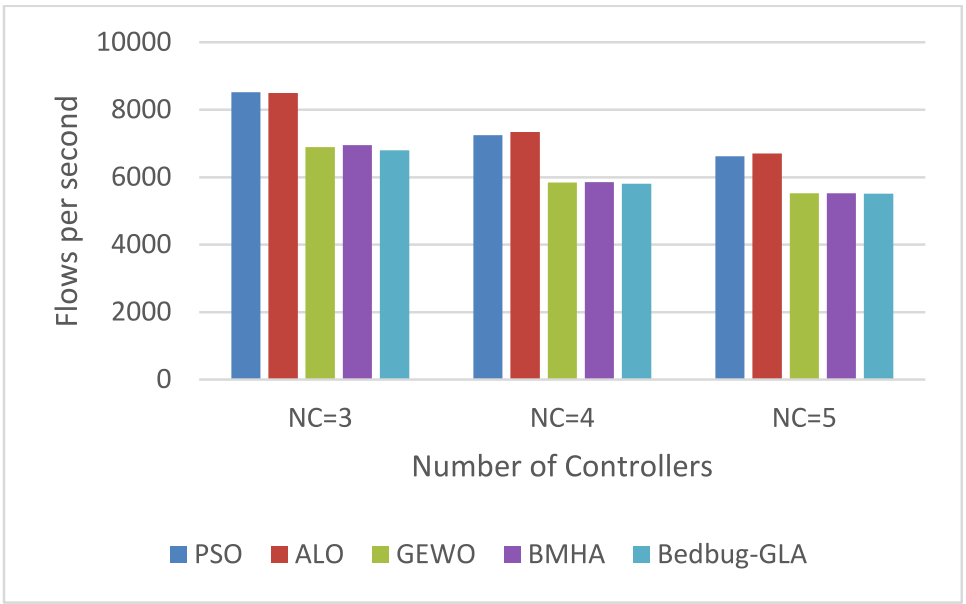


Fig. 17 The maximum load of controllers in the Colt topology (Color figure online)

Table 7 The max loads of controllers' analysis of Bedbug-GLA in comparison with other methods (Color table online)

Method	NC=3	NC=4	NC=5	Average	Percent improvement (%)
PSO	1705	1470	1305	1493.33	18.24
ALO	1400	1370	1301	1357	10.02
GEWO	1340	1240	1240	1273.333	4.11
BMHA	1350	1250	1200	1266.67	3.61
Bedbug-GLA	1300	1200	1163	1221	0.00

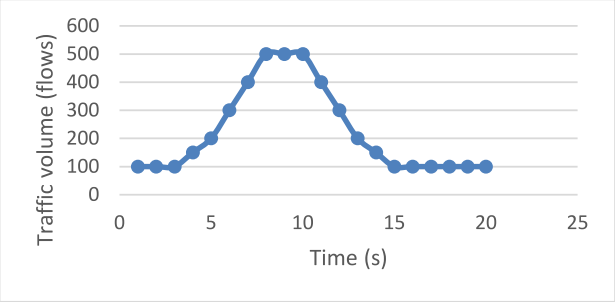


Fig. 18 Traffic Flow with Sudden Spikes

BMHA showing resilience in maintaining efficient load balancing despite increased network size.

The results underscore the importance of considering multiple performance metrics when evaluating SDN controller placement strategies. While PSO and ALO are ideal for latency-sensitive applications, Bedbug-GLA, GEWO and BMHA are better suited for environments prioritizing energy efficiency and load balancing. This analysis provides a comprehensive view of how different algorithms perform under varying network conditions, aiding in the selection of the most appropriate algorithm based on specific network requirements.

Table 8 Performance comparison of methods under traffic spikes in Internet2 OS3E topology

Algorithm	NC	Average latency (ms)	Energy consumption (J)	Load balancing index
Bedbug-GLA	3	12.5	**180**	0.85
Bedbug-GLA	4	10.2	200	0.92
Bedbug-GLA	5	8.5	220	**0.95**
BMHA	3	15.1	190	0.80
BMHA	4	12.8	210	0.88
BMHA	5	11.2	230	0.90
GEWO	3	13	185	0.82
GEWO	4	10.4	210	0.92
GEWO	5	9.2	230	0.92
PSO	3	9.5	280	0.60
PSO	4	8.2	300	0.65
PSO	5	7.1	320	0.70
ALO	3	8.8	290	0.55
ALO	4	7.5	310	0.60
ALO	5	**6.9**	330	0.65

Bold represent the best (most optimal) value for each parameter within the table

Table 9 Performance comparison of methods under traffic spikes in IRIS topology

Algorithm	NC	Average latency (ms)	Energy consumption (W)	Load balancing index
Bedbug-GLA	5	11.5	**240**	0.88
Bedbug-GLA	6	9.8	275	0.92
Bedbug-GLA	7	8.5	300	**0.95**
BMHA	5	14.1	250	0.82
BMHA	6	11.9	280	0.85
BMHA	7	10.5	310	0.88
GEWO	5	11.5	**240**	0.84
GEWO	6	10.1	280	0.89
GEWO	7	9.2	310	0.90
PSO	5	9.5	320	0.60
PSO	6	8.2	350	0.65
PSO	7	**7.1**	380	0.70
ALO	5	10.8	330	0.55
ALO	6	9.3	360	0.60
ALO	7	8.5	390	0.65

Bold represent the best (most optimal) value for each parameter within the table

Table 10 Performance comparison of methods under traffic spikes in IRIS topology

Algorithm	NC	Average latency (ms)	Energy consumption (W)	Load balancing index
Bedbug-GLA	10	14.2	**420**	0.85
Bedbug-GLA	11	12.1	450	0.90
Bedbug-GLA	12	10.5	490	**0.93**
BMHA	10	17.1	450	0.80
BMHA	11	14.5	470	0.85
BMHA	12	12.8	500	0.88
GEWO	10	16.2	420	0.82
GEWO	11	14.5	460	0.85
GEWO	12	10.8	490	0.90
PSO	10	11.5	550	0.60
PSO	11	9.8	600	0.65
PSO	12	**8.5**	650	0.70
ALO	10	13.5	580	0.55
ALO	11	11.8	630	0.60
ALO	12	10.2	680	0.65

Bold represent the best (most optimal) value for each parameter within the table

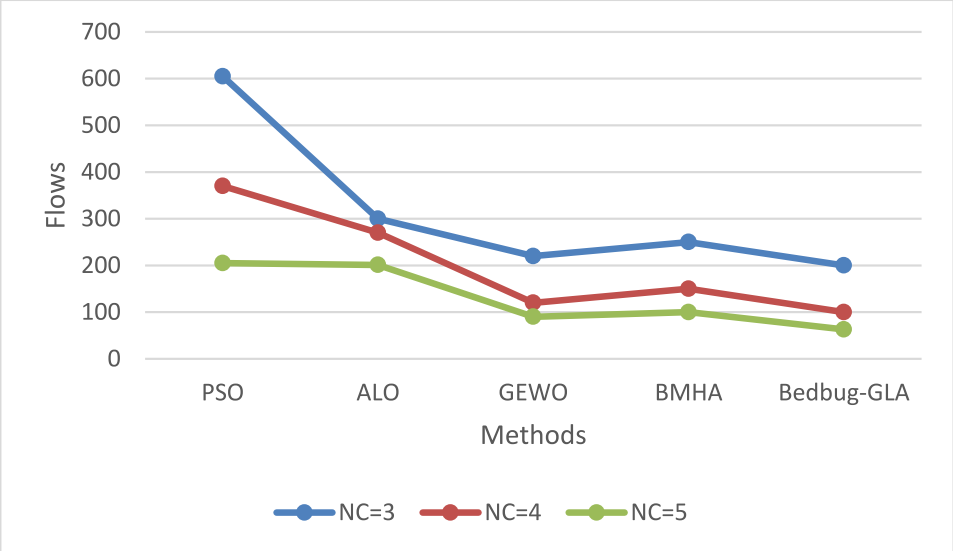


Fig. 19 The average load of the congested controllers in the Internet2 OS3E topology (Color figure online)

4.6 Congested controllers

When the load level (Lc) and queue length (Qlc) in a specific controller (c) reach a threshold (Qt), the controller is considered congested, which is defined as occurring at 90% of its load capacity. To evaluate congestion and overloads in the controllers, each controller's capacity and queue length are set to accommodate 10 flows, with Qt also set at 90%. Figures 19, 20, and 21 illustrate the total overload of congested controllers within the Internet2 OS3E, IRIS, and Colt topologies under random traffic conditions.

Bedbug-GLA, BMHA, and GEWO effectively assign controllers in SDN to achieve load balancing at the controller level and reduce congestion. To this end, these methods and algorithm consider the load of controllers in their fitness functions. In contrast, PSO and ALO demonstrate lower performance regarding the load on controllers and the number of congested controllers, primarily because they do not take controller load into account during the assignment process. Bedbug-GLA addresses this issue by prioritizing load balancing and minimizing congestion. Additionally, Bedbug-GLA determines the appropriate NCs through DCLA, increasing the NCS as needed to avoid congestion while reducing the count to conserve energy and costs when possible.

The Internet2 OS3E topology with different controllers is selected to calculate the improvement percentage in 100

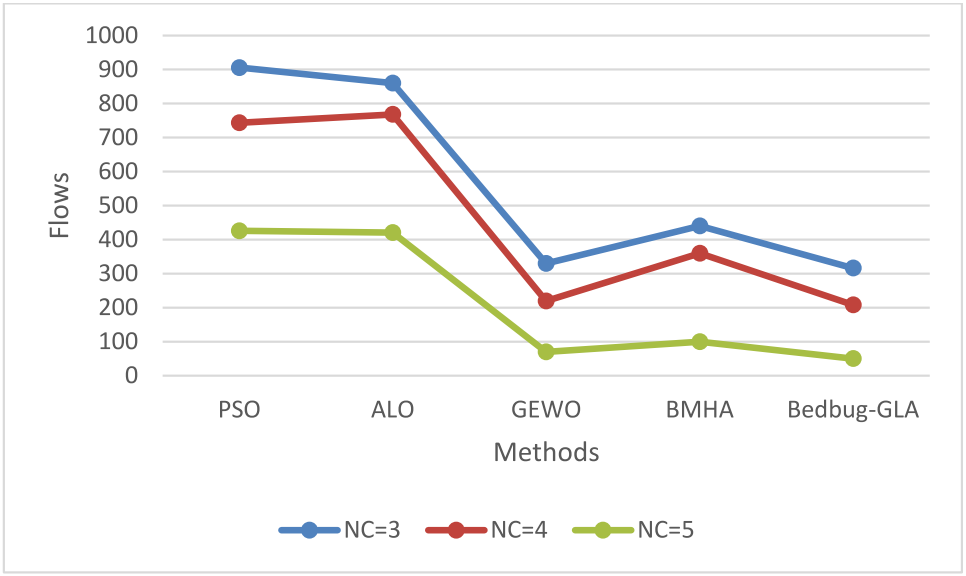


Fig. 20 The average load of the congested controllers in the IRIS topology (Color figure online)

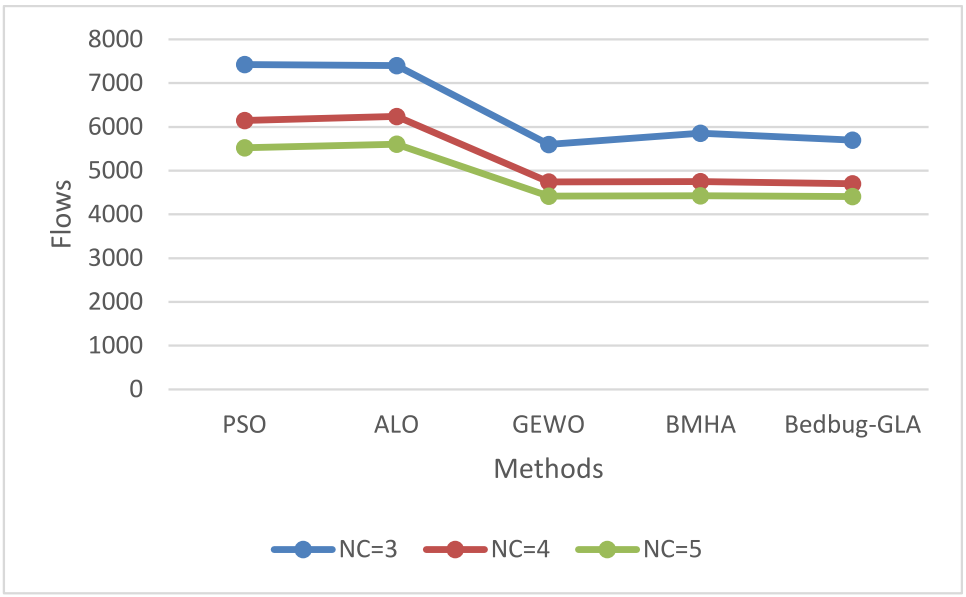


Fig. 21 The average load of the congested controllers in the Colt topology (Color figure online)

runs in the overload of congested controllers for Bedbug-GLA. As shown in Table 11, Bedbug-GLA exhibits about 69%, 53%, 16% and 27% improvement in the over load of congested controllers' parameter compared to PSO, ALO, GEWO and BMHA, respectively.

4.7 End-to-end delay

Figures 22, 23, and 24 illustrate the end-to-end delay in the Internet2 OS3E, Iris, and Colt topologies for the PSO, ALO, BMHA, and Bedbug-GLA with 3, 4, and 5 NCs, respectively. From these figures, it is evident that ALO demonstrates the best performance in terms of end-to-end delay compared to the other methods evaluated. This superior performance can be attributed to the fact that both PSO and ALO primarily focus on minimizing latency and network delay alongside reliability parameters.

In contrast, while BMHA and Bedbug-GLA also take into account propagation delay, they additionally incorporate energy usage and load parameters into their optimization processes. This inclusion is crucial as it highlights a significant trade-off between energy efficiency and latency. Specifically, optimizing for lower energy consumption may lead to increased delays in packet transmission due to factors such as reduced processing power or longer paths taken to conserve energy.

Moreover, in scenarios where stringent latency requirements exist, such as real-time applications, the

Table 11 The loads of congested controllers' analysis of Bedbug-GLA in comparison with other methods (Color table online)

Method	NC=3	NC=4	NC=5	AVG	Percent improvement (%)
PSO	605	370	205	393.33	69.24
ALO	300	270	201	257.00	52.92
GEWO	220	120	90	143.33	15.58
BMHA	250	150	100	166.67	27.40
Bedbug-GLA	200	100	63	121.00	0.00

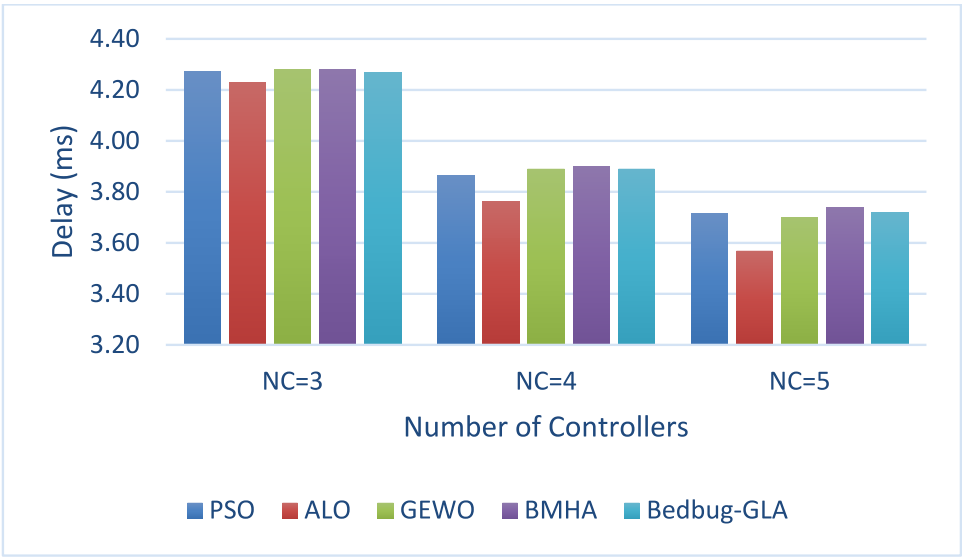


Fig. 22 End-to-end delay in the Internet2 OS3E topology with different controller numbers (Color figure online)

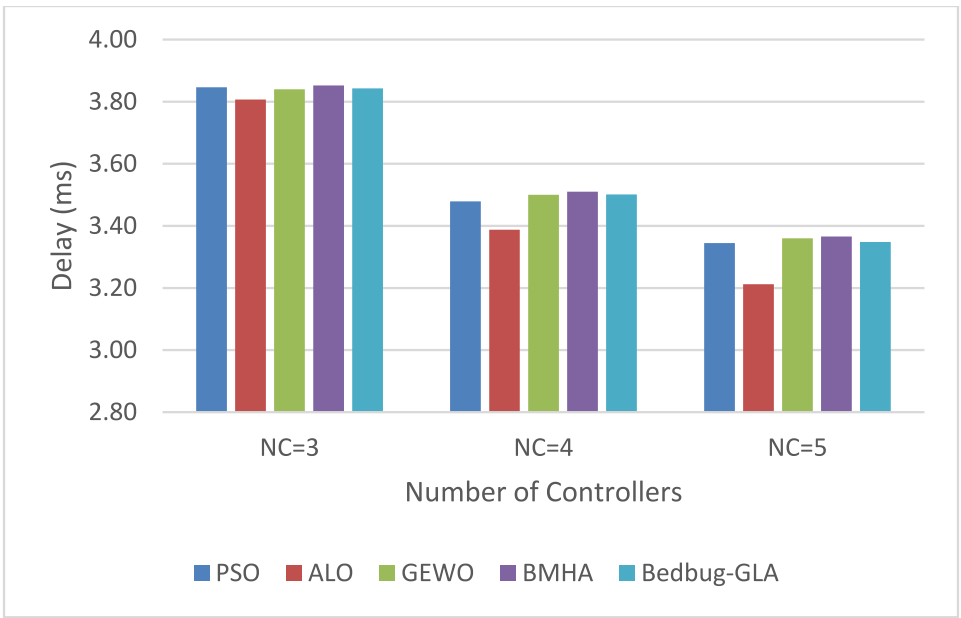


Fig. 23 End-to-end delay in the IRIS topology with different controller numbers (Color figure online)

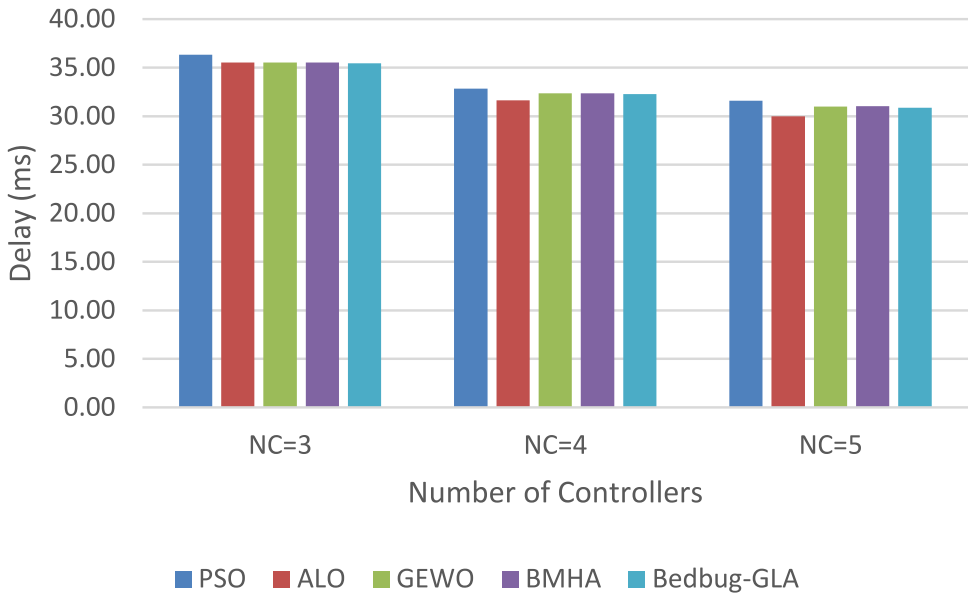


Fig. 24 End-to-end delay in the Colt topology with different controller numbers (Color figure online)

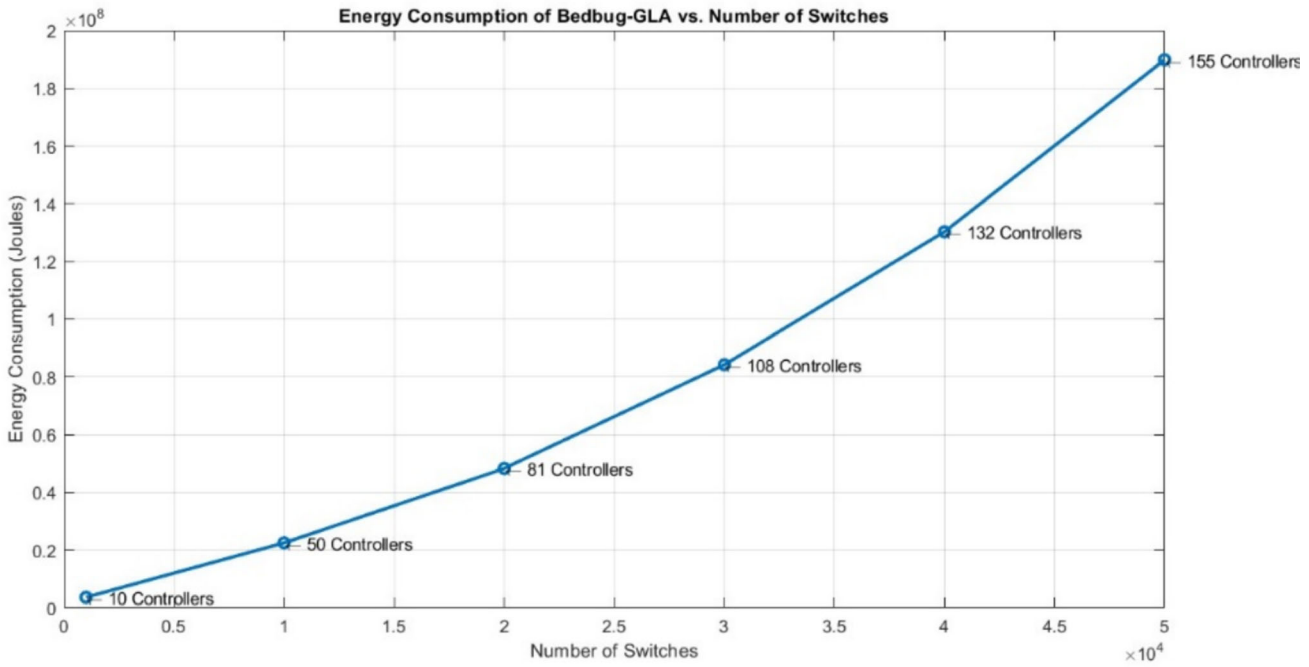


Fig. 25 Evaluating the effect of energy consumption across different network scales

prioritization of energy savings could compromise performance. Therefore, it is essential to strike a balance between these competing objectives. The Bedbug-GLA algorithm aims to address this trade-off by intelligently managing the placement of controllers to optimize both energy efficiency and latency.

In summary, while ALO excels in reducing end-to-end delay by focusing solely on latency parameters, Bedbug-GLA's approach illustrates the complex interplay between energy consumption and latency in network performance. This analysis underscores the importance of considering multiple performance metrics when evaluating controller placement strategies in SDNs.

4.8 Scalability evaluation of bedbug-GLA

In the context of SDN, scalability is crucial for efficiently managing large-scale networks comprising tens of thousands of devices while maintaining performance under varying loads. This section evaluates the performance of

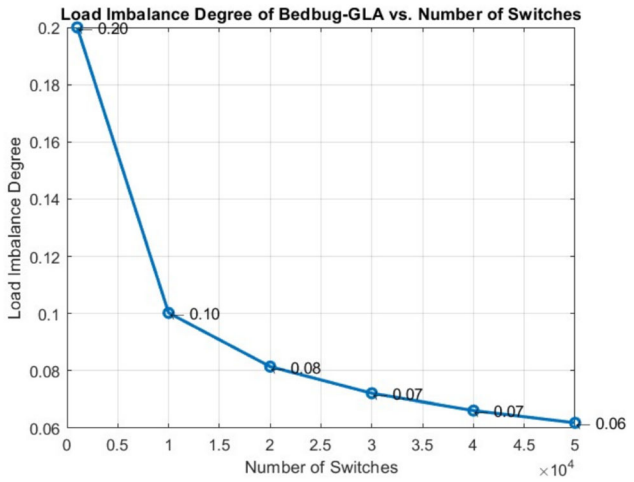


Fig. 26 Evaluating the effect of Load imbalancing degree across different network scales

Bedbug-GLA in terms of energy consumption, load balancing, and latency across different network sizes. It is assumed that each controller can effectively manage approximately 500 switches. A standard 10 Gbps bandwidth per link is assumed in the simulations, allowing the evaluation of Bedbug-GLA's scalability without variable bandwidth constraints.

As illustrated in Fig. 25, the total energy consumption increases from 10,000 kJ for 1000 switches to 19,500 kJ for 50,000 switches. This non-linear increase in energy consumption is attributed to the growing number of controllers and inefficiencies that arise at larger scales. Although Bedbug-GLA optimizes controller placement using the ICLA and the DBMHA, scaling up the network size still results in significant increases in energy usage. For smaller networks (e.g., approximately 1000 switches),

Fig. 27 Evaluating the effect of latency across different network scales

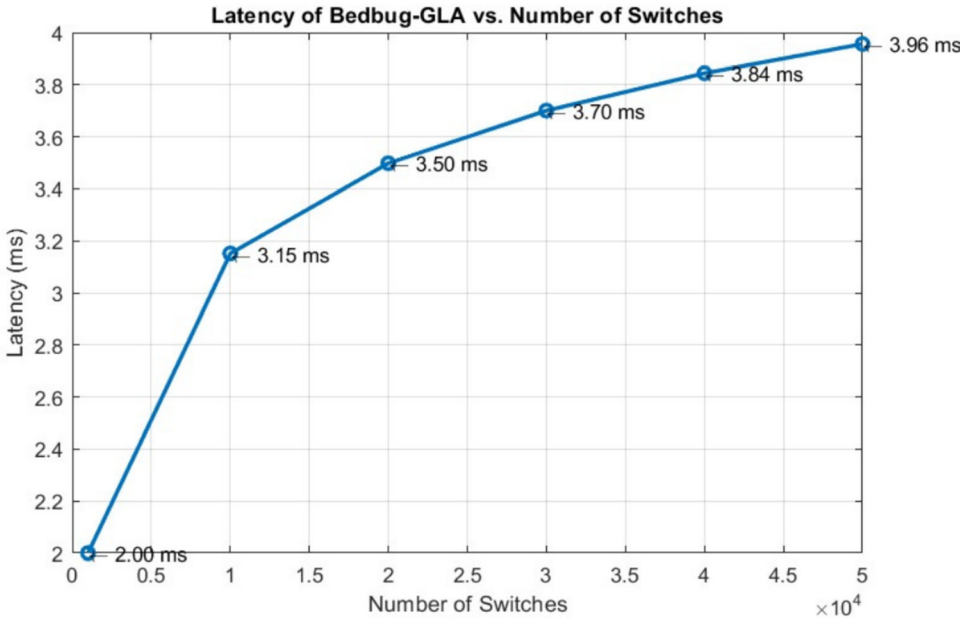
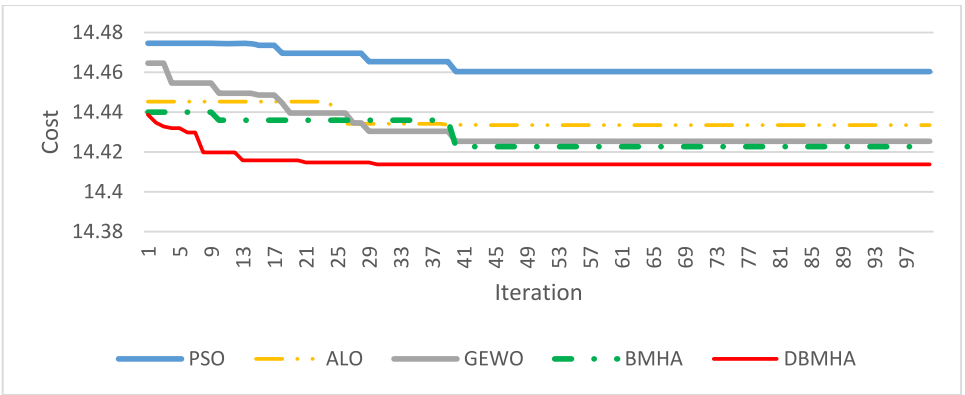


Fig. 28 The convergence rates of the algorithms in the Internet2 OS3E topology (Color figure online)



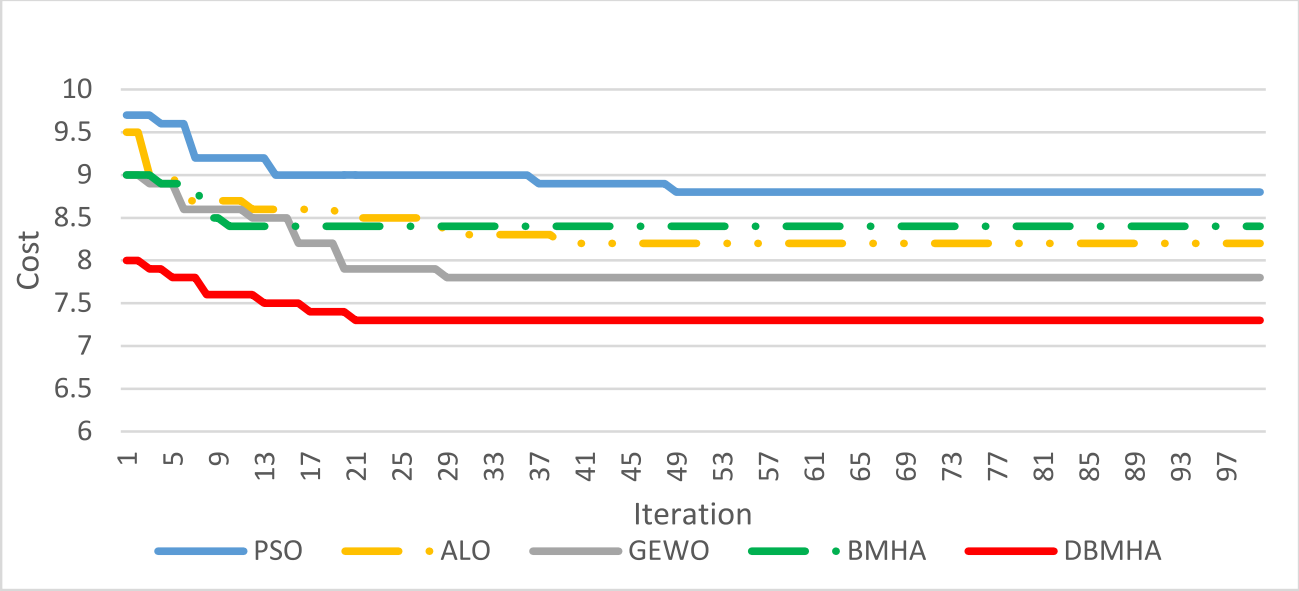
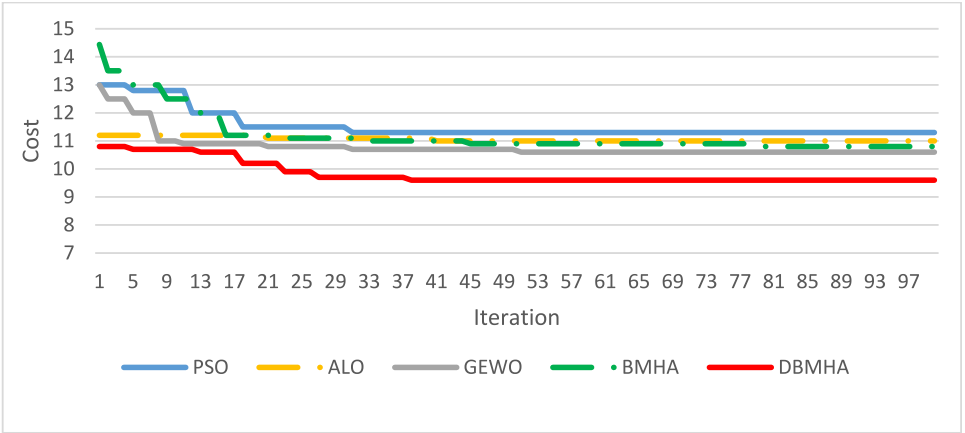


Fig. 29 The convergence rates of the algorithms in the IRIS topology (Color figure online)

Fig. 30 The convergence rates of the algorithms in the Colt topology (Color figure online)



energy consumption remains relatively low due to fewer controllers. However, for large-scale networks (e.g., approximately 50,000 switches), energy consumption grows substantially.

As shown in Fig. 26, the load imbalance degree decreases non-linearly as the number of switches increases. This indicates that Bedbug-GLA effectively enhances load balancing in larger networks. The ICLA and the enhanced metaheuristic algorithm (DBMHA) contribute to this improvement by optimizing controller placement and traffic distribution. Bedbug-GLA demonstrates strong scalability in terms of load balancing, making it suitable for large-scale SDNs.

The average latency experienced by flows increases slightly as the network scales. However, this increase is manageable, rising from 5 ms for 1000 switches to 15 ms for 50,000 switches (Fig. 27). The latency grows logarithmically with the number of switches, indicating that Bedbug-GLA handles scalability well. Logarithmic growth is slower compared to linear or exponential growth. Bedbug-GLA's two-step approach (ICLA for determining controllers and DBMHA for assigning them) ensures that latency remains manageable even as the network size grows. The results suggest that Bedbug-GLA is suitable for large-scale SDNs where low latency is critical.

4.9 The convergence rates

Figures 28, 29, and 30 illustrate the rapid convergence of the algorithms over 100 iterations across various network topologies. The convergence rate charts reveal that the cost associated with PSO is higher than that of the other algorithms, likely due to PSO becoming trapped in local optima. The graphs clearly demonstrate that Bedbug-GLA

consistently outperforms other advanced algorithms by identifying superior solutions in a shorter time frame. This success can be attributed to Bedbug-GLA's efficient search strategies, effective use of heuristics, and its adeptness at navigating complex and dynamic network environments.

The use of chaotic maps in Bedbug-GLA enhances the diversity of the initial population, resulting in lower costs from the first iteration. Chaotic dynamics facilitate more effective exploration by generating a wide range of search points quickly, which is particularly beneficial in complex, non-linear problems. Additionally, chaotic maps help balance exploration and exploitation phases, leading to improved convergence towards global optima.

Additionally, the integration of GA operators within Bedbug-GLA, combined with the enhancement of initial population diversity through chaotic maps, offers several benefits and boosts its performance. GAs are well-known stochastic search methods effective for a wide range of optimization problems. By incorporating these operators, Bedbug-GLA enhances its global exploration and exploitation capabilities, avoiding local optima and improving its convergence rate. This leads to faster and more efficient solution discovery. The hybrid approach leverages the strengths of multiple methods, making Bedbug-GLA more robust and adaptable to complex, nonlinear problems, allowing it to optimize efficiently across various domains.

5 Conclusion

The Controller Placement Problem involves determining an optimal number of network controllers, their placements, and the assignments of these controllers to switches. Previous studies have not adequately addressed the joint optimization of multiple critical factors, including energy consumption, load balancing, latency, and the identification of an appropriate NCs within their methodologies. While some research has concentrated on specific aspects—such as optimizing the NCs while improving latency or enhancing a subset of objectives—there remains a notable gap in the literature regarding a comprehensive approach that simultaneously tackles energy efficiency, load balancing, latency, and controller count.

This paper aims to fill the existing gap by employing a hybrid approach that integrates reinforcement learning algorithms with metaheuristics. A novel version of reinforcement learning is introduced alongside an enhanced version of the BMHA, termed Bedbug-GLA. Bedbug-GLA employs an ICLA to determine the optimal NCs and utilizes the BMHA to enhance performance and speed by incorporating genetic operators and chaotic maps. These genetic operators help avoid local optima traps and facilitate the assignment of controllers to switches more effectively. The ITZ dataset and the Internet2 OS3E topology are used to evaluate the proposed method, with results showing improvements ranging from 4 to 18% in the maximum load of controllers, about 16% to 69% improvement in the overload of congested controllers, and 4% to 20% in energy usage compared to PSO, ALO, GEWO, and BMHA. Bedbug-GLA's performance is comparable to PSO and ALO in terms of latency, but it uniquely balances energy consumption and latency as a multi-objective method. Unlike PSO and ALO, which only improve latency without considering energy consumption, Bedbug-GLA may exhibit slightly higher latency but offers significantly lower energy consumption, providing a balanced approach to optimizing both energy efficiency and latency.

In future research, enhancing the reliability of Bedbug-GLA will involve incorporating robustness measures capable of withstanding network failures. This will include testing in edge environments, such as networks with high variability in node connectivity, to assess the resilience of the algorithm under extreme conditions. The proposed Bedbug-GLA is a hybrid of reinforcement learning and metaheuristics. Due to its metaheuristic structure in the controller assignment step, it may find near-optimal solutions but may not always find the optimal answer. Therefore, in the future, efforts to address this limitation by using more advanced optimization techniques will be necessary. Additionally, the integration of advanced optimization techniques, such as deep reinforcement learning and hybrid evolutionary algorithms, will be explored to further improve performance in the context of the CPP. Extensive comparative studies will be conducted to evaluate the efficacy of these emerging optimization algorithms in complex network environments, alongside other tailored approaches.

Author contributions All authors contributed to the study's conception and design. The main idea was by Maedeh Abedini Bagha. Material preparation, data collection, and analysis were performed by Mohammad Sadegh Sirjani. The first draft of the manuscript was written by Ali Maleki and Amir Pakmehr then Ali Ghaffari and Ali Asghar Pour Haji Kazem commented and on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Data availability The ITZ dataset is used in this research that is available on http://www.topology-zoo.org/archived_datasets.html.

The simulation files are available from the corresponding author on reasonable request.

Declarations

Competing interests The authors declare no competing interests.

References

1. Abderrahmane, A., Drid, H., Behaz, A.: A survey of controller placement problem in SDN-IoT network. Int. J. Netw. Distrib. Comput. (2024). https://doi.org/10.1007/s44227-024-00035-y
2. Jafarian, T., et al.: Security anomaly detection in software-defined networking based on a prediction technique. Int. J. Commun. Syst. **33**(14), e4524 (2020). https://doi.org/10.1002/dac.4524
3. Arzo, S.T., et al.: Msn: a playground framework for design and evaluation of microservices-based SDN controller. J. Netw. Syst. Manage. **30**, 1–31 (2022). https://doi.org/10.1007/s10922-021-09631-7
4. Hu, T., et al.: SEAPP: a secure application management framework based on REST API access control in SDN-enabled cloud environment. J. Parallel Distrib. Comput. **147**, 108–123 (2021). https://doi.org/10.1016/j.jpdc.2020.09.006
5. Darwish, T., Alhaj, T.A., Elhaj, F.A.: Controller placement in software defined emerging networks: a review and future directions. Telecommun. Syst. **88**(1), 1–33 (2025)
6. Singh, A.K., et al.: Heuristic approaches for the reliable SDN controller placement problem. Trans. Emerging Telecommun. Technol. **31**(2), e3761 (2020). https://doi.org/10.1002/ett.3761
7. Gong, J., Rezaeipanah, A.: A fuzzy delay-bandwidth guaranteed routing algorithm for video conferencing services over SDN networks. Multimedia Tools Appl. (2023). https://doi.org/10.1007/s11042-023-14349-6
8. Jafarian, T., et al.: SADM-SDNC: security anomaly detection and mitigation in software-defined networking using C-support vector classification. Computing **103**, 641–673 (2021). https://doi.org/10.1007/s00607-020-00866-x
9. Jafarian, T., et al.: A survey and classification of the security anomaly detection mechanisms in software defined networks. Clust. Comput. **24**(2), 1235–1253 (2021). https://doi.org/10.1007/s10586-020-03184-1
10. Fan, Y., Wang, L., Yuan, X.: Controller placements for latency minimization of both primary and backup paths in SDNs. Comput. Commun. **163**, 35–50 (2020). https://doi.org/10.1016/j.comcom.2020.09.001
11. Singh, G.D., et al.: A novel framework for capacitated SDN controller placement: balancing latency and reliability with PSO algorithm. Alex. Eng. J. **87**, 77–92 (2024). https://doi.org/10.1016/j.aej.2023.12.018
12. Khojand, M., et al.: Controller placement in SDN using game theory and a discrete hybrid metaheuristic algorithm. J. Supercomput. **80**(5), 6552–6600 (2024). https://doi.org/10.1007/s11227-023-05709-y
13. Lu, C., et al.: Human-robot collaborative scheduling in energy-efficient welding shop. IEEE Trans. Industr. Inf. (2023). https://doi.org/10.1109/TII.2023.3271749
14. Mojez, H., et al.: Controller placement issue in software-defined networks with different goals: a comprehensive survey. J. Supercomput. (2024). https://doi.org/10.1007/s11227-024-06230-6
15. Guo, J., et al.: Static placement and dynamic assignment of SDN controllers in LEO satellite networks. IEEE Trans. Netw. Serv. Manage. **19**(4), 4975–4988 (2022). https://doi.org/10.1109/TNSM.2022.3184989
16. Shirmarz, A., Ghaffari, A.: Performance issues and solutions in SDN-based data center: a survey. J. Supercomput. **76**(10), 7545–7593 (2020). https://doi.org/10.1007/s11227-020-03180-7
17. Shirmarz, A., Ghaffari, A.: Taxonomy of controller placement problem (CPP) optimization in software defined network (SDN): a survey. J. Ambient. Intell. Humaniz. Comput. **12**(12), 10473–10498 (2021). https://doi.org/10.1007/s12652-020-02754-w
18. Abderrahmane, A., Drid, H.: Enhancing control placement in SDN-IoT networks using the Louvain algorithm and betweenness-centrality. IEEE Access. (2024). https://doi.org/10.1109/ACCESS.2024.3479916
19. Firouz, N., et al.: A hybrid multi-objective algorithm for imbalanced controller placement in software-defined networks. J. Netw. Syst. Manage. **30**(3), 51 (2022). https://doi.org/10.1007/s10922-022-09650-y
20. Pakmehr, A., Gholipour, M., Zeinali, E.: ETFC: Energy-efficient and deadline-aware task scheduling in fog computing. Sustain. Comput. Inf. Syst. **43**, 100988 (2024)
21. Ibrahim, A.A., et al.: Reliability-aware swarm based multi-objective optimization for controller placement in distributed SDN architecture. Dig. Commun. Netw. **10**(5), 1245–1257 (2024)
22. Isong, B., et al.: Comprehensive review of SDN controller placement strategies. IEEE Access **8**, 170070–170092 (2020)
23. Bagha, M.A., et al.: ELA-RCP: an energy-efficient and load balanced algorithm for reliable controller placement in software-defined networks. J. Netw. Comput. Appl. **225**, 103855 (2024)
24. Heller, B., Sherwood, R., McKeown, N.: The controller placement problem. ACM SIGCOMM Comput. Commun. Rev. **42**(4), 473–478 (2012). https://doi.org/10.1145/2377677.2377767
25. Bagha, M.A., et al.: Improving delay in SDNs by metaheuristic controller placement. Int. J. Ind. Electron. Control Optim. (2022). https://doi.org/10.22111/IECO.2022.42638.1436
26. Srisamarn, U., Pradittasnee, L., Kitsuwan, N.: Resolving load imbalance state for SDN by minimizing maximum load of controllers. J. Netw. Syst. Manage. **29**(4), 46 (2021). https://doi.org/10.1007/s10922-021-09612-w
27. Sahoo, S.K., et al.: An improved moth flame optimization algorithm based on modified dynamic opposite learning strategy. Artif. Intell. Rev. **56**(4), 2811–2869 (2023). https://doi.org/10.1007/s10462-022-10218-0
28. Hu, Y., et al.: The energy-aware controller placement problem in software defined networks. IEEE Commun. Lett. **21**(4), 741–744 (2016). https://doi.org/10.1109/LCOMM.2016.2645558
29. Pradeesshma, T., et al.: An effective of bio-inspired multiverse optimization based controller placement in software defined networks environment. Int. J. Inf. Technol. (2025). https://doi.org/10.1007/s41870-025-02434-y
30. Rezvani, K., Gaffari, A., Dishabi, M.R.E.: The bedbug metaheuristic algorithm to solve optimization problems. J. Bionic Eng. **20**(5), 2465–2485 (2023)
31. Esnaashari, M., Meybodi, M.R.: Irregular cellular learning automata. IEEE Trans. Cybern. **45**(8), 1622–1632 (2014)
32. Ateya, A.A., et al.: Chaotic salp swarm algorithm for SDN multi-controller networks. Eng. Sci. Technol. Int. J. **22**(4), 1001–1012 (2019). https://doi.org/10.1016/j.jestch.2018.12.015
33. Wang, G., et al.: An effective approach to controller placement in software defined wide area networks. IEEE Trans. Netw. Serv. Manage. **15**(1), 344–355 (2017). https://doi.org/10.1109/TNSM.2017.2785660
34. Liu, S., et al. NCPSO: A solution of the controller placement problem in software defined networks. In: Algorithms and Architectures for Parallel Processing: 15th International

Conference, ICA3PP 2015, Zhangjiajie, China, November 18–20, 2015, Proceedings, Part III 15. Springer (2015)
35. Li, Y., Sun, W., Guan, S.: A Multi-controller deployment method based on PSO algorithm in SDN environment. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). IEEE (2020)
36. Vizarreta, P., et al.: Assessing the maturity of SDN controllers with software reliability growth models. IEEE Trans. Netw. Serv. Manage. **15**(3), 1090–1104 (2018). https://doi.org/10.1109/TNSM.2018.2848105
37. Oliveira, T.F., Xavier-de-Souza, S., Silveira, L.F.: Improving energy efficiency on SDN control-plane using multi-core controllers. Energies **14**(11), 3161 (2021). https://doi.org/10.3390/en14113161
38. Ali, J., Roh, B.-H.: An effective approach for controller placement in software-defined internet-of-things (SD-IoT). Sensors **22**(8), 2992 (2022). https://doi.org/10.3390/s22082992
39. Babbar, H., Rani, S.: PUAL-DBSCP: personalized ubiquitous adaptive learning for density-based splitting controller placement in software-defined networks. Comput. Hum. Behav. **154**, 108135 (2024)
40. Zhang, T., et al.: The role of the inter-controller consensus in the placement of distributed SDN controllers. Comput. Commun. **113**, 1–13 (2017). https://doi.org/10.1016/j.comcom.2017.09.007
41. Ramya, G., Manoharan, R.: Enhanced optimal placements of multi-controllers in SDN. J. Ambient. Intell. Humaniz. Comput. **12**, 8187–8204 (2021). https://doi.org/10.1007/s12652-020-02554-2
42. Killi, B.R., Rao, S.V.: Poly-stable matching based scalable controller placement with balancing constraints in SDN. Comput. Commun. **154**, 82–91 (2020). https://doi.org/10.1016/j.comcom.2020.02.053
43. Kazemian, M.M., Mirabi, M.: Controller placement in software defined networks using multi-objective antlion algorithm. J. Supercomput. (2022). https://doi.org/10.1007/s11227-021-04109-4
44. Naseri, A., Ahmadi, M., PourKarimi, L.: Placement of SDN controllers based on network setup cost and latency of control packets. Comput. Commun. **208**, 15–28 (2023)
45. Hemagowri, J., Selvan, P.: Demming regressive multiobjective dragonfly optimized controller placement in SDN environment. Int. J. Nonlinear Anal. Appl. **13**(1), 1747–1761 (2022). https://doi.org/10.22075/IJNAA.2022.5789
46. Li, C., et al.: Deep reinforcement learning based controller placement and optimal edge selection in SDN-based multi-access edge computing environments. J. Parallel Distrib. Comput. **193**, 104948 (2024)
47. Chen, J., et al.: A review of vision-based traffic semantic understanding in ITSs. IEEE Trans. Intell. Transp. Syst. (2022). https://doi.org/10.1109/TITS.2022.3182410
48. VB., Glen.: Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry. Princeton University Press, Princeton (2013)
49. Killi, B.P.R., Rao, S.V.: Controller placement in software defined networks: a comprehensive survey. Comput. Netw. **163**, 106883 (2019). https://doi.org/10.1016/j.comnet.2019.106883
50. Ma, X., et al.: Real-time assessment of asphalt pavement moduli and traffic loads using monitoring data from built-in sensors: optimal sensor placement and identification algorithm. Mech. Syst. Signal Process. **187**, 109930 (2023). https://doi.org/10.1016/j.ymssp.2022.109930
51. Yang, J., et al. Characterizing and modeling of large-scale traffic in mobile network. In: 2015 IEEE Wireless Communications and Networking Conference (WCNC). IEEE (2015)
52. Xiong, B., et al.: Performance evaluation of OpenFlow-based software-defined networks based on queueing model. Comput. Netw. **102**, 172–185 (2016). https://doi.org/10.1016/j.comnet.2016.03.005
53. Mohanty, S., Sahoo, B.: Metaheuristic algorithms for capacitated controller placement in software defined networks considering failure resilience. Concurr. Comput. Pract. Exp. **36**(24), e8254 (2024)
54. Topa, P., Wąs, J.: New trends in complex collective systems. Journal of Comput. Sci. **5**, 35 (2017). https://doi.org/10.1016/j.jocs.2017.05.020
55. Thathachar, M.A., Sastry, P.S.: Varieties of learning automata: an overview. IEEE Trans. Syst. Man Cybern. B Cybern. **32**(6), 711–722 (2002). https://doi.org/10.1109/TSMCB.2002.1049606
56. Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Awni Hammouri, V.B., Prasath, S.: Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. Information **390**(10), 1–39 (2019). https://doi.org/10.3390/info10120390
57. Dantu, R., et al.: Internet2 open science, scholarship and services exchange (2004)
58. Telecom. colt.net. http://www.colt.net/oracleUCM/groups/public/documents/digitalasset/colt_011195.pdf
59. Knight, S., Nguyen, H.X., Falkner, N., Bowden, R., Roughan, M.: The internet topology zoo. IEEE J. Sel. Areas Commun. **29**(9), 1765–1775 (2011)

Mohammad Sadegh Sirjani received his B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad, Iran, in 2024. During his undergraduate studies, he worked as a software engineer at leading technology companies, gaining practical experience that complemented his academic pursuits. This industry exposure shaped his research interests, leading him to explore topics in the Internet of Things (IoT) and heuristic algorithms for scheduling. Currently, he is pursuing a Ph.D. in Computer Science at the University of Texas at San Antonio, where his research focuses on IoT and Tiny AI, aiming to advance efficient and intelligent systems for resourceconstrained environments.

Ali Maleki is an undergraduate student in Computer Engineering at Islamic Azad University, Shiraz Branch, expected to graduate in September 2026. His research interests include algorithmic approaches in medical sciences, optimization of reinforcement learning algorithms for resource management in cloud computing environments, and applications of cloud computing in big data management.

Amir Pakmehr holds a Ph.D. in Computer Engineering from the Islamic Azad University, Qazvin Branch (2025), and received his M.Sc. in Computer Engineering from the Islamic Azad University, Tabriz Branch (2014). His research mainly focuses on Fog Computing, IoT, task scheduling, and optimization algorithms. With a solid academic background in wireless sensor networks and energy-aware systems, Dr. Pakmehr has contributed to several peer-reviewed journals and conferences indexed by SCI. His recent work includes research on deep reinforcement learning for task offloading, energyefficient scheduling, and DDoS detection techniques in IoT networks. Alongside his academic endeavors, he has translated technical books and actively participated in international workshops and exhibitions.

Maedeh Abedini Bagha received her B.Sc., M.Sc., and Ph.D. degrees in Computer Engineering (Software) from Islamic Azad University, Zahedan, Tabriz, and Urmia, Iran, in 2007, 2013, and 2024, respectively. She is currently an Assistant Professor at the Roshdiyeh Higher Education Institute and the Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Iran. Her research interests include Machine Learning Optimization Problems, and Metaheuristic Algorithms.

Ali Ghaffari received his BSc, MSc and Ph.D. degrees in computer engineering from the University of Tehran and IAU (Islamic Azad University), TEHRAN, IRAN in 1994, 2002 and 2011 respectively. Dr. Ghaffari has been featured among the World's Top 2% Scientists List in computer science, according to a conducted study by US-based Stanford University in 2020, 2021, 2023, and 2024 and Top 1% Scientists List in computer science, according to Clarivate analytics in 2022, 2023, and 2024. As an professor of computer engineering, his research interests are mainly in the field of software defined network (SDN), Wireless Sensor Networks (WSNs), Mobile Ad Hoc Networks (MANETs), Vehicular Ad Hoc Networks (VANETs), networks security and Quality of Service (QoS). He has published more than 200 international conference and reviewed journal papers. He has served as a reviewer for some high-ranked journal such as IEEE transaction on mobile computing, IEEE/ACM Transactions on Networking, IEEE Transactions on Network and Service Management, Applied Soft Computing, Ad Hoc networks, Future Generation Computer System (FGCS), Journal of Ambient Intelligent and Humanized Computing (AIHC) and computer networks.

Ali Asghar Pour Haji Kazem received his B.Sc., M.Sc., and Ph.D. degrees in Computer Engineering (Software Engineering) from the University of Isfahan, Shahid Beheshti University, and Islamic Azad University (Science and Research Branch), respectively. He is currently an Assistant Professor in the Software Engineering Department at Istinye University, Istanbul, Türkiye, and has previously served at the Tabriz Branch of Islamic Azad University, Iran. Dr. Pour Haji Kazem has authored and co-authored over 50 research articles in reputable journals and international conference proceedings. His research interests include Distributed Systems, Cloud Computing, Data Science, Machine Learning, Computational Intelligence, Optimization Problems, Evolutionary Computing, and Database.