Computer Networks 281 (2026) 112207

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

ROCDSG: a routing optimization framework for DCN

Qingjie Lin [a,b], Shuwu Chen [a,b], Haihui Xie [a,b], Tarik Taleb [c], Yu Luo [a,b], Zhaogang Shu [a,b,*]

[a] The Computer and Information College, Fujian Agriculture and Forestry University, Fuzhou, China
[b] Engineering Research Center of Smart Sensing and Agricultural Chip Technology, Fujian Province University, Fuzhou, 350002, China
[c] Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany

ARTICLE INFO

Keywords:
SDN
DCN
Louvain algorithm
Firefly algorithm
Routing awareness

ABSTRACT

In recent years, the deep integration of Software-Defined Networking (SDN) and data center networks (DCNs) has provided a programmable foundation for global resource scheduling and path optimization. However, the rapid growth of data-intensive applications has posed a fundamental trade-off in DCNs among Quality of Service (QoS) assurance, load balancing, energy consumption control, and routing computational complexity under dynamic traffic fluctuations: enforcing low latency and strong balancing often incurs substantial global solving overhead, whereas lightweight routing reduces computational cost but tends to cause traffic concentration and degrade energy control performance, making it difficult to consistently guarantee QoS under hotspot and bursty traffic. To address this challenge, this paper proposes ROCDSG, a routing optimization framework based on the collaboration of dynamic subnet partitioning and gateway deployment, aiming to achieve coordinated improvements in QoS, load, and energy objectives with controllable computational complexity. Specifically, a Louvain-based dynamic subnet partitioning mechanism is first introduced to adaptively aggregate network nodes under traffic, load, and energy constraints, thereby shrinking the search space via structural dimensionality reduction. Next, a Mixed-Integer Nonlinear Programming (MINLP) model is formulated for gateway deployment to optimize border-gateway configuration for inter-subnet forwarding and is efficiently solved using the proposed Solution Space-Optimized Firefly Algorithm (SOFA). Finally, a latency-aware hierarchical routing algorithm is designed based on the resulting subnet-gateway skeleton to construct low-latency and stable paths for both intra- and inter-subnet traffic. Simulation results show that, compared with the next-best algorithm, ROCDSG reduces the end-to-end average latency by 25.2%, decreases the average hop count and execution time by 14.4% and 25.6%, respectively, while effectively maintaining balanced load distribution and energy consumption.

1. Introduction

With the rapid growth of data-driven applications such as cloud computing, edge computing, and artificial intelligence, data center networks (DCNs), the core carriers of digital infrastructure, are experiencing massive real-time traffic [1]. As performance requirements shift from bare connectivity to intelligent services, scenarios such as interactive online applications and large-scale distributed training demand millisecond-level latency and dynamic resource orchestration [2]. However, under highly dynamic conditions, traditional distributed control struggles to obtain a global view; routing decisions rely on local state and thus often yield suboptimal cross-network paths [3]. Static subnet partitioning and fixed resource provisioning at the rack or cluster granularity cannot adapt to spatiotemporal traffic fluctuations, leading to queueing latencies and load surges [4]. In addition, energy management in existing data center networks is still dominated by device-level, slow-timescale, and coarse-grained control, which is difficult to coordinate with traffic dynamics and QoS requirements, thereby limiting the overall effectiveness of energy control. Studies indicate that data centers already account for approximately 2% of global energy consumption and that this share continues to increase [5]. Therefore, relying solely on traditional paradigms, including distributed routing based on local information, static domain partitioning with fixed configurations, and device-level energy control, often makes it difficult to achieve an effective trade-off among stringent QoS requirements, load balancing, and energy consumption control.

SDN provides the foundation for this goal. An SDN controller can obtain the global network state and install programmable policies,

* Corresponding author.
E-mail addresses: qjlin@fafu.edu.cn (Q. Lin), chenshuwu@fafu.edu.cn (S. Chen), xiehh@fafu.edu.cn (H. Xie), talebtarik@gmail.com (T. Taleb), yuluo@fafu.edu.cn (Y. Luo), zgshu@fafu.edu.cn (Z. Shu).

https://doi.org/10.1016/j.comnet.2026.112207
Received 30 June 2025; Received in revised form 6 February 2026; Accepted 7 March 2026
Available online 19 March 2026

theoretically enabling global path and resource orchestration [6]. In practice, however, two key constraints remain: on the one hand, heterogeneous data center workloads require routing to simultaneously satisfy millisecond-level latency guarantees, load balancing, and energy efficiency [7]; on the other hand, in ultra-large-scale topologies, multi-objective constraints inflate computational complexity, making centralized optimization slow to converge [8]. To address these challenges, previous research has explored multiple directions but still shows notable limitations: one line of work [9,10] employs integer programming to jointly optimize energy and load, achieving high-quality solutions but suffering from poor scalability and high computational cost in large networks; another line [11,12] relies on static partitioning and fixed gateways to simplify path computation, which reduces complexity but fails to track traffic evolution and thus degrades to suboptimal routing. This indicates that, under dynamic workloads and large-scale constraints, neither relying solely on computationally expensive global optimization models nor relying solely on static structural simplifications can remain effective in the long run. Therefore, there is an urgent need for a comprehensive routing optimization framework that can online and jointly adjust subnet partitioning, gateway deployment, and path selection while maintaining computational scalability and solution stability as the network scales.

To this end, this paper proposes a Routing Optimization Framework based on the Collaboration of Dynamic Subnet Partitioning and Gateway Deployment, called ROCDSG, which transforms the global network view into executable decisions through a collaborative closed loop. First, dynamic subnet partitioning is performed under multi-dimensional constraints on traffic, load, and energy consumption, partitioning the network into logical subdomains to achieve structural dimensionality reduction and significantly shrink the path search space. Second, to address the fact that cross-subnet traffic (i.e., end-to-end flows whose source and destination reside in different subnets) typically relies on border gateways and inter-subnet links for inter-subnet forwarding and is prone to traffic aggregation at gateways-thereby evolving into a key performance bottleneck that increases end-to-end latency and exacerbates congestion propagation-we conduct elastic gateway deployment within each subnet. This deployment jointly considers latency, energy consumption, and load balancing, and is obtained using a lightweight solution strategy to ensure scalability and stable performance in large-scale settings. Finally, based on the resulting subnet-gateway skeleton, a latency-aware routing scheme is executed to construct end-to-end paths following minimum-latency and stability criteria, thereby achieving both service quality and path efficiency. The overall framework follows a “sense-orchestrate-enforce” closed loop: sensing collects and abstracts global network states; orchestration generates coordinated decisions on subnet partitioning, gateway deployment, and routing; and enforcement installs and updates forwarding rules via SDN to enable online operation and iterative optimization. Specifically, the main contributions of this paper are summarized as follows:

1. To reduce the complexity of routing computation, a dynamic subnet partitioning mechanism based on the Louvain algorithm is developed. The mechanism achieves adaptive subnet partitioning of data center networks under traffic, load, and energy consumption constraints through modular community detection and hierarchical iterative optimization strategies.
2. Given the critical role of gateways in cross-subnet communication, a Mixed-Integer Nonlinear Programming (MINLP) model is constructed for optimizing the gateway deployment scheme with the objective of minimizing the latency, energy consumption, and load variance of gateway deployment. In order to enhance the solution efficiency, the Solution Space-Optimized Firefly Algorithm (SOFA), a method proposed in this paper, is employed to solve the model.
3. After the subnet partitioning and gateway deployment are completed, a latency-aware hierarchical routing algorithm is designed and adopted to construct the end-to-end optimal request path.

The remainder of the paper is organized as follows. Section II reviews related work, Section III gives the network architecture, Section IV describes the design and realization of ROCDSG, and Section V shows the experimental results. Finally, Section VI summarizes the paper and discusses possible future research directions.

2. Related work

Routing optimization design for data center networks, as a core topic to enhance network performance, has become a focus of attention in both academia and industry. In recent years, there has been much literature on its theoretical exploration and engineering practice.

In terms of traditional optimization models, researchers commonly adopt linear programming models to solve multi-constraint optimization problems and combine heuristic strategies to reduce the solution complexity. In the literature [13], He et al. proposed a flow management strategy that integrated energy consumption optimization and load balancing. They achieved this by modeling the multipath routing and device activation minimization problems separately. Heuristic algorithms were then designed to provide an efficient solution. Pathan et al. [9] further extended the idea by constructing a Multi-Objective Integer Linear Programming (MILP) model that considered flow priority and proposed the PEMA and PEDL dual algorithms based on energy consumption threshold screening and load standard deviation evaluation, which focused on energy consumption minimization and load balancing optimization, respectively. Naeem et al. [10] developed a GPU-accelerated parallel routing framework for the centralized control potential of SDN architectures, modeling the multi-constraint QoS problem as a maximum-flow and minimum-cost model and utilizing greedy algorithms to achieve parallelism in path computation. Jiawei et al. [14], on the other hand, designed a hierarchical scheduling mechanism to prioritize traffic through message ToS fields, with high-priority flows using a composite cost function based on latency and packet loss rate to drive dynamic routing, and low-priority flows implementing load balancing based on residual bandwidth. In addition, Lu et al. [15] proposed a policy-based minimum energy consumption heuristic algorithm, which balanced energy-saving goals and QoS satisfaction rate through traffic classification and device start/stop policies, especially showing advantages in high-load scenarios. Although these routing optimization methods can achieve a compromise between computational efficiency and solution quality through rule-driven full optimization, they are always limited by the inherent complexity of the problem and are difficult to cope with the real-time decision-making needs of large-scale networks.

With the development of artificial intelligence, data-driven methods provide new solutions for routing optimization. In terms of intelligent optimization, Sanchez et al. [16] introduced Deep Reinforcement Learning (DRL) to construct a multi-objective reward function, with link states and queue depths as inputs, to train routing strategies autonomously. Wang et al. [17] integrated a prediction mechanism into this approach. They used a long and short-term memory network to predict traffic trends and constructed a hybrid MILP-DDPG framework to achieve dynamic adjustment of bandwidth allocation and device energy consumption policies. Kim et al. [18] proposed an offline pre-training framework based on the M/M/1/K queueing model to address the problem of slow convergence in DRL online training. This framework dynamically generated link weight optimization policies by simulating network traffic, which were then configured to the switches by SDN controllers. The goal was to balance latency and packet loss rate, significantly reducing retraining overhead during topology changes. Chen et al. [19] focused on modeling network state spatio-temporal features and proposed a proximal policy optimization algorithm based on an attention mechanism and spatio-temporal correlation to optimize SDN routing. The authors extracted the link temporal evolution and node spatial dependency characteristics by fusing gated loop cells and graph attention networks. They introduced a jump-connectivity mechanism to enhance state

Table 1
Comparison of the proposed framework with related works.

Literatures	Dynamic subnet partitioning	Gateway deployment	QoS	Load balancing	Energy consumption	Routing efficiency	Major limitations
Pathan et al. [9]	×	×	√	√	√	×	Large-scale networking
Naeem et al. [10]	×	×	√	×	√	√	Dynamic load fluctuation
Wang et al. [17]	×	×	×	×	√	×	High computational complexity
El-Hefnawy et al. [11]	×	×	√	×	×	√	Fixed number of subnets
Wang et al. [21]	√	√	√	√	×	√	Energy correlation
Dev et al. [12]	×	√	×	√	√	√	Insufficient QoS guarantee
Matni et al. [23]	×	√	√	×	×	×	Lack of scalability
Zhang et al. [25]	×	√	√	×	×	×	Long computation time
ROCDSG(Ours)	√	√	√	√	√	√	

• In this table, "√" means that this work contain this feature, whereas "×" means that it does not contain.

characterization, enabling DRL agents to adaptively generate routing policies in multiple scenarios. Although these optimization methods can generate dynamic routing policies and adapt to topology changes in various scenarios, they are essentially black-box optimizations, relying too heavily on parameter tuning and lacking stability guarantees for practical needs.

In order to reduce the size of the network and improve the routing efficiency, many studies have taken the approach of subnet partitioning to optimize the routing mechanism. El-Hefnawy et al. [11] used the box covering method combined with K-means clustering to partition the network topology and integrated the ant colony algorithm with the variational operator to optimize dynamic routing on the partitioned topology. Tiwari et al. [20] proposed a novel subnet-based partitioned SDN architecture, which leveraged the wildcard matching feature of tristate content-addressable memory for prefix-based IP address matching. This approach significantly reduced switch storage requirements and minimized switch-to-controller communication. In the literature [21], Wang et al. designed a QoS and data privacy routing-aware protocol for 5G Industrial Internet of Things (IIoT). They used an improved information mapping algorithm along with load and latency constraints to divide optimal subnets but neglected the impact of node energy consumption. Zhang et al. [3] also proposed a box-coverage-based routing algorithm, dividing the network into a fractal-dimensional reorganization with a subnet hierarchy. They verified its applicability to hyperscale SDN on the Mininet platform. However, as in the other studies [11], the number of subnet divisions was predefined, which was not dynamically adjusted according to changes in the structure of the network. These methods also failed to consider the impact of network load, energy consumption, and other factors on QoS. On the other hand, the authors [12] focused on energy efficiency in wireless sensor networks. They generated the initial cluster structure using the K-means algorithm and dynamically adjusted the cluster head (gateway) position based on the Gray Wolf optimization algorithm. This method aimed to achieve energy-balanced partitioning of heterogeneous networks, with residual energy of the nodes, distance from the base station, and load as optimization objectives.

There also have been many recent studies on routing optimization based on gateway deployment. Torkzabn et al. [22] proposed a Mixed-Integer Linear Programming model for the joint optimization of satellite gateway deployment and SDN controller placement. This model achieved efficient solutions under complex constraints through problem decomposition and submodular function transformation. They then proposed a routing protocol for terrestrial networks, considering both total cost and average latency. Matni et al. [23] developed a model to optimize both cost expenditure and QoS for dynamic IoT scenarios. They achieved joint optimization of the number of gateways and deployment locations by grouping IoT devices with high similarity. In the literature [24], Cao et al. proposed a satellite gateway deployment method under capacity constraints, combining an enumeration algorithm and a heuristic greedy algorithm to minimize runtime and find the optimal gateway combination. In the literature [21], the authors applied a deep deterministic policy gradient algorithm to construct a local gateway deployment model and achieved gateway optimization through a four-dimensional state space, a discretized action space, and a weighted latency ratio reward function. Zhang et al. [25] reconfigured the gateway collaboration paradigm by introducing a federated learning mechanism. They used border routers as lightweight agent nodes, extracted cross-network consistency abstract features using a cryptographic feature alignment technique, and implemented global federated model updates through asynchronous gradient aggregation. This approach circumvented in-domain sensitive data exposure while exchanging only KB-level overlapping parameter gradients. While these routing protocols can optimize gateway deployment, they do not fully account for the correlation of device energy consumption. Additionally, most of the current research ignores the queue management latency and protocol stack processing latency of the devices themselves, which can easily cause fluctuations in the end-to-end latency of the data center network.

In summary, although prior work on routing optimization for data center networks has advanced path selection, subnet partitioning, and gateway deployment, common shortcomings persist. To clarify our positioning, Table 1 contrasts ROCDSG with representative approaches in terms of key features and principal limitations. The comparison shows that large-scale deployment, computational complexity, dynamic load variability, energy-performance coupling, and stringent QoS requirements remain the dominant bottlenecks. Consequently, there is a clear need for a comprehensive routing framework that combines global scalability with dynamic adaptiveness to meet the optimization demands of modern data center networks.

Accordingly, we present ROCDSG, a framework that performs multi-constraint-driven dynamic subnet partitioning and elastic gateway placement to coordinate network structure and traffic distribution, while employing latency-aware routing to adjust forwarding paths and enhance service quality. Through this strategy, ROCDSG effectively reduces computational complexity and enhances system scalability, thereby achieving a balanced trade-off between performance and efficiency in large-scale network environments.

3. Network architecture

The ROCDSG proposed in this paper adopts the decoupled design of the control plane and data plane to realize intelligent routing of heterogeneous traffic through multi-module collaboration. The overall network architecture is shown in Fig. 1, and the main body of the architecture consists of an edge access plane, a programmable data plane, a centralized control plane, and an application plane. In the access plane, multiple data sources such as base stations, user terminals, video monitor equipment, enterprise organizations, and satellite ground stations aggregate raw traffic to the underlying SDN switch through standardized data transmission interfaces; the data plane consists of intelligent switches that support programmable functions, including flow table matching, packet encapsulation and decapsulation, as well as network telemetry data collection and other key functions. The control plane mainly deploys four core modules, namely, state awareness, dynamic subnet partitioning, elastic gateway deployment, and route management, in which the state awareness module obtains real-time network data and builds a

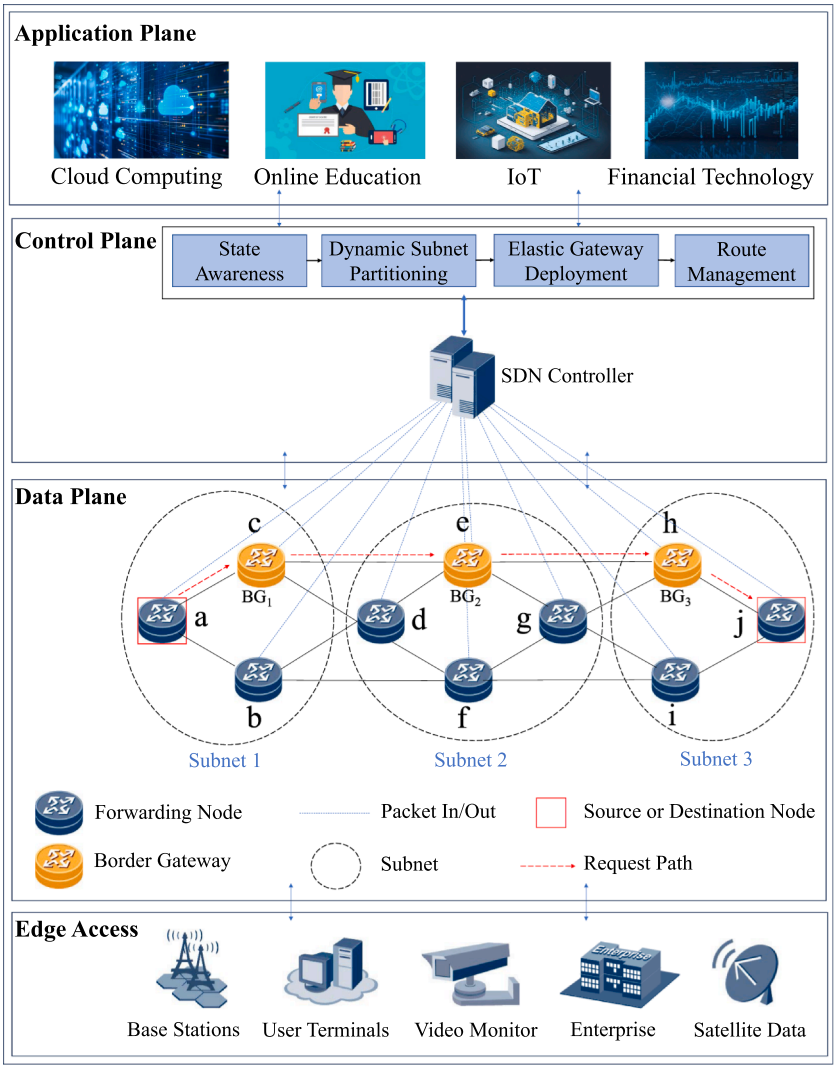

Fig. 1. Overall network architecture.

topology map with weights by continuously listening to packet-in messages and active LLDP packet probing. The dynamic subnet partitioning module generates logically separated virtual subnets based on the Louvain algorithm, targeting constraints such as traffic, load, and energy consumption. The term "subnet" in this paper does not refer to a group of end hosts aggregated under an IP prefix. Instead, under the SDN controller's global view, it represents a logical forwarding domain dynamically aggregated through multi-dimensional constraints. The members of each subnet are forwarding devices (SDN switches or routers, including candidate gateways), forming a logical layer to reduce routing complexity and enable hierarchical scheduling. End hosts and IP prefixes are abstracted and not used as criteria for subnet division. To ensure inter-subnet connectivity, the baseline configuration assumes that each subnet deploys exactly one Boundary Gateway (BG) for inter-subnet relaying. This configuration is implemented by the Elastic Gateway Deployment module, which dynamically selects one BG per subnet based on the partitioning results and optimizes its physical placement using the SOFA algorithm to minimize inter-domain hop count. The route management module finally generates the cross-subnet optimal path policy based on the global view, which is sent down to the data plane via packet-out messages. Taking the data flow from $a \rightarrow j$ the figure as an example, the architecture first identifies the subnets to which the source and destination nodes belong (subnet 1 and subnet 3). It then selects the $a \rightarrow c$ path to the border gateway $BG1$ based on the latency-aware algorithm in subnet 1. At the cross-subnet stage, the shortest path $c \rightarrow e \rightarrow h$ from $BG1$ to $BG3$ is obtained, and finally, the path routes to the destination node j in subnet 3. This ultimately forms the overall path $a \rightarrow c \rightarrow e \rightarrow h \rightarrow j$, thereby reducing routing complexity. With the two-layer optimization mechanism of subnet autonomy and cross-network collaboration, this architecture can effectively support high concurrency, low latency, and differentiated quality of service requirements in scenarios such as cloud computing, online education, and IoT.

This paper uses a connected graph $G(V, E)$ to represent the network composition in ROCDSG. Here, V denotes the set of nodes, and $E \subseteq V \times V$ denotes the set of edges. Specifically, each node $v_i \in V$ represents a device (i.e., an SDN switch or gateway), while each edge $e_i \in E$ represents a communication link between devices. For example, e_{ab} is the connection between vertex a and b (i.e., $e_{ab} \in E$, where $a, b \in V$ and $a \neq b$). As shown in Fig. 1, $V = \{a, b, c, d, e, f, g, h, i, j\}$ and $E = \{e_{ab}, e_{ac}, e_{cd}, e_{bd}, \ldots\}$. In this paper, it is assumed that the source and destination nodes of each communication in the graph are unique and that SDN switches or gateways are homogeneous devices with the same functions.

In an SDN-enabled data center network, the data flow is first sent to the SDN switch, and then the SDN controller calculates the path based on the global view, routing the data flow from the source node to the

Table 2
Main notations.

Notation	Definition
V	the set of network nodes.
E	the set of network links.
v_i	network device (e.g., SDN switch or gateway).
e_i	link between devices.
F_k	tuple representation of the data flow.
G_b	set of boundary gateways.
$f_{v_i}^T$	total traffic load processed on node v_i.
M_i	set of nodes within the i-th subnet.
C	total system capacity.
$L(M_i)$	total load of the i-th subnet.
$d_{avg}(v_i)$	average latency of node v_i.
$d_{max}(v_i)$	maximum latency of node v_i.
$d_{avg}(v, g_i)$	average latency from node v to boundary gateway g_i.
$d_{max}(v, g_i)$	maximum latency from node v to boundary gateway g_i.
$d_{avg}(g_i, g_j)$	average latency between gateways g_i and g_j.
$d_{max}(g_i, g_j)$	maximum latency between gateways g_i and g_j.
τ_{intra}	upper bound of node-to-gateway latency.
τ_{inter}	upper bound of gateway-to-gateway latency.
τ_{device}	upper bound of node latency.
E_i^T	total energy consumption within subnet M_i.
y_{ik}	gateway selection variable indicating whether node i is selected as the boundary gateway of subnet k.
$z_{v,g}$	node–gateway binding variable indicating whether node v is associated with gateway g.
$x_{i,j}^f$	path occupancy variable indicating whether flow f uses link (i, j).
r_f	traffic volume of flow f.
C_{ij}	bandwidth capacity of link (i, j).
$E_{\max}$	unified maximum energy consumption allowed for all gateways.
E	actual energy consumption in gateway deployment.
L	load variation in gateway deployment.
C_i	gateway deployment combination.

destination node. Suppose a data flow is represented by a tuple $F_k = \{s_k, d_k, p_k, a_k\}$, where s_k and d_k denote the source and destination nodes of the flow, respectively, and $s_k, d_k \in V$ (with V representing the set of all nodes in the network). p_k represents the size of the data packets and a_k represents the arrival time of the flow. The main symbols used in this paper and their meanings are listed in Table 2.

4. Design and realization of ROCDSG

In this section, the design and realization of the core part of ROCDSG will be elaborated in detail, specifically including dynamic subnet partitioning, elastic gateway deployment, and latency-aware routing.

4.1. Dynamic subnet partitioning

4.1.1. Constraint modeling

The traditional static subnet partitioning approach relies on a fixed topology and a preset traffic model, which was effective in the relatively stable network environment in the early days. However, with the diversified and dynamic changes in data center business requirements, this approach suffers from high load, high latency, high energy consumption, and low resource utilization. Therefore, it is necessary to introduce a subnet partitioning strategy that can dynamically adapt to the business demands and fully consider multiple constraints such as load balancing, energy consumption overhead and communication latency.

Subnet partitioning should allow the sum of traffic to be as balanced as possible across subnets. Assuming that the set of border gateways is denoted by $G_b = \{g_1, g_2, \ldots, g_k\}$, $f_{v_i}^{(\mathrm{in})}$ denotes the traffic handled by node v_i inside the subnet, $f_{v_i}^{(\mathrm{s})}$ denotes the cross-subnet traffic sent by node v_i to the border gateway g_i, and $f_{v_i}^{(\mathrm{r})}$ denotes the cross-subnet traffic received by node v_i from the border gateway g_i, then the total traffic $f_{v_i}^{(\mathrm{T})}$ on node v_i is:

$$f_{v_i}^{(T)} = f_{v_i}^{(in)} + f_{v_i}^{(s)} + f_{v_i}^{(r)} \tag{1}$$

The entire subnet partitioning needs to be satisfied:

$$\left| \sum_{v_i \in M_i} f_{v_i}^{(T)} - \sum_{v_j \in M_j} f_{v_j}^{(T)} \right| \le \epsilon^f \tag{2}$$

$$\sum_{i=1}^{K} \sum_{v_i \in M_i} f_{v_i}^{(T)} \le \alpha C \tag{3}$$

where $M_i = \{M_1, M_2, \ldots, M_K\}$ represents the set of nodes within the i-th subnet, ϵ^f is the traffic balancing factor, and α is the safety margin factor, representing the maximum allowable usage ratio of the system capacity C (where $0 < \alpha < 1$). In the network, each link has its load capacity, denoted as $l_{(v_i,v_j)}$, and each node has the load that it needs to process, denoted as $c(v_i)$. The load is measured by the bandwidth utilization of the links or nodes. Assuming that in the time step t, links and nodes will have upcoming load values $l_{(v_i,v_j)}{}'$ and $c(v_i)'$, then the total load of the subnet M_i is

$$L(M_i) = \sum_{v_i \in M_i} \left(c(v_i) + c(v_i)' \right) + \sum_{v_i, v_j \in M_i, v_i \neq v_j} \left(l_{(v_i,v_j)} + l'_{(v_i,v_j)} \right) \tag{4}$$

Measuring the load balancing constraints between subnets is defined as:

$$\left| L(M_i) - L(M_j) \right| \le \lambda \tag{5}$$

where λ represents the load balancing factor. The smaller it λ is, the more balanced the load is between subnets, and vice versa. Since each subnet will deploy a boundary gateway, and each node will have processing latency and queuing latency, three types of latency are considered in this paper: node latency, node-to-boundary gateway latency, and gateway-to-gateway latency. Node latency is typically related to the load and processing capacity of the node. We define the average latency $d_{\mathrm{avg}}(v_i)$ and maximum latency $d_{\max}(v_i)$ for a node as:

$$d_{avg}(v_i) = \frac{1}{|M_i|} \sum_{v_i \in M_i} \frac{c(v_i)}{\mu(v_i)} \tag{6}$$

$$d_{max}(v_i) = max \left\{ \frac{c(v_i)}{\mu(v_i)} | v_i \in M_i \right\} \tag{7}$$

where $\mu(v_i)$ represents the processing capability of node v_i, i.e., the amount of data it can process per unit time. We define the average latency $d_{\text{avg}}(v, g_i)$ and the maximum latency $d_{\max}(v, g_i)$ from a node within the subnet to the boundary gateway g_i as follows:

$$d_{avg}(v, g_i) = \frac{1}{|M_i|} \sum_{v_i \in M_i} d(v_i, g_i) \tag{8}$$

$$d_{max}(v, g_i) = max\{d(v_i, g_i) \mid v_i \in M_i, g_i \in G_b\} \tag{9}$$

$d(v_i, g_i)$ represents the link latency from node v_i to boundary gateway g_i. For notational clarity, we let v_i denote a node belonging to the i-th subnet, i.e., $v_i \in M_i$. In all expressions where such terms $\sum_{v_i \in M_i}$ appear, the summation is taken over the nodes within a *fixed* subnet i, and thus does not imply an additional summation over the subnet index i. We define the average latency $d_{\text{avg}}(g_i, g_j)$ and the maximum latency $d_{\max}(g_i, g_j)$ between gateways g_i and g_j as follows:

$$d_{avg}(g_i, g_j) = \frac{1}{K(K-1)} \sum_{i=1}^{K} \sum_{\substack{j=1 \\ j \neq i}}^{K} d(g_i, g_j). \tag{10}$$

$$d_{max}(g_i, g_j) = max\{d(g_i, g_j) \mid g_i, g_j \in G_b, i \neq j\} \tag{11}$$

where $d(g_i, g_j)$ represents the communication latency between boundary gateways g_i and g_j of subnets M_i and M_j, respectively. To ensure the efficient operation and quality of service of the network system, the maximum latency from nodes within the subnet to the boundary gateway and the maximum latency between gateways should be considered. Therefore, we define the constraints that the maximum intra-subnet latency from nodes to the border gateway and the maximum inter-gateway latency do not exceed a given threshold. These constraints are used for subsequent re-triggering decisions in subnet partitioning and serve as modeling inputs to the gateway deployment module. The specific formulations are given as follows:

$$d_{max}(v, g_i) \leq \tau_{intra} \tag{12}$$

$$d_{max}(g_i, g_j) \leq \tau_{inter} \tag{13}$$

When a request flow uses a path, each node (i.e., switch or gateway) and link incurs a certain amount of energy consumption when it is enabled, while packets also consume a certain amount of energy as bytes are encapsulated and de-encapsulated during transmission. Therefore, the energy consumption within the subnet can be defined as follows:

$$E_i^T = E_s + \sum_{i \in M_i} \left(k * E_t(v_i)\right) \tag{14}$$

where E_i^T represents the total energy consumption within subnet M_i, E_s represents the energy consumption of nodes and links within the subnet M_i (including node and link startup energy consumption, as well as other static energy consumption such as processing and sensing energy), and $k * E_t(v_i)$ represents the energy consumed by node v_i when transmitting k bytes of data. The energy consumption between subnets must satisfy the following:

$$\left|E_i^T - E_j^T\right| \leq \varepsilon^e, \quad \forall i \in M_i, \forall j \in M_j \tag{15}$$

where ε^e represents the energy consumption balance constraint factor, with a smaller value being more favorable. This helps to reduce the risk of local overload or underload caused by energy consumption imbalance, thus lowering the risk of system failure and link congestion. At the same time, for any subnet M_i, its corresponding subgraph $G_{M_i} = (V_{M_i}, E_{M_i})$, where $V_{M_i} = M_i \cap V$ and $E_{M_i} = \{(u, v) \in E \mid u, v \in V_{M_i}\}$, must satisfy the connectivity requirement, i.e., all nodes within the subnet must be in a connected state to avoid the occurrence of isolated nodes.

4.1.2. Improved Louvain algorithm based on constraints

In order to achieve subnet partitioning in dynamic network environments, this paper introduces multidimensional constraints such as traffic, load, and energy consumption into the Louvain algorithm. The core idea of the Louvain algorithm is to assess the quality of community partitioning by maximizing the modularity degree, and the formula for the modularity degree Q is as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m}\right] \delta(c_i, c_j) \tag{16}$$

where A_{ij} represents the edge weight between node i and node j, k_i and k_j denote the degrees of nodes i and j, respectively (thus $k_i k_j$ is their product), m is the total weight of all edges, and $\delta(c_i, c_j)$ is the indicator function, which equals 1 if nodes i and j belong to the same community and 0 otherwise. The improved Louvain algorithm in this paper optimizes subnet partitioning not only by maximizing modularity as in the traditional algorithm but also by incorporating constraints such as traffic balance, node and link load distribution, and energy consumption. In the execution of the algorithm, each node is initially treated as an independent community. Then, iterations of node movement and merging operations are performed, where constraint conditions are introduced in each iteration to adjust the partitioning. Ultimately, a subnet partitioning scheme that satisfies multiple constraints is formed. Fig. 2 illustrates an example of how to partition a network of seven nodes into two subnets. From the node and edge connectivity relations and weights in subgraph(1), the total network weight m is found to be 44. Initially, each node is treated as an independent community. Then, each node is moved to the community of its neighboring node, and the modularity increment is computed. The formula for calculating the modularity increment is

$$\Delta Q = \frac{1}{2m}\left(k_{i,in} - \frac{\sum_{tot} k_i}{m}\right) \tag{17}$$

In the formula, $k_{i,\text{in}}$ represents the weight between node i and the target community, and $\sum_{\text{tot}} k_i$ represents the product of the weights of all connections in the target community and all connections of node i. When the modularity increment $\Delta Q > 0$, it indicates that the node can be moved to the target community. Using subgraph (1) as an example, node a is a neighbor of nodes b and c. We consider moving node a to the community of node b. In this case, $k_{i,\text{in}} = 8$, and $\sum_{\text{tot}} k_i = 14 * 13$. Therefore, the modularity increment is: $\Delta Q = \frac{8 - \frac{14*13}{44}}{88} \approx 0.044$. Since $\Delta Q > 0$, node a can be merged with another node b to form an initial subnet, and this community is treated as a super-node, the edge weights in the graph are updated to build a new weighted graph [26]. Next, for the updated graph, the remaining nodes continue the process of node movement and community merging until modularity converges. Finally, all subnets are checked to see whether they satisfy Eqs. (2), (3), (5), (15), and the definition of connectivity constraints. If all constraints are satisfied, the final subnet partitioning is completed. It is worth noting that, due to variations in network conditions and constraint threshold settings, the improved algorithm may produce infeasible partitioning results in certain scenarios. In such cases, the algorithm returns an "infeasible" flag, which is then handled by the system-level mechanism.

At the system level, a control cycle is defined for subnet partitioning in this work. At the beginning of each cycle, the controller executes the improved Louvain algorithm based on the real-time global network state to generate the logical subnet partition for the current period, which serves as the input for subsequent gateway deployment and routing optimization. During the current cycle, the membership of each subnet remains unchanged; re-partitioning is triggered only at the boundary of the next control cycle when a significant deviation in the network state is detected (i.e., when any key constraint is violated; see Eqs. (2), (3), (5), (12), (13), and (15)), thereby preventing structural oscillations within the cycle. When a subnet partitioning result is deemed infeasible within a control period, the system reuses the feasible partitioning from the previous period. If infeasibility continues for several periods, a predefined fallback strategy (e.g., relaxing related constraint thresholds) is activated to avoid prolonged stagnation.

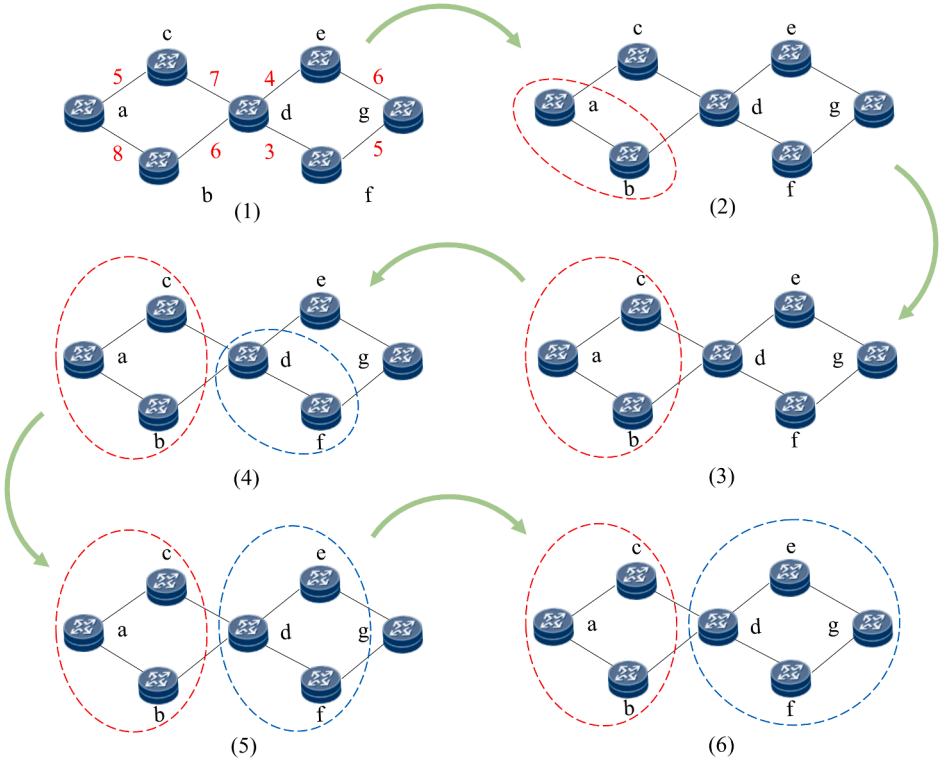

Fig. 2. Dynamic subnet partitioning process.

Algorithm 1 presents the process of executing the improved Louvain algorithm based on constraints. The algorithm first applies the Louvain algorithm to partition the network graph and obtain a candidate subnet partitioning result (shown in lines 4-9). It then enters an iterative retry procedure: for each candidate subnet, the algorithm computes the total flow, capacity, load, and energy consumption statistics (shown in lines 11-13) and checks whether the subnet violates any predefined multi-dimensional constraints, including capacity limits, load balancing, energy balancing, and connectivity requirements (shown in lines 14-19). If any subnet fails to satisfy these constraints, the partitioning result of the current iteration is discarded, and the algorithm re-runs the Louvain method in the next iteration to generate a new community partitioning (shown in lines 16 and 23; lines 7-9 in the subsequent round). This process continues until one of the following termination conditions is met: if all subnets satisfy the constraints, the final subnet partitioning result is returned (shown in lines 20-22); otherwise, if no feasible partition is found after reaching the maximum number of iterations I, the algorithm returns "No feasible partition result found."

In Algorithm 1, the time complexity of the Louvain algorithm itself is $O(m)$, where m denotes the number of edges in the network. The time complexity of each round of constraint checking is $O(n)$, and the maximum number of retries for community reorganization is I. Therefore, the overall time complexity of Algorithm 1 is $O(I \cdot (m+n))$.

4.2. Elastic gateway deployment

4.2.1. Deployment target modeling

To balance routing hierarchy efficiency and model tractability in a multi-subnet structure, this study models each subnet as containing exactly one Boundary Gateway (BG) serving as the inter-subnet communication node. To jointly characterize gateway selection, node binding, and path utilization, a unified set of binary decision variables is defined as follows. Each candidate node $i \in M_k$ within the subnet k can potentially serve as its gateway.

We define a binary selection variable $y_{ik} \in \{0,1\}$ to indicate whether node i is chosen as the boundary gateway of subnet k:

$$y_{ik} = \begin{cases} 1, & \text{if node } i \text{ is selected as the BG of subnet } k, \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

The association between nodes and boundary gateways is represented by the binary variable $z_{v,g} \in \{0,1\}$, where $z_{v,g} = 1$ indicates that node v aggregates or forwards traffic through the boundary gateway g:

$$z_{v,g} = \begin{cases} 1, & \text{if node } v \text{ is associated with BG } g, \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

To capture flow transmission and link utilization in the routing layer, a binary variable $x_{ij}^f \in \{0,1\}$ is introduced, where $x_{ij}^f = 1$ if flow f traverses a link (i,j) and 0 otherwise:

$$x_{ij}^f = \begin{cases} 1, & \text{if flow } f \text{ traverses link } (i,j), \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

To further determine the optimal placement of the boundary gateway in each subnet, it is necessary to balance latency, load, and energy consumption to ensure that the selected gateway location meets performance requirements while avoiding potential single-point bottlenecks. Consequently, the gateway placement problem can be formulated as a multi-constrained optimization model, whose objectives and constraints jointly capture the trade-offs among QoS, network efficiency, and energy consumption.

Intuitively, this model aims to minimize the communication latency between intra-subnet nodes and their boundary gateways, as well as the inter-gateway transmission latency, while reducing the deployment cost of boundary gateways. This formulation reflects the controller's global decision-making logic under dynamic subnet structures. The first optimization objective focuses on latency minimization:

Algorithm 1: Improved Louvain Algorithm based on constraints.

Input: Network graph containing switches and edges $G(V, E)$, parameters containing ε^f, α, λ, and ε^e, maximum number of retries I

Output: Partition result P

```
P ← ∅;
iter ← 0;
while iter < I do
    if P is empty then
        P ← Community partition via the Louvain algorithm on G;
    end
    else
        P ← Re-run Louvain algorithm on G;
    end
    valid ← True;
    for each community S ∈ P do
        Compute total flow f(S), total capacity C(S), total load L(S), total energy consumption E(S);
    end
    for each community S ∈ P do
        if S violates Eqs. (2), (3), (5), (15), or the connectivity constraint then
            valid ← False;
            break;
        end
    end
    if valid then
        return P;
    end
    iter ← iter + 1;
end
return "No feasible partition result found";
```

$$Minimize\left(\sum_{k=1}^{3}\frac{\omega_k * d_{avg}(k)}{d_{max}(k)}\right) \tag{21}$$

where $d_{\text{avg}}(k)$ represents the average latency for each dimension, with $k \in \{1, 2, 3\}$ corresponding to gateway latency (i.e., node latency), link latency from node to gateway, and inter-gateway latency, respectively. $d_{\max}(k)$ represents the maximum latency for each dimension and ω_k is the weight associated with the corresponding latency term. The second optimization objective minimizes the total energy consumption of boundary gateway deployment while achieving load balancing:

$$Minimize(\omega_4 * E_{g_i} + \omega_5 * L) \tag{22}$$

where E_{g_i} represents the actual energy consumption of the gateway deployment, L represents the load difference in the gateway deployment, and ω_4 and ω_5 are the weights for the actual energy consumption and load difference, respectively. Specifically, the definitions of E_{g_i} and L are as follows:

$$E_{g_i} = E_s(g_i) + E_t(g_i) \tag{23}$$

$$L = \sum_{g_i \in G_b}\left(L_{g_i} - \frac{\sum_{g_i \in G_b} L_{g_i}}{|G_b|}\right)^2 \tag{24}$$

where $E_s(g_i)$ represents the static energy consumption of the boundary gateway, $E_t(g_i)$ represents the transmission energy consumption of the boundary gateway, L_{g_i} represents the actual load of the boundary gateway, $|G_b|$ represents the number of elements in the set of boundary gateways, and $\sum_{g_i \in G_b} L_{g_i}$ represents the total load of the set of boundary gateways. Based on the above two objectives, we propose a joint optimization formula as follows:

$$Minimize\ Z = \left(\sum_{k=1}^{3}\frac{\omega_k * d_{avg}(k)}{d_{max}(k)} + (\omega_4 * E_{g_i} + \omega_5 * L)\right) \tag{25}$$

This optimization formulation is subject to the following constraints:

s.t. (12)–(13)

$$\sum_{k=1}^{3}\omega_k + \omega_4 + \omega_5 = 1 \tag{26}$$

$$d_{max}(v_i) \le \tau_{device} \tag{27}$$

$$\sum_{f \in F} r_f x_{ij}^f \le C_{ij}, \quad \forall (i,j) \in E \tag{28}$$

$$\frac{L_{g_i}}{L(M_i)} \le \delta, \quad \forall i \in M_i, \forall g_i \in G_b \tag{29}$$

$$\sum_{i \in M_k} y_{ik} = 1, \quad \forall k \tag{30}$$

$$E_{g_i} \le E_{max} \tag{31}$$

$$P_{uv}^{ij} \ge 1, \quad \forall u \in M_i, \forall v \in M_j, i \ne j \tag{32}$$

$$\forall k \in P(i \to j), k \notin M_i \tag{33}$$

$$\sum_{g \in G_b} z_{v,g} = 1, \quad z_{v,g} \le y_{gk}, \quad \forall v \in M_i, g \in G_b \tag{34}$$

Eq. (26) is used to assign different weights among the metrics, and the sum of these weights should be 1. If only the weight ω_4 is set to 1, the objective function will focus only on minimizing the energy consumption of the gateway deployment while ignoring the impact of other metrics. Therefore, a reasonable trade-off must be made between multiple factors such as latency, energy consumption, and load to ensure that the optimization scheme integrates these key factors. Eqs. (12), (13), and (27) set an upper limit on the maximum latency of the gateway itself, the maximum latency of nodes to the gateway, and the maximum latency between gateways to ensure that the latency of intra-subnet and inter-subnet communication is controlled within reasonable limits. Eq. (28) constrains the total amount of cross-network traffic not to exceed the upper bound of the link bandwidth, where r_f denotes the traffic volume and C_{ij} represents the bandwidth capacity of link (i, j). Eq. (29) limits the load share of each gateway to prevent individual gateways from becoming performance bottlenecks due to overload. Eq. (30) ensures that each subnet is allowed to select exactly one boundary gateway, maintaining hierarchical independence and structural integrity of the network. Eq. (31), on the other hand, specifies that the energy consumption of each border gateway does not exceed its maximum allowable value, where E_{g_i} denotes the energy consumption of gateway g_i, and $E_{\max}$ is the unified upper bound for all gateways, preventing device failures or downtime caused by excessive energy usage. To further enhance the reliability and connectivity of the network, Eq. (32) requires that each border gateway must be connected to other subnets through at least one valid path. Here, P_{uv}^{ij} indicates whether there exists a feasible path from node u in a subnet M_i to node v in subnet M_j. Eq. (33) further specifies that, for any reachable path between subnets, none of the nodes on the selected path may belong to the node set of the subnet where the border gateway resides. Here, $P(i \to j)$ denotes the set of nodes contained in the selected cross-network path from subnet M_i to subnet M_j, and k represents any intermediate node on the path. Eq. (34) guarantees that each node can be associated with only one selected gateway, with the binding relationship restricted within the same subnet to prevent cross-network coupling. Under sudden traffic surges or overly stringent threshold settings, the above constraints may render the optimization problem infeasible; therefore, we do not assume that the optimization problem is always feasible in all scenarios. During solving, penalty terms are imposed on candidate deployment solutions that violate constraints to guide the search, and solutions that satisfy all constraints are prioritized as final outputs; if no feasible solution exists, the candidate solution with the minimum degree of constraint violation is

selected as the optimal output so as to ensure valid input for subsequent routing optimization.

The proposed joint optimization problem aims to minimize the energy consumption of border gateway deployment and load imbalance while satisfying bandwidth, connectivity, traffic allocation, and energy constraints and jointly considering communication latency. Since the problem involves integer decision variables and nonlinear objectives, it can be formulated as a mixed-integer nonlinear programming (MINLP) model.

Theorem 1. *Our proposed MINLP model is NP-hard.*

Proof. The solution complexity of the MINLP model shows an exponential growth trend with the increase of network size and constraints. To prove that the model belongs to the NP-hard class of problems, we argue by equivalence and complexity analysis with existing classical NP-hard problems [27]. In the MINLP model, bandwidth constraints for cross-network traffic, connectivity requirements for nodes and gateways, and flow control for inter-gateway communication together form a multi-commodity flow problem with capacity constraints. The multi-commodity flow problem, which involves the combinatorial complexity of multi-source and multi-destination path selection and flow allocation, has been rigorously proved to be an NP-hard problem [28]. Since the computational complexity of the MINLP model is similar to that of the multi-commodity flow problem, it is NP-hard in nature. □

It should be noted that the term elastic deployment in this paper does not imply high-frequency or continuous reassignment of boundary gateways. Its core mechanism operates within a defined operation and maintenance configuration window, where the SDN controller performs low-frequency, optimization-based selection among pre-deployed, gateway-capable candidate nodes to determine the boundary gateway for each subnet. The physical locations and hardware configurations of the devices remain unchanged, and re-selection is triggered only when a significant global change is detected. To further avoid unnecessary reconfigurations, a significant global change is defined as an adjustment of the subnet partition structure. When the controller re-executes dynamic subnet partitioning and the newly generated partitioning differs substantially from the previous cycle in topological relationships, the gateway deployment module is automatically re-optimized to determine the most suitable boundary gateway placement within the updated subnets. In other words, boundary gateway deployment is synchronized with subnet re-partitioning: when the subnet structure remains stable, the boundary gateway configuration is preserved; when the subnet structure is redefined, the boundary gateway deployment is updated accordingly. This design requires no hardware migration and aligns with SDN-based operational practices commonly adopted in data centers.

4.2.2. SOFA

Conventional solving methods are often difficult to effectively deal with the MINLP model due to the high computational complexity. For this reason, the Firefly Algorithm (FA) is chosen to solve it in this paper. However, with the expansion of network scale, the solution space of gateway deployment combinations presents a scale of millions, which makes the Firefly Algorithm's running time increase significantly when facing such a large solution space. To solve this problem, this paper proposes the Solution Space-Optimized Firefly Algorithm (SOFA), which first optimizes the solution space to reduce the size of the search space to improve the efficiency of the algorithm.

The optimization process of the solution space is shown in the upper half of the second stage in Fig. 3. Suppose the network topology contains K subnets, where each subnet M_i contains n_k candidate boundary gateways. We generate all possible boundary gateway combinations through the Cartesian product, forming the solution space $C = \{C_1, C_2, \dots, C_N\}$, where $N = \prod_{k=1}^{K} n_k$. Each combination C_i represents a tuple of selected candidate gateways from each subnet, $C_i = \{M_1^i, M_2^i, \dots, M_K^i\}$. For each combination C_i, its performance metrics are defined as follows: load balancing degree f_L, node latency ratio f_v, node-to-gateway latency ratio f_{v-g}, inter-gateway latency ratio f_{g-g}, and energy consumption value f_E. Based on these metrics, a solution space matrix $\mathcal{M} \in \mathbb{R}^{N\times 5}$ is constructed, where each row corresponds to a feature vector of a gateway deployment combination [29], specifically defined as follows:

$$\mathcal{M} = \begin{bmatrix} f_L^{(1)} & f_v^{(1)} & f_{v-g}^{(1)} & f_{g-g}^{(1)} & f_E^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_L^{(N)} & f_v^{(N)} & f_{v-g}^{(N)} & f_{g-g}^{(N)} & f_E^{(N)} \end{bmatrix} \tag{35}$$

In the joint optimization problem, the weights of each index should be dynamically assigned according to the statistical characteristics and intrinsic correlation of the data itself, rather than relying on subjective experience setting. In the following, the process of dynamic allocation of weights will be described in detail. In order to avoid the interference of the difference of the scale on the weight calculation, the matrix $\mathcal{M}$ is firstly normalized:

$$\mathcal{M}_{std}[i,j] = (\mathcal{M} - \mu) \oslash \mathcal{L} \tag{36}$$

where $\mu \in \mathbb{R}^5$ denotes the mean vector of each column, $\mathcal{L} \in \mathbb{R}^5$ is the standard deviation vector of each column, and $\oslash$ denotes element-by-element division by column. Next, the variance of each indicator $\theta_i \in \mathbb{R}^5$ is calculated and normalized to obtain the initial weight of the indicator ω_i:

$$\omega_i = \frac{\theta_i}{\sum_{k=1}^{5} \theta_k}, \quad i = 1, 2, \dots, 5 \tag{37}$$

To prevent overfitting, the minimum weight threshold ω_{min} is set to obtain the adjusted weights ω':

$$\omega_i' = max(\omega_i, \omega_{min}), \quad \omega' = \frac{\omega_i'}{\sum \omega_i'} \tag{38}$$

Then, the original matrix is weighted by the weight vector to obtain the weighted solution space matrix $\mathcal{M}_{wtd} = \mathcal{M} \cdot \mathrm{diag}(\omega')$. Based on the above operation, the Singular Value Decomposition (SVD) method is utilized to extract the principal components of the solution space, thus achieving the compression of the solution space dimension. Specifically, the weighting matrix $\mathcal{M}_{wtd}$ is first subjected to Singular Value Decomposition, and the following results are obtained:

$$\mathcal{M}_{wtd} = U\Sigma V^T \tag{39}$$

where U, Σ and V denote the left singular matrix, diagonal matrix, and right singular matrix, respectively. The matrices $U \in \mathbb{R}^{N\times r}$ and $V \in \mathbb{R}^{5\times r}$ satisfy $U^TU = I$ and $V^TV = I$, and the left singular vectors and right singular vectors are orthogonal vectors. The diagonal elements of the diagonal matrix $\Sigma \in \mathbb{R}^{r\times r}$ satisfy $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq 0$, where σ denotes the diagonal factor of the matrix [30], and its value reflects the degree of importance of the combination. The SVD itself has the following properties:

$$\mathcal{M} = \sigma_1 u_1 x_1^T + \sigma_2 u_2 x_2^T + \cdots + \sigma_q u_q x_q^T \tag{40}$$

Immediately after that, the cumulative variance share of the first k singular values is calculated with the following equation:

$$EV(k) = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \tag{41}$$

By setting the threshold $\rho = 0.9$, the smallest q is selected such that $EV(q) \geq \rho, i.e.$, the first q principal components are retained, and the solution space is projected onto these principal components to obtain the reduced matrix $\mathcal{M}_{rd}$:

$$\mathcal{M}_{rd} = U_{[:,1:q]}\Sigma_{[1:q,1:q]} \tag{42}$$

By the above definition, $\mathcal{M}_{rd}$ retains more than 90% of the information content of the original matrix. In order to filter the highly representative candidate solutions from the reduced solution space, we compute

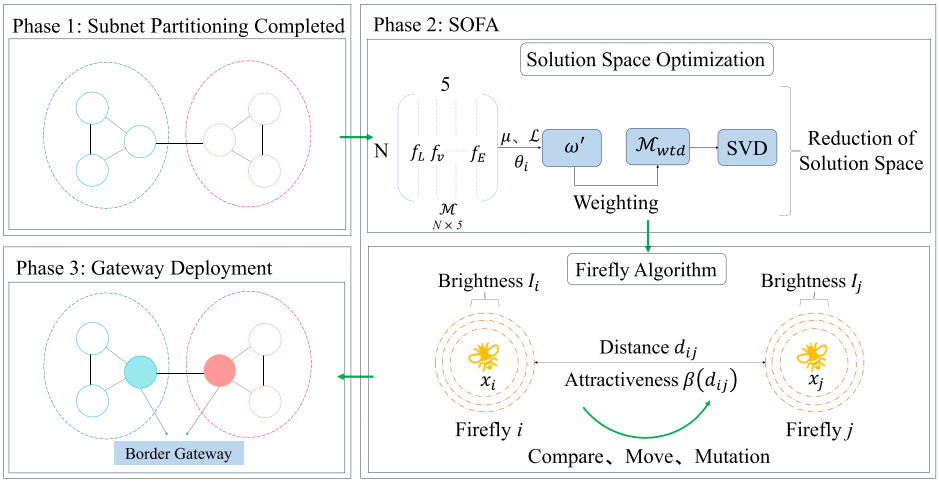

Fig. 3. Basic process of gateway deployment.

its composite score for each row of the reduced matrix $\mathcal{M}_{rd}$:

$$s_i = \sum_{j=1}^{q} \mathcal{M}_{rd}[i,j], \quad i = 1, 2, \ldots, N \tag{43}$$

the score s_i reflects the strength of each combination's projection in the principal component space, and we sort the scores from highest to lowest, selecting the top $K = \varphi$ N candidate combinations as the solution space for the next stage of algorithm execution.

Based on the above solution space optimization results, this paper further adopts the Firefly Algorithm to solve the optimal deployment location of the gateway. The Firefly Algorithm is a biomimetic group intelligence optimization algorithm based on biomimicry, and its core mechanism simulates the group behavioral characteristics of fireflies on summer nights, which transmit information through light signals and collaborate to forage for food [12]. The luminance $I(x)$ of fireflies at a position x is determined by the objective function $F(x)$ and is proportional to it; according to Eq. (25), we define the objective function $F(x)$ of the Firefly Algorithm as

$$F(x) = \left(\sum_{k=1}^{3} \frac{\omega_k * d_{avg}(k)}{d_{max}(k)} + (\omega_4 * E_{g_i} + \omega_5 * L) \right) + P_i \tag{44}$$

P_i in the Eq. (44) denotes the penalty associated with violating the ith constraint term. The attraction between fireflies $\beta(d_{ij})$ is calculated by the following equation:

$$\beta(d_{ij}) = \beta_0 e^{-\gamma d_{ij}^2} \tag{45}$$

where d_{ij} is the distance cost between fireflies i and j, β_0 denotes the maximum attraction when d_{ij} is 0, and γ is the attenuation factor of attraction, which controls the convergence rate of firefly behavior. The Euclidean distance d_{ij} between fireflies is defined by the following equation:

$$d_{ij} = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \tag{46}$$

d is the dimension of the problem, and $x_{i,k}$ and $x_{j,k}$ denote the coordinates of fireflies i and j in the kth dimension, respectively. The position of each firefly is updated according to the firefly that is brighter than it, with the specific update formula:

$$x_i = x_i + \beta_0 e^{-\gamma d_{ij}^2} + (x_j - x_i) + \psi S_k (rand - 0.5) \tag{47}$$

where x_i and x_j denote the position vectors of fireflies i and j, respectively; ψ is a random perturbation factor for controlling the diversity of the solutions; S_k is a scaling factor for the kth heap to ensure that the update amount is adapted to the features in each dimension; and rand denotes a uniform random number in the range of 0 to 1. In order to prevent the algorithm from falling into local optimal solutions during the search process, we introduce a mutation factor [11], which randomly selects a portion of fireflies for mutation operation after each update of the position to improve the global search effect on the gateway deployment combinations. The mutation operation is shown below:

$$x_i = x_i + \eta \cdot (x_{best} - x_i) + \xi \cdot (rand - 0.5) \tag{48}$$

where x_{best} is the brightest individual in the current population, i.e., the gateway deployment combination with the optimal objective function value. η is the variation strength, which controls the degree of individuals' proximity to the optimal solution in the variation operation, and ξ is the variation magnitude, which controls the randomness of the variation operation.

The basic process of gateway deployment is shown in Fig. 3. After completing the subnet partitioning operation (Phase 1), we use the SOFA to perform gateway deployment. Specifically, Phase 2 consists of two steps: solution space optimization and Firefly Algorithm solving. We first optimize and downsize the solution space of candidate gateway deployment combinations to filter out representative candidate solutions and narrow down the search scope. Then, the Firefly Algorithm is applied in the streamlined solution space for solving. The Firefly Algorithm calculates the brightness of each solution according to the objective function and, based on the principle of "bright attracts dark," drives the worse solution to the better solution and enhances the diversity of the solutions through mutation operations. The process is iterated until the convergence condition is satisfied, and finally the optimal gateway deployment combination is obtained (Phase 3).

Algorithm 2 describes the specific process for calculating the fitness value of a gateway deployment combination, which is mainly applied in SOFA (Algorithm 3). In the case of a non-empty set of gateways, the algorithm traverses each gateway in turn to compute its initial fitness value (shown in lines 2–5). Subsequently, the gateway itself and its interactions with other gateways are checked to see if they satisfy the predefined constraints (Eqs. (12), (13), (27)–(34)). For gateways that do not satisfy the constraints, a corresponding penalty term (shown in line 7) is added to the fitness to lower the evaluation score for that gateway. Ultimately, the algorithm returns the combined fitness value for the current gateway deployment combination, which serves as the basis for the subsequent optimization process.

The execution process of SOFA is shown in Algorithm 3. In lines 1–9, the steps of solution space optimization are mainly implemented. Specifically, the algorithm constructs the solution space matrix based on the candidate solutions generated by the Cartesian product and calculates the weight factors in the objective function by normalization and

Algorithm 2: Calculation of the fitness value.

Input: Network graph containing switches and edges $G(V, E)$, gateway deployment scheme G_b
Output: Fitness

```
penalty ← 0;
if G_b ≠ ∅ then
    for g_i ∈ G_b do
        for g_j ∈ G_b, i ≠ j do
            Compute Fitness using Eq. (25);
            if the constraints in Eqs. (12)-(13),(27)-(34) are not satisfied then
                penalty ← penalty + penalty_value;
            end
        end
    end
    Fitness ← Fitness + penalty;
end
return Fitness;
```

Algorithm 3: SOFA.

Input: Number of subnets K, $\{M_1^i, M_2^i, \ldots, M_K^i\}$, minimum weight $\omega_{\min}$, cumulative variance threshold for SVD ρ, candidate solution ratio φ, MaxIter, PopSize, MutationRate
Output: Optimal gateway deployment G_b, weight vector ω'

```
1  C ← CartesianProduct{M_1^i, M_2^i, ..., M_K^i};
2  Compute performance matrix M ∈ R^{N×5}, M_std;
3  ω ← max( Variance(M_std) / ΣVariance(M_std), ω_min );
4  ω' ← ω / Σω;
5  M_wtd ← M · diag(ω');
6  (U, Σ, _) ← SVD(M_wtd);
7  q ← min{q : EV(q) ≥ ρ} and compute M_rd;
8  s ← rowSum(M_rd);
9  Select ⌊φN⌋ solutions based on s;
10 Initialize firefly population {x_1, ..., x_PopSize} from selected candidates ;
11 Compute Fitness for each candidate from Algorithm 2;
12 for i = 1 to PopSize do
13     I_i ← Fitness(x_i);
14 end
15 for t = 1 to MaxIter do
16     for i = 1 to PopSize do
17         for j = 1 to PopSize do
18             if I_j > I_i then
19                 x_i ← x_i + Attraction(x_j, x_i);
20                 Update I_i;
21             end
22         end
23         if Random() < MutationRate then
24             x_i ← x_i + η · (x_best − x_i) + ξ · Random() − 0.5;
25             Update I_i;
26         end
27     end
28 end
29 return x_best, ω', G_b;
```

variance normalization methods. Subsequently, the normalized matrix is weighted and downsized to retain the key feature information, and finally, the Singular Value Decomposition (SVD) is used to determine the number of principal components, generate the low-dimensional approximation matrix, and screen out the most representative candidate solutions as the initial population. Lines 10–26 show the process of solving the optimal gateway deployment combinations using the Firefly Algorithm. In each generation, the migration of individuals to a better solution is driven by two-by-two comparisons between fireflies, and probabilistic double mutation operations are introduced to prevent the algorithm from falling into a local optimal solution. Finally, the algorithm reaches the optimal solution through multiple iterations.

Algorithm 2 contains two nested `for` loops, so its time complexity is $O(n_g^2)$, where $n_g = |G_b|$ denotes the number of gateways in each combination. For Algorithm 3, the time complexity can be expressed as $O(m^K + T \cdot N^2 \cdot n_g^2)$, where m^K comes from the Cartesian product of the candidate solution space, m is the number of candidate gateways for each subnet, and K is the number of subnets. T represents the number of iterations of the Firefly Algorithm, N is the population size (i.e., `PopSize`), and the factor n_g^2 accounts for the cost of calling Algorithm 2 during each fitness update.

4.3. Latency-aware routing

On the basis of dynamic subnet partitioning and elastic gateway deployment, this paper designs a latency-aware hierarchical routing algorithm. The core idea of the algorithm is to dynamically select the routing strategy based on the logical subnet affiliation of the source and destination nodes of a data request. Specifically, if the source node and the destination node are located in the same subnet, the algorithm will directly calculate the end-to-end optimal path based on the link latency weight matrix maintained by the SDN controller in real time so as to ensure the millisecond response of intra-subnet communication. As shown by the purple dotted line in Fig. 4, both node 8 and node 13 belong to subnet 2, so node 8 can quickly find the optimal path to node 13 within the subnet.

If the source and destination nodes belong to different subnets, the algorithm will use a hierarchical optimization mechanism. Within the subnet, routing will be based on the optimized gateway deployment combination, and the optimal path will be determined between nodes and gateways according to the principle of minimum latency [21]; in cross-subnet communication scenarios, packets will be relayed and forwarded through the optimal gateway nodes, and the cross-subnet path selection will still be based on the optimization goal of minimum latency and rely on the SDN global view to achieve the collaboration of the cross-network link states. Ultimately, the optimal intra-subnet path and the cross-subnet relay path are seamlessly integrated to construct a globally optimal end-to-end transmission route, thereby meeting the requirements of low latency and high efficiency in data center networks. As shown in the pink dotted line in Fig. 4, node 8 first calculates and finds the optimal path to gateway 5 in subnet 2; then, gateway 5 locates subnet 3, where node 4 is located according to the cross-subnet routing table, and forwards it across subnets through gateway 10; finally, gateway 10 finds the optimal path to node 4 in subnet 3, completing the global construction of end-to-end paths.

Algorithm 4 describes the execution process of the latency-aware hierarchical routing algorithm. In lines 1-3, the algorithm extracts the source and destination nodes from the request stream and determines whether they belong to the same subnet based on the subnet mapping information. If the source and destination nodes are in the same subnet, the algorithm computes the minimum-latency path between them based on the dynamically updated link-latency weights and directly returns the result (as shown in line 4). If the source and destination nodes belong to different subnets, the algorithm then computes three path segments: the minimum-latency path from the source node to the source gateway, the cross-subnet relay path from the source gateway to the destination gateway, and the minimum-latency path from the destination gateway to the destination node. Subsequently, the algorithm concatenates these three paths and removes duplicate nodes to generate the complete end-

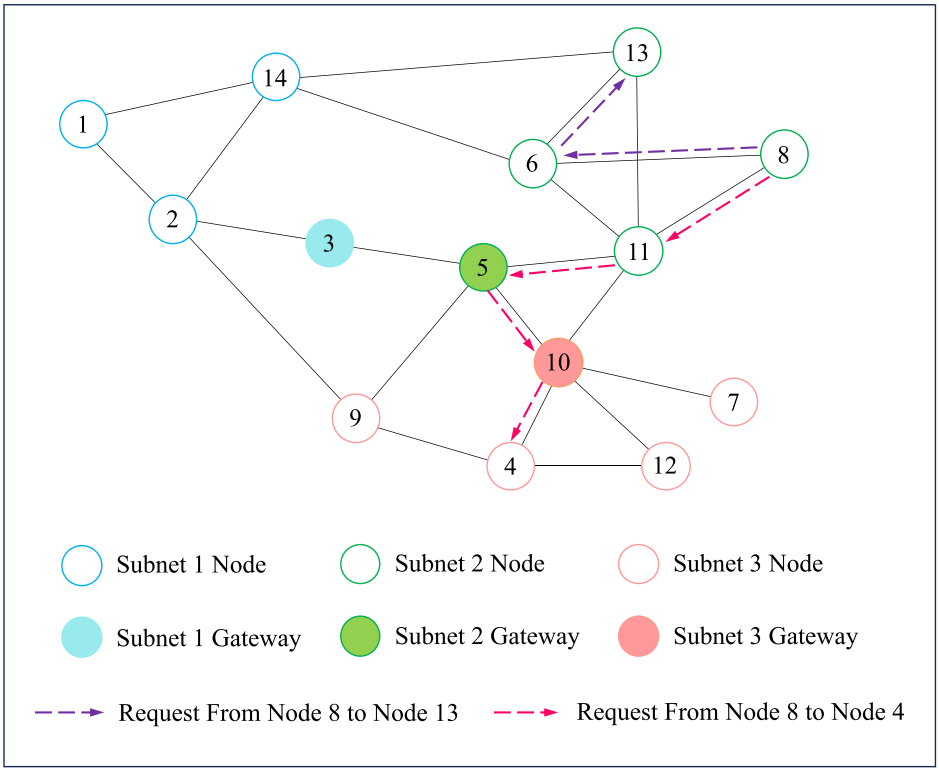

Fig. 4. Examples of latency-aware hierarchical routing algorithms.

to-end route from the source node to the destination node, which serves as the final cross-subnet routing solution (as shown in lines 6-9).

Algorithm 4: Latency-aware hierarchical routing algorithm.

Input: Network graph $G(V, E)$, gateway deployment scheme G_b, request flow f_k, latency weight ω_{latency}, SubnetInfo
Output: Path

```
1  s ← s_k;
2  d ← d_k;
3  if SubnetInfo[s] = SubnetInfo[d] then
4  |   Path ← LatencyOptimalPath(G, s, d, ω_latency);
5  else
6  |   path1 ← LatencyOptimalPath(G, s, s_gw, ω_latency);
7  |   path2 ← LatencyOptimalPath(G, s_gw, d_gw, ω_latency);
8  |   path3 ← LatencyOptimalPath(G, d_gw, d, ω_latency);
9  |   Path ← RemoveDuplicates(path1 + path2 + path3);
10 end
11 return Path;
```

In Algorithm 4, the overall time complexity is primarily determined by the computation of the latency-weighted optimal paths. Although the form of the computation is consistent with that of the classical shortest-path algorithm, the routing metric used here is the dynamically updated link latency, and the algorithm therefore computes the minimum-latency path rather than the traditional hop-based or statically weighted shortest path.

When the source and destination nodes are within the same subnet, the minimum-latency path is computed only inside that subnet, with a time complexity of $O(m_i + n_i \log n_i)$, where m_i and n_i denote the number of edges and nodes in the subnet, respectively. When the source and destination nodes belong to different subnets, three segments of latency-weighted optimal paths must be computed. In this case, the overall time complexity of the path computation becomes $O(m_i + m_g + m_j + n_i \log n_i + n_g \log n_g + n_j \log n_j)$, where m_i, n_i and m_j, n_j represent the number of edges and nodes in the source and destination subnets, respectively, and m_g, n_g represent the number of edges and nodes between the subnets.

In addition, the deduplication operation in the algorithm has a time complexity of $O(l)$, which is typically much smaller than that of computing the minimum-latency paths. Therefore, the overall time complexity of Algorithm 4 remains $O(m_i + m_g + m_j + n_i \log n_i + n_g \log n_g + n_j \log n_j)$. Considering that in practical scenarios, each subnet is significantly smaller than the entire network (i.e. $n_i, n_j \ll n$, $n_g \ll n$), and inter-subnet connections are relatively sparse, the overall computational complexity of Algorithm 4 is substantially lower than that of traditional centralized global computation.

5. Experimental setup and evaluation

In this section, the experimental environment setup of ROCDSG and the evaluation of its results will be described in detail.

5.1. Experimental environment setup

The experiments are conducted on the Ubuntu 18.04 operating system on a virtual machine to validate the performance of the proposed ROCDSG based on the Ryu controller, Mininet platform, and Python, where the version of the Ryu controller is 4.34, the version of the Mininet is 2.3.1b4, and the version of the Python is 3.11.7. The experiments are conducted using a network topology from the Internet Topology Zoo [21], and the following three typical topologies are selected for testing: (1) the BICS topology, which contains 33 nodes and 48 edges; (2) the DFN topology, which contains 58 nodes and 87 edges; and (3) the Colt Telecom topology, from which we select 100 nodes and 122 edges. We assume that the energy required to activate the switch is 118 units, the energy required to activate the link is 48 units, and the energy consumed for each byte transmission is $4e^{-10}$ units. The energy parameters originate from different sources: the switch and link activation

energies follow the energy consumption model presented in reference [9], whereas the per-byte transmission energy is based on commonly used empirical values in mainstream data center networking studies to approximate the transmission overhead of typical high-performance switching devices. The switch used in the experiment is a Gigabit switch, and the bandwidth capacity of each link is set to 500 Mbps.

In the experiments, this paper models the arrival of data flows using a Poisson distribution [31] to generate dynamic traffic inputs and evaluate the performance of the proposed framework. Let the random variable $X(t)$ denote the number of flows arriving in time step t, then its probability mass function is given by:

$$P[X(t) = k] = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k = 0, 1, 2, \ldots \tag{49}$$

where λ is the average flow arrival rate per unit time, k represents the actual number of flows arriving in that time step, and e is the base of the natural logarithm. To ensure consistency and reproducibility of the experimental settings, this paper generates input flows according to Eq. 49. Specifically, the flow arrival process follows a Poisson distribution with rate $\lambda = 5$ flows/s, and the packet size of each flow follows a normal distribution with mean 500 KB and standard deviation 50 KB.

The parameter settings for the Firefly Algorithm are as follows: The population size is set to 100, the attractiveness β takes the value of 1.0, the decay factor γ is set to 1.0, the random perturbation factor ψ is set to 0.3 (the decay rate of the perturbation factor is 0.97), the strength of the variance η is set to 0.2, and the magnitude of the variance ξ is set to 0.05 [32]. The specific parameter settings for the experiments are described in detail in the relevant instructions in Table 3. To provide a fair and representative evaluation, we compare the proposed scheme with three categories of existing routing algorithms:

1. BCR [3]: a box-covering-based routing algorithm for large-scale SDNs that partitions the network into logical subnets via renormalization and then applies Dijkstra between and within subnets to obtain end-to-end paths.
2. HGWO-Firefly [12]:a swarm-intelligence routing scheme for cluster-based heterogeneous sensor networks, where a hybrid K-means and Grey Wolf Optimization strategy selects cluster heads and the Firefly algorithm searches energy-efficient multi-hop routes between sensor nodes and the base station.
3. QoSPR [21]: a QoS- and privacy-aware routing protocol for 5G-enabled IIoT that improves Infomap for dynamic subdomain division and uses deep and federated reinforcement learning to learn gateway deployment and routing under latency, load-balancing, and privacy constraints.

These three schemes represent three design paradigms: topology-aware structural routing (BCR), meta-heuristic routing with clustering (HGWO-Firefly), and ML-based gateway deployment and routing (QoSPR). Their construction ideas are highly relevant to our problem of coordinated dynamic subnet formation and gateway deployment.

5.2. Evaluation of experimental results

5.2.1. Subnet partitioning differences

Differences in the division of subnets in terms of load, traffic, and energy consumption are key metrics for assessing the rationality and effectiveness of subnet partitioning. In this paper, the difference in the division of each metric is defined as the gap between the subnet with the maximum value and the subnet with the minimum value. For example, the difference in the division of load is defined as the difference between the subnet with the maximum amount of load and the subnet with the minimum amount of load.

Fig. 5 shows the comparison results of different algorithms in terms of subnet load, traffic, and energy consumption distribution differences. As can be seen from Fig. 5(a), ROCDSG demonstrates superior load-balancing capability across different network topologies. As shown in the results of the three topologies, the load disparity achieved by ROCDSG generally falls within the range of 9.5%-14%, whereas the second-best algorithms typically exhibit disparities in the 12%-20% range. This corresponds to relative reductions of approximately 20%, 29%, and 39% in the three topologies, with an average improvement of about 29%. These results indicate that the subnet partitioning module provides stable and significant benefits in achieving balanced load distribution.

Table 3
Experiment-specific parameter settings.

Sr. No	Parameters	Value
1	BICS topology	33 nodes, 48 edges
2	DFN topology	58 nodes, 87 edges
3	Colt Telecom partial topology	100 nodes, 122 edges
4	Switch activation energy	118 units
5	Link activation energy	48 units
6	Single byte transmission energy	$4e^{-10}$ units
7	Switch bandwidth capacity	1000Mbps
8	Link bandwidth capacity	500Mbps
9	Average arrival rate	5 flow/s
10	Packet size mean	500KB
11	Packet size standard deviation	50KB
12	Population size	100
13	β	1.0
14	γ	1.0
15	ψ	0.3
16	η	0.2
17	ξ	0.05

Fig. 5(b) and (c) further validate the advantages of ROCDSG in traffic and energy consumption balancing. Experimental results show that the traffic disparity achieved by ROCDSG is approximately 9%, 13.9%, and 12.3% in the three topologies, while the corresponding disparities of the second-best algorithms are about 12.8%, 18%, and 16.6%. In terms of energy disparity, ROCDSG reaches about 9%, 8.6%, and 9%, whereas the second-best algorithms exhibit values of 9.6%, 20.7%, and 19%. These results indicate that ROCDSG reduces traffic disparity by an average of about 26% and energy disparity by about 39% compared with the second-best algorithms. Overall, by incorporating multi-dimensional constraints on traffic, load, and energy consumption during subnet partitioning, ROCDSG achieves an aggregated improvement of roughly 31% in resource balance across the three network topologies, providing a solid foundation for subsequent gateway deployment and routing optimization.

5.2.2. Average and maximum latency

The average and maximum latency of nodes are calculated by Eqs. (6) and (7), the average and maximum latency of nodes to gateways are calculated by Eqs. (8) and (9), and the average and maximum latency between gateways are calculated by Eqs. (10) and (11), respectively.

Fig. 6 reveals the differential performance of different algorithms after gateway deployment through multi-dimensional latency evaluation: as shown in Fig. 6(a), the average and maximum node latency of ROCDSG in Colt Telecom topology are stable at around 0.38 ms and 1.35 ms, respectively, which are lower than those of the BCR, HGWO-Firefly, and QoSPR algorithms, indicating that ROCDSG is able to effectively control the latency stacking on nodes in complex topologies. Further analyzing the comparison results in Fig. 6(b), it is found that the average and maximum node-to-gateway latency of ROCDSG in Colt Telecom topology are slightly higher than those of the QoSPR algorithm. This may be due to the fact that, during the gateway deployment process, due to the consideration of multiple constraints, the number of subnets increases as the network size grows, and in order to achieve the optimization objective, the gateway deployment locations of some subnets vary widely, resulting in an increase in the latency to

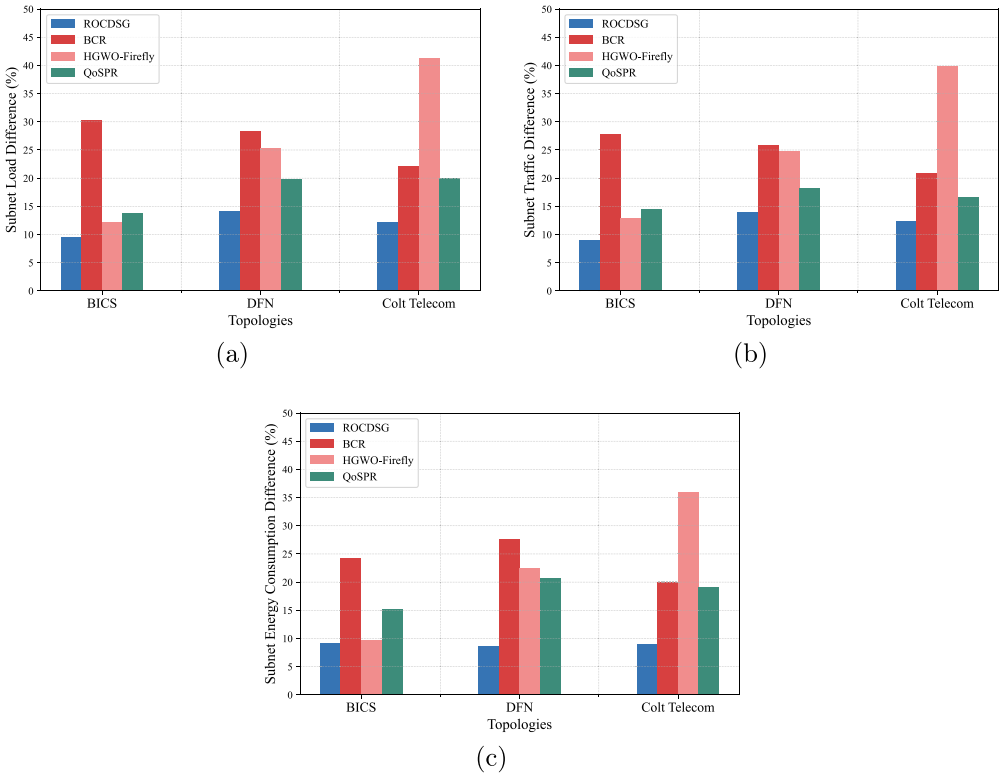

Fig. 5. Comparison of differences in subnet load, traffic, and energy consumption distribution: (a) Differences in load distribution; (b) Differences in traffic distribution; (c) Differences in energy consumption distribution.

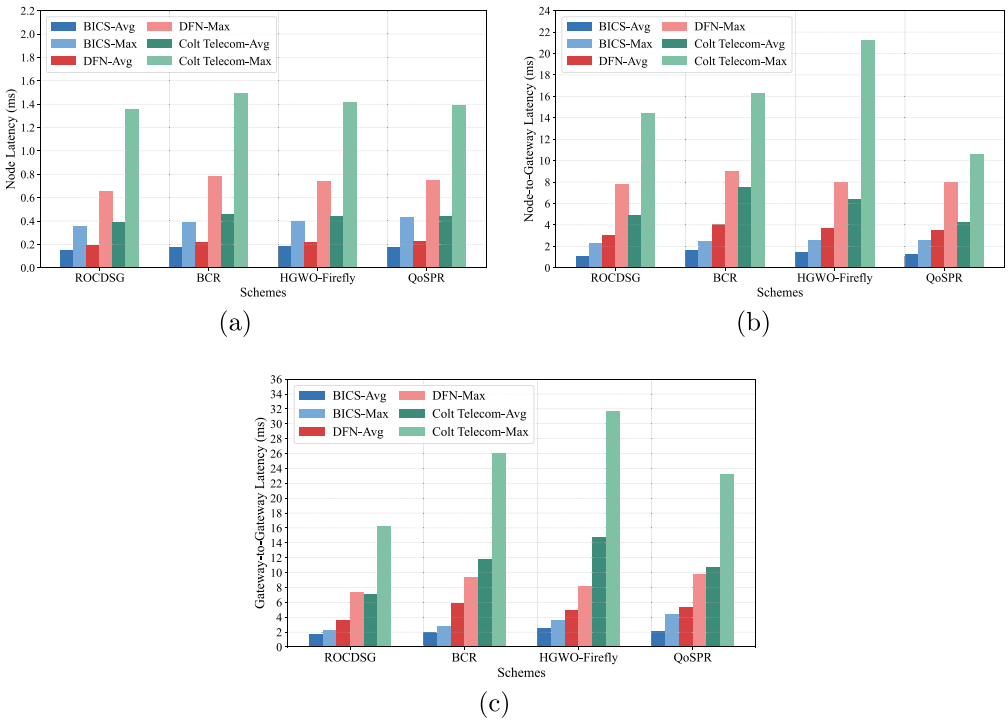

Fig. 6. Comparison of multidimensional average and maximum latency in different topologies: (a) Average and maximum latency of nodes; (b) Average and maximum latency from node to gateway; (c) Average and maximum latency between gateways.

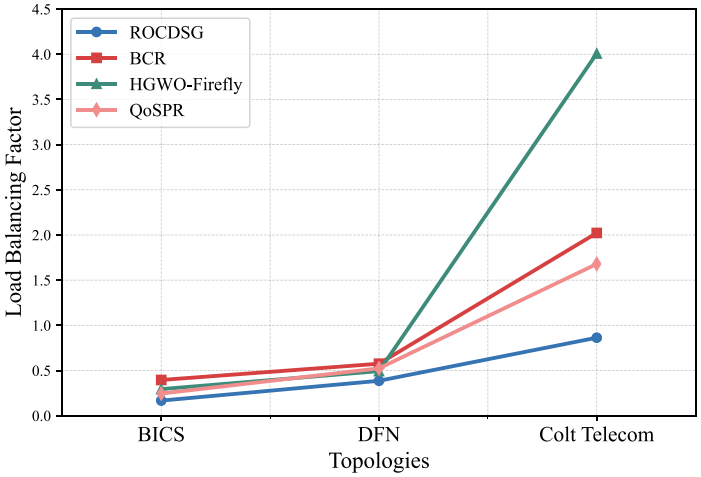

Fig. 7. Evaluation of network load balancing factor for different topologies.

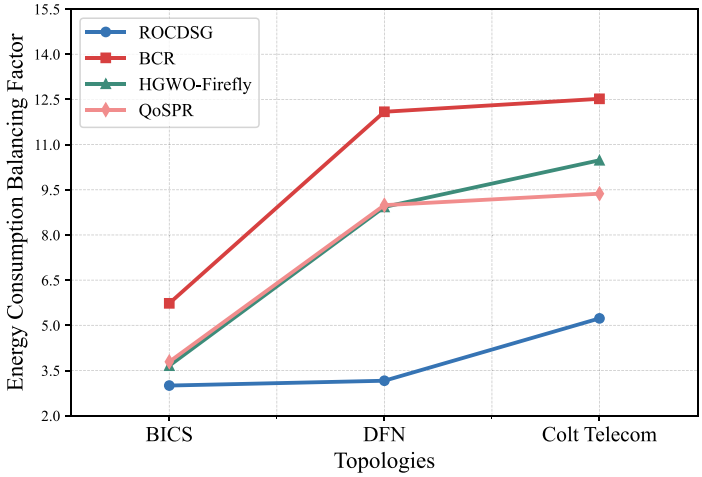

Fig. 8. Evaluation of network energy consumption balancing factor for different topologies.

reach the gateway. It is worth noting that the average and maximum node-to-gateway latency of ROCDSG outperforms the benchmark algorithm on both BICS and DFN topologies. Fig. 6(c) shows the evaluation results of inter-gateway latency, and it can be observed that ROCDSG performs well in cross-network link latency control, and the average and maximum inter-gateway latency of ROCDSG are substantially lower than those of the other algorithms in three different topologies. In summary, ROCDSG can effectively guarantee the QoS requirements in the data center network by comprehensively considering the minimization of node latency, node-to-gateway latency, and intergateway latency during gateway deployment.

5.2.3. *Load balancing and energy consumption balancing*

Since the ROCDSG proposed in this paper has used load constraints and energy consumption constraints to realize the subnet partitioning, for the global load and energy consumption performance evaluation of the network, this paper uses additional metrics, which are defined in the following formulas:

$$\lambda_L = \sqrt{\frac{1}{M}\sum_{i=1}^{|M|}\left(L(M_i) - \frac{L_T}{|M|}\right)^2} \tag{50}$$

$$\lambda_E = \sqrt{\frac{1}{M}\sum_{i=1}^{|M|}\left(E_i^T - \frac{E_T}{|M|}\right)^2} \tag{51}$$

λ_L and λ_E denote the load balancing factor and energy consumption balancing factor of the network, respectively; $L(M_i)$ denotes the total load of the i-th subnet; L_T denotes the total load of the whole network; E_i^T denotes the total energy consumption of the i-th subnet; and E_T denotes the total energy consumption of the whole network. This indicates that the smaller λ_L and λ_E are, the more balanced the load and energy consumption of the network are.

Figs. 7 and 8 show the overall load balancing and energy consumption balancing performance of different algorithms for each topology, respectively. Compared to the benchmark algorithms, ROCDSG shows better equalization capability in all topologies. Fig. 7 shows that in terms of load balancing, the performance of ROCDSG is relatively close to that of the benchmark algorithm in the BICS and DFN topologies, but it has a significant advantage in the Colt Telecom topology: the load balancing factor of ROCDSG is always maintained in the stable range below 1.0, whereas that of the benchmark algorithms is generally more than 1.6, which indicates that ROCDSG is more adaptable in larger-scale networks. Fig. 8 further shows that in terms of energy consumption balancing, the energy consumption balancing factor of ROCDSG only fluctuates to a limited extent as the topology complexity increases, while the benchmark algorithms show a clear performance degradation trend.

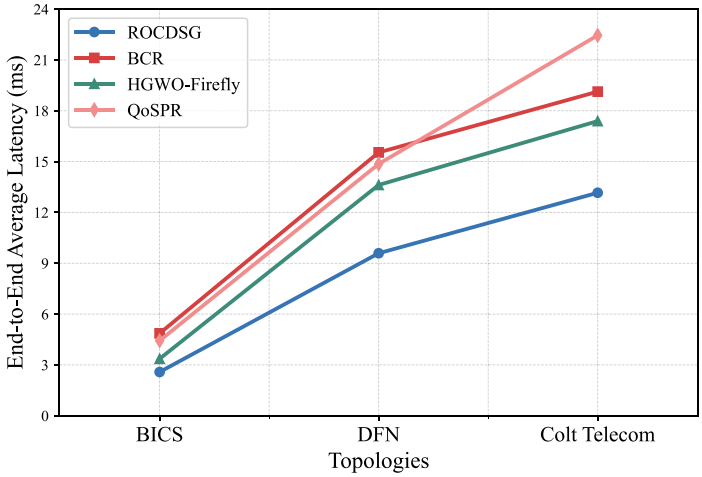

Fig. 9. Comparison of end-to-end average latency of request flows.

From the overall trends across the three topologies, as the network scale increases from BICS to DFN and Colt Telecom, the load and energy balancing coefficients of the baseline algorithms generally exhibit a clear tendency to increase or fluctuate more strongly, whereas the curves of ROCDSG remain much smoother with only limited growth in magnitude, indicating that ROCDSG is able to maintain stable balancing performance even in larger and more complex topologies.

5.2.4. *End-to-end latency*

The end-to-end average latency is obtained by the ratio of the sum of end-to-end latency of all request streams to the number of request streams. In this context, end-to-end latency refers to the sum of node latency and link latency experienced by the data during transmission. Node latency consists mainly of queuing latency and processing latency, while link latency consists mainly of transmission latency. This metric is used to measure the overall timeliness of data transmission from the source node to the destination node in the network.

Fig. 9 compares the end-to-end average latency performance of the algorithms in different topologies. It can be clearly seen that ROCDSG significantly outperforms the benchmark algorithm in all topologies. Specifically, the end-to-end average latency of ROCDSG in the three topologies is stable at around 2.6 ms, 9.5 ms, and 13.0 ms, respectively, while those of the suboptimal algorithm are about 3.3 ms, 13.5 ms, and 17.3 ms, respectively. Quantitative analysis shows that compared to the suboptimal algorithm, the end-to-end average latency of ROCDSG in

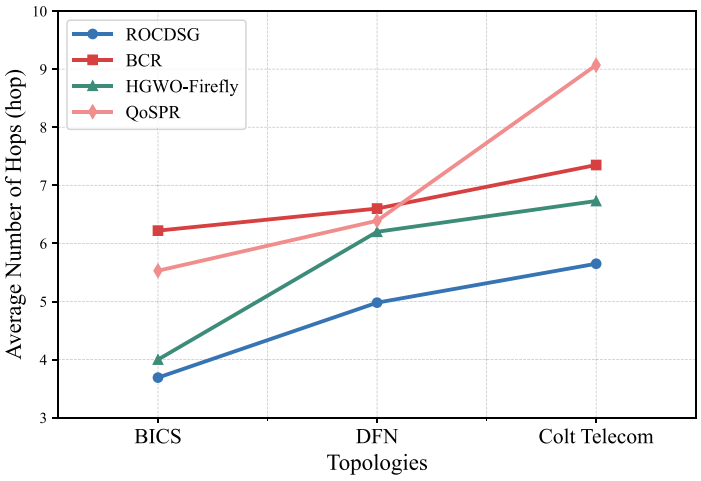

Fig. 10. Comparison of average hop counts of request flows in different topologies.

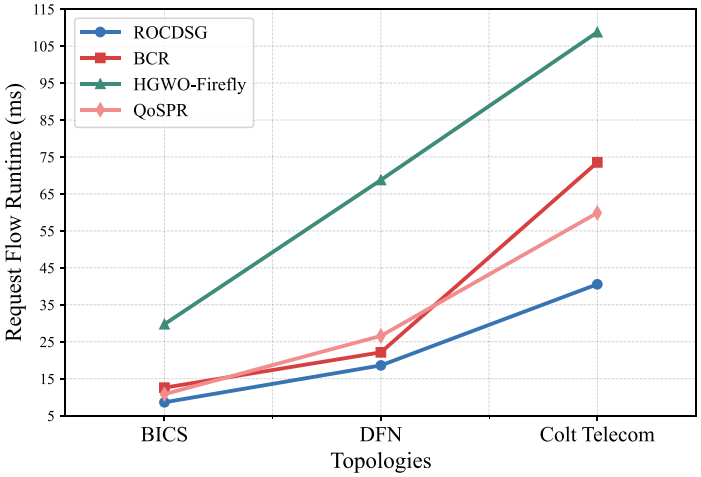

Fig. 11. Comparison of request flow runtime in different topologies.

the three topologies is reduced by 21.2%, 29.6%, and 24.9%, with an overall reduction of 25.2%. Meanwhile, it can be observed that as the topology scale increases, the overall level of end-to-end latency rises for all algorithms; however, the growth of ROCDSG is significantly slower than that of the others, indicating that it is more effective at restraining the increase in latency overhead as the network scales and thus exhibits better scalability.

5.2.5. Routing efficiency

In terms of routing efficiency, the average hop count and runtime are selected as metrics in this paper. The average hop count can be defined as the average value of the number of intermediate nodes that the data flow passes through from the source node to the destination node, reflecting the length of the routing path. Running time refers to the time required for the algorithm to complete one global routing decision or path computation, which measures the computational complexity and actual execution efficiency of the algorithm.

From Fig. 10, it can be observed that in terms of path hop count optimization, the average hop counts of ROCDSG in the three topologies are about 3.7, 5.0, and 5.6, respectively, while those of the suboptimal algorithms are about 4, 6.2, and 6.7, respectively. This suggests that the average hop counts of ROCDSG compared to the suboptimal algorithms have been reduced by 7.5%, 19.4%, and 16.4%, respectively, for the three topologies, with an overall reduction of 14.4%. In terms of runtime, Fig. 11 shows that the runtime of ROCDSG in BICS and DFN topologies is about 8.7 ms and 18.7 ms, respectively, which is 30.4% and 15% lower than that of the suboptimal algorithm of 12.5 ms and 22 ms, respectively. Notably, in the more complex Colt Telecom topology, the runtime of ROCDSG is about 40.5 ms, which significantly outperforms the 60 ms of the suboptimal algorithm QoSPR by 32.5%. Overall, ROCDSG shows a 25.6% reduction in runtime over the suboptimal algorithm, demonstrating better scalability and computational efficiency. In contrast, the HGWO-Firefly algorithm exhibits poor runtime performance in all topologies due to its reliance on heuristic algorithms, population initialization, and multiple rounds of iterative processes, further highlighting the advantages of ROCDSG.

A comparison of the trends across the three topologies further shows that, although the average hop count and running time of all algorithms increase with network size, the growth slope of ROCDSG is noticeably smaller, suggesting that in larger-scale networks it can maintain comparatively better scalability in both path quality and computational cost.

6. Conclusion

This paper addresses the multidimensional conflicts among QoS requirements, load balancing, energy consumption control, and routing complexity in data center networks. We propose ROCDSG, a framework that combines Louvain-based dynamic subnet partitioning, optimized gateway deployment, and latency-aware hierarchical routing. By jointly considering latency, load, and energy factors, ROCDSG efficiently adapts to complex data center environments. Experimental results show that ROCDSG significantly improves QoS performance and routing efficiency while achieving balanced load and energy distribution, demonstrating its effectiveness for large-scale data center applications.

Although the proposed framework leverages the controller's global awareness to drive the collaborative optimization of subnet partitioning, gateway deployment, and routing, achieving a balance between efficiency and flexibility, the overall solution still relies primarily on heuristic local updates, leaving room for further improvement toward global optimality. In future work, we will explore the collaborative integration of local and global optimization and plan to incorporate factors such as loss and link reliability into explicit modeling and optimization to enhance the robustness and adaptability of the framework.

CRediT authorship contribution statement

Qingjie Lin: Writing – original draft, Visualization, Validation, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Shuwu Chen:** Writing – review & editing, Software, Funding acquisition; **Haihui Xie:** Writing – review & editing, Software; **Tarik Taleb:** Writing – review & editing; **Yu Luo:** Supervision, Resources, Data curation, Conceptualization; **Zhaogang Shu:** Writing – review & editing, Supervision, Software, Funding acquisition.

Data availability

The authors do not have permission to share data.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Zhaogang Shu reports financial support was provided by Fujian Provincial Department of Industry and Information Technology. Zhaogang Shu reports financial support was provided by Fujian Provincial Department of Science and Technology. Zhaogang Shu reports financial support was provided by Fujian Provincial Department of Science and Technology. Zhaogang Shu reports financial support was provided by Network Communication Company. If there are other authors, they declare that they have no known competing financial interests or personal

relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the Project of Fu-Xia-Quan Collaborative Innovation Platform in Fujian Province, China (no. 2025E3012), the Project of Industry-University-Institute Cooperation in Colleges and Universities in Fujian Province, China (no. 2024H6007), the Project of Industry-University-Institute Joint Innovation in Colleges and Universities in Fujian Province, China (No. 2024H6030), the Industry-Research Project from Network Communication Company, China (no. KH240131A). It was also supported in part by the Federal Ministry of Research, Technology, and Space (BMFTR), Germany, through the Project 6GEM + under Grant 16KIS2411.

References

[1] T. Chen, X. Gao, G. Chen, The features, hardware, and architectures of data center networks: a survey, J. Parall. Distrib. Comput. 96 (2016) 45–74.
[2] H. Yang, W. Bai, A. Yu, J. Zhang, Performance evaluation of software-defined clustered-optical access networking for ubiquitous data center optical interconnection, Photon. Netw. Commun. 34 (2017) 1–12.
[3] L. Zhang, Q. Deng, Y. Su, Y. Hu, A box-covering-based routing algorithm for large-scale SDNs, IEEE Access 5 (2017) 4048–4056.
[4] S. Zhang, X. Xue, F. Yan, B. Pan, X. Guo, K. Mekonnen, E. Tangdiongga, N. Calabretta, Feasibility study of optical wireless technology in data center network, IEEE Photon. Technol. Lett. 33 (15) (2021) 773–776.
[5] Y. Zhao, X. Wang, F. Li, Q. He, X. Liu, Power optimization for low transmission delay in software defined data center networks, IEEE Trans. Mob. Comput. 24 (7) (2025).
[6] O. Hohlfeld, J. Kempf, M. Reisslein, S. Schmid, N. Shah, Guest editorial scalability issues and solutions for software defined networks, IEEE J. Sel. Areas Commun. 36 (12) (2018) 2595–2602.
[7] Y. Wang, X. Wang, H. Li, Y. Dong, Q. Liu, X. Shi, A multi-service differentiation traffic management strategy in SDN cloud data center, Comput. Netw. 171 (2020) 107143.
[8] Z.-W. Wei, B.-H. Wang, X.-T. Wu, Y. He, H. Liao, M.-Y. Zhou, Sampling-based box-covering algorithm for renormalization of networks, Chaos: Interdiscipl. J. Nonlinear Sci. 29 (6) (2019).
[9] M.N. Pathan, M. Muntaha, S. Sharmin, S. Saha, M.A. Uddin, F.N. Nur, S. Aryal, Priority based energy and load aware routing algorithms for SDN enabled data center network, Comput. Netw. 240 (2024) 110166.
[10] F. Naeem, M. Tariq, H.V. Poor, SDN-enabled energy-efficient routing optimization framework for industrial Internet of Things, IEEE Trans. Ind. Inf. 17 (8) (2020) 5660–5667.
[11] N.A. El-Hefnawy, O.A. Raouf, H. Askr, Dynamic routing optimization algorithm for software defined networking, Comput. Mater. Continua 70 (1) (2022) 69–89.
[12] J. Dev, J. Mishra, Energy efficient routing in cluster based heterogeneous wireless sensor network using hybrid GWO and firefly algorithm, Wirel. Person. Commun. 137 (2) (2024) 997–1028.
[13] Y. He, Z. Lu, J. Lei, S. Deng, X. Gao, Joint optimization of energy saving and load balancing for data center networks based on software defined networks, Concurr. Comput.: Pract. Exper. 33 (9) (2021) e6134.
[14] W. Jiawei, Q. Xiuquan, J. Chen, PDMR: priority-based dynamic multi-path routing algorithm for a software defined network, IET Commun. 13 (2) (2019) 179–185.
[15] Z. Lu, J. Lei, Y. He, Z. Li, S. Deng, X. Gao, Energy optimization for software-defined data center networks based on flow allocation strategies, Electron. (Basel) 8 (9) (2019) 1014.
[16] L.P.A. Sanchez, Y. Shen, M. Guo, DQS: A QoS-driven routing optimization approach in SDN using deep reinforcement learning, J. Parall. Distrib. Comput. 188 (2024) 104851.
[17] Y. Wang, Y. Li, T. Wang, G. Liu, Towards an energy-efficient data center network based on deep reinforcement learning, Comput. Netw. 210 (2022) 108939.
[18] G. Kim, Y. Kim, H. Lim, Deep reinforcement learning-based routing on software-defined networks, IEEE Access 10 (2022) 18121–18133.
[19] J. Chen, X. Huang, Y. Wang, H. Zhang, C. Liao, X. Xie, X. Li, W. Xiao, ASTPPO: A proximal policy optimization algorithm based on the attention mechanism and spatio–temporal correlation for routing optimization in software-defined networking, Peer-to-Peer Network. Applic. 16 (5) (2023) 2039–2057.
[20] N.K. Tiwari, A. Bajpai, S. Maurya, D. Chaurasiya, Efficient forwarding rule management in software defined network via subnet-Based pattern matching, SN Comput. Sci. 5 (6) (2024) 796.
[21] X. Wang, J. Hu, H. Lin, S. Garg, G. Kaddoum, M.J. Piran, M.S. Hossain, QoS and privacy-aware routing for 5G-enabled industrial Internet of Things: a federated reinforcement learning approach, IEEE Trans. Ind. Inf. 18 (6) (2021) 4189–4197.
[22] N. Torkzaban, A. Gholami, J.S. Baras, C. Papagianni, Joint satellite gateway placement and routing for integrated satellite-terrestrial networks (2020) 1–6.
[23] N. Matni, J. Moraes, H. Oliveira, D. Rosário, E. Cerqueira, LoRaWAN gateway placement model for dynamic Internet of Things scenarios, Sensors 20 (15) (2020) 4336.
[24] Y. Cao, H. Guo, J. Liu, N. Kato, Optimal satellite gateway placement in space-ground integrated networks, IEEE Netw. 32 (5) (2018) 32–37.
[25] Y. Zhang, Y. Tian, W. Wang, P. Cong, C. Chen, D. Li, K. Xu, Federated routing scheme for large-scale cross domain network (2020) 1358–1359.
[26] A.K. Bhowmick, K. Meneni, M. Danisch, J.-L. Guillaume, B. Mitra, Louvainne: hierarchical Louvain method for high quality and scalable network embedding (2020) 43–51.
[27] S. Hirahara, NP-hardness of learning programs and partial MCSP (2022) 968–979.
[28] Y. Wang, R. Kang, L. Guo, C. Zhang, J. Deng, P. Liu, M. Wen, T. Le, Deadline-constrained multi-commodity flow routing and scheduling optimization with consideration of edge lengths and capacities, Comput. Ind. Eng. 192 (2024) 110193.
[29] W. Rafique, A.S. Hafid, S. Cherkaoui, Softcaching: a framework for caching node selection and routing in software-defined information centric internet of things, Comput. Netw. 235 (2023) 109966.
[30] L. Hua, L. Tao, W. Xing, L.S. Bo, Correlated SVD and its application in bearing fault diagnosis, IEEE Trans. Neural Netw. Learn. Syst. (2021).
[31] J.E. Lozano-Rizk, J.I. Nieto-Hipolito, R. Rivera-Rodriguez, M.A. Cosio-Leon, M. Vazquez-Briseño, J.C. Chimal-Eguia, QOSCOMM: A data flow allocation strategy among sdn-based data centers for iot big data analytics, Appl. Sci. 10 (21) (2020) 7586.
[32] J. Wu, Y.-G. Wang, K. Burrage, Y.-C. Tian, B. Lawson, Z. Ding, An improved firefly algorithm for global continuous optimization problems, Expert Syst. Appl. 149 (2020) 113340.