frontiers | Frontiers in Energy Research

TYPE Original Research
PUBLISHED 26 January 2024
DOI 10.3389/fenrg.2023.1342854

OPEN ACCESS

EDITED BY
Haitao Zhang,
Xi'an Jiaotong University, China

REVIEWED BY
Zhiqiang Zhang,
Hefei University of Technology, China
Jice Wang,
University of Chinese Academy of Sciences, China
Lin Ma,
Hong Kong Polytechnic University, Hong Kong SAR, China

*CORRESPONDENCE
Wei Cui,
abcuiglad@139.com

RECEIVED 22 November 2023
ACCEPTED 18 December 2023
PUBLISHED 26 January 2024

CITATION
Liao X, Bao B, Cui W and Liu D (2024), Load balancing and topology dynamic adjustment strategy for power information system network: a deep reinforcement learning-based approach.
Front. Energy Res. 11:1342854.
doi: 10.3389/fenrg.2023.1342854

Load balancing and topology dynamic adjustment strategy for power information system network: a deep reinforcement learning-based approach

Xiao Liao, Beifang Bao, Wei Cui* and Di Liu

State Grid Information and Telecommunication Group Co., LTD., Beijing, China

As power information systems play an increasingly critical role in modern society, higher requirements are placed on the performance and reliability of their network infrastructure. In order to cope with the growing data traffic and network attack threats in the power information system, we select the power information system data center network as the research object and design an overall system solution based on software defined network, including the application layer, control layer and infrastructure layer. A typical fat tree network topology is simulated and analyzed. We define the load balancing and network topology dynamic adjustment problem as a Markov decision process, and design a data flow path acquisition method based on breadth-first search to construct the action space of each host. Then, a deep reinforcement learning algorithm based on deep Q-network, priority experience replay and target network is introduced to provide solutions for optimizing the performance of power information systems and responding to network attacks. Simulation results show that the proposed method is better than the traditional equal-cost multi-path algorithm in terms of average bandwidth utilization, average jitter and average packet loss, and can reduce the probability of network nodes being attacked by more than 11%.

KEYWORDS

power information systems, load balancing, flood attack, Markov decision process, deep reinforcement learning

1 Introduction

The smart grid (Fanlin and Wei, 2020; Gunduz and Das, 2020; Tufail et al., 2021) is the core of power grid modernization. The burgeoning evolution of power systems necessitates advanced capabilities in data acquisition, transmission, and processing. This evolution fosters a mutually beneficial relationship between information systems and power systems, marking their integration as a key feature of smart grid development. The power information system (PIS) exemplifies this trend by enhancing the interconnectedness of various power system components. This integration facilitates a seamless operational harmony across the spectrum of power generation, transmission, distribution, and consumption.

Alongside these power system developments, the expansion of the Internet has also occurred, which has precipitated an exponential increase in network devices and, correspondingly, network traffic. Such growth imposes substantial demands on network

resource allocation and management. As a new type of network architecture (Hamdan et al., 2021), software defined network (SDN) has a more flexible, dynamic, and frequent form of network resource allocation compared to traditional networks. SDN, characterized by its decoupled control and forwarding functions, centralized management, programmability, and open interfaces, embodies a stratified design. This idea of layered decoupling divides the network into application, control and infrastructure layers. OpenFlow, a standardized communication protocol (Nisar et al., 2020; Wazirali et al., 2021), operates at the intersection of the control and forwarding layers, enabling their disentanglement. The demarcation of control and forwarding planes, alongside features like resource virtualization and programmability, yields multiple advantages. It allows network hardware to concentrate on forwarding efficiency, thus diminishing costs; enables network intelligence through programmable software; and empowers a centralized controller to tailor network configurations in real-time, thereby enhancing service adaptability.

Unbalanced distribution of network traffic not only leads to congestion on a certain link of the network, but also leads to the suboptimal use of available resources. In the context of smart grids, load balancing technology is pivotal, as it allocates the load across operational units, thereby leveraging finite resources to accomplish a broader array of tasks. Using load balancing technology can maximize the use of network resources and improve the performance of the network. The problem of load balancing in SDN is an important research direction for power information systems. Deep reinforcement learning (DRL), a technique that has recently gained traction in artificial intelligence, is increasingly being employed to resolve various challenges in PIS due to its exceptional learning and adaptive capabilities, making it a robust solution for refining SDN load balancing.

However, potential attack risks pose great challenges to the stable operation of power information systems (Dash et al., 2022). External attackers can launch attacks through terminal nodes in the network, such as common flooding attacks, which will have a negative impact on network performance. Current research on the resistance of power information systems to external attacks is lacking. The application of deep reinforcement learning algorithms, which devise optimal strategies through environmental interactions, is significant. Their inherent intelligence and adaptability are critical in identifying, mitigating, and safeguarding against malicious traffic intrusions, thereby bolstering network robustness.

In short, using DRL to solve SDN load balancing and defense attack problems is a very promising research direction. It can maximize the utilization of network resources and improve network performance, which has important practical significance for the development of power information systems and SDN networks. This paper proposes a load balancing and topology dynamic adjustment strategy based on deep reinforcement learning for power information systems. To the best of our knowledge, this is the first work that considers both load balancing and defense against external traffic attacks. The main contributions of this article are as follows:

(1) The problem of load balancing and network topology dynamic adjustment is modeled as a Markov decision process, and both traffic forwarding and defense against flooding attacks are paid attention to under the SDN framework.
(2) This paper proposes a load balancing and network topology dynamic adjustment method based on deep reinforcement learning, and introduces deep Q network, priority experience replay and target network to improve the performance and anti-attack capabilities of the power information system.
(3) Experimental simulations are conducted on the common fat tree topology of power information system data center networks to verify the effectiveness of the method proposed in this article.

The remainder of this article is organized as follows. Section 2 briefly reviews the research efforts related to load balancing and attack defense. In Section 3, we introduce the system architecture of our study. Section 4 presents a detailed description of our proposed method. In Section 5, we provide the simulation results. Finally, in Section 6, we conclude this article.

2 Related work

The escalation of power system capacities inevitably leads to an upsurge in data processing and transmission, thereby intensifying the informational network's transmission burden. To circumvent network link congestion and enhance network performance, the SDN load balancing algorithm has emerged as a pivotal research area within the SDN domain. Scholars globally have delved deeply into SDN load balancing research. The study in Priyadarsini et al. (2019) introduces a self-adaptive load balancing scheme that dynamically distributes load across multiple controllers, effectively managing high-load conditions while accounting for the proximity between switches and target controllers. In Jamali et al. (2019), Genetic Programming based Load Balancing (GPLB) is proposed to select the most efficient path by integrating real-time load data. The study by Chakravarthy and Amutha (2022) presents an innovative algorithm for load balancing that proactively computes the capacity of switches along a packet's routing path. Rupani et al. (2020) proposed a load balancing solution in SDN that utilizes a global network view to select the optimal data transmission path, significantly reducing latency through a neural network model. In Ejaz et al. (2019), explored traffic load balancing within SDN and NFV frameworks, achieving enhanced network performance by deploying a virtual SDN controller as a VNF that dynamically adds secondary controllers to distribute increased traffic loads. The study by Xue et al. (2019) introduces the Genetic-Ant Colony Optimization (G-ACO) scheme, which synergizes Genetic Algorithm (GA) for a rapid global search with ACO for efficient optimal solution finding, significantly enhancing pathfinding efficiency and reducing round-trip times and packet loss rates. Xu et al. (2019) demonstrated enhanced traffic management efficiency through dynamic switch-to-controller mapping in SDN. They introduced the 'BalCon' and 'BalConPlus' migration schemes that balance loads across controllers with minimal migration costs. In Fang et al. (2019), a reinforcement learning-based load balancing algorithm is put forward, applying neural learning for SDN routing and crafting a Q-learning based routing protocol. Nonetheless, such algorithms face challenges like high computational demands and limited scalability in extensive networks. How to design reliable

algorithms with good real-time performance and strong robustness is the focus of this article.

The stable operation of the power system can not be separated from the network security protection. Beyond firewall applications, real-time monitoring through intrusion detection is vital for timely anomaly detection and power system protection. Literature Haghnegahdar and Wang (2020) presents a novel intrusion detection model utilizing a whale optimization algorithm-enhanced artificial neural network to effectively classify various levels of cyber-attacks and incidents within power systems. Li et al. (2020) introduced 'DeepFed', a novel federated deep learning approach for detecting cyber threats in industrial CPSs, utilizing a unique combination of CNNs and GRUs within a privacy-preserving federated learning framework secured by Paillier cryptosystem protocols. A study in Choi et al. (2019) showcases a network intrusion detection system developed using an autoencoder, an unsupervised learning algorithm, boasting a 91.70% accuracy rate. Addressing DDoS attacks, Mendonça et al. (2021) proposes an IDS based on a Tree-CNN with a Soft-Root-Sign (SRS) activation function, enhancing model generalization and expediting training through batch normalization. Literature Pontes et al. (2021) presents the Energy-based Flow Classifier (EFC), an innovative anomaly-based classifier using inverse statistics for flow-based network intrusion detection. Khalid et al. (2019) proposed novel, resource-efficient algorithms that integrate distributed and intrusion detection systems to mitigate flood attacks. This paper innovatively introduces deep reinforcement learning methods to defend against external malicious traffic attacks from the perspective of dynamic adjustment of network edge topology of power information systems.

3 System architecture

In this section, we elaborate on the system architecture of our study. This section is divided into two subsections: the first subsection provides an overview of the overall system architecture, including the application layer, control layer, and infrastructure layer, while the second subsection delves into the details of the fat-tree topology commonly used in power information system data centers.

3.1 Overall system architecture

Power information system plays a vital role in the emerging landscape of smart grid, and its efficient operation relies heavily on a well-constructed system architecture. The core concept of SDN is to separate the control layer and data forwarding layer in the network. The logically centralized control layer uses communication interfaces to implement centralized control of network devices in the data forwarding layer. The application layer can flexibly control network devices in the data forwarding layer by writing software. It requires its own control network to achieve programmable control. Based on the current SDN research, this paper introduces deep reinforcement learning to achieve load balancing and resist attacks in PIS network.

The system architecture of the load balancing and dynamic adjustment strategy studied in this article is shown in Figure 1, which includes three main layers: application layer, control layer and infrastructure layer. These layers work synergistically to create an adaptive and efficient network infrastructure tailored for the unique challenges posed by the PIS data center environment.

Application Layer: The application layer resides at the top of the entire architecture, running trained deep reinforcement learning agents for real-time decision-making. The goal of this layer is to generate load balancing and topology adjustment strategies, ultimately deployed to the infrastructure layer to instruct the underlying terminal devices, ensuring efficient data forwarding and rapid responsiveness in the PIS. Each agent employs the DQN algorithm, using network terminal node information as state input, and leveraging deep Q-networks to select actions. During interaction with the environment, they learn strategies that maximize reward values.

Control Layer: The control layer serves as the central command center for the entire network. It is mainly composed of SDN controllers, bridging the application layer and the infrastructure layer. SDN controllers are responsible for managing network policies, traffic engineering, and routing. The control layer transfers network information collected from the infrastructure layer and terminal requests to the application layer via a northbound interface, and forwards load balancing and topology adjustment strategies received from the application layer to the infrastructure layer via a southbound interface.

Infrastructure Layer: The infrastructure layer resides at the bottom, responsible for data processing, forwarding, and state collection. The infrastructure layer comprises network devices such as switches, routers, etc., often referred to as the data plane. These underlying network devices lack control capabilities and possess only basic data processing functions, such as data forwarding and state collection based on flow tables issued by the controller. In the SDN environment, communication between switches and controllers is facilitated through the OpenFlow protocol, allowing the controller to instruct switches on where to forward data packets, processing packets based on the combination of packet content and switch configuration state.

3.2 Fat-tree topology in power information system data centers

In the power information system, data centers play a critical role. For many power distribution and power consumption end equipment, the switches within the power information system data centers need to process the traffic requests of each terminal equipment in real time and select appropriate communication paths for forwarding. The currently commonly used network architecture in data centers is the fat-tree topology, which is the backbone of network communication and data exchange. It is carefully designed to meet the requirements of high performance, fault tolerance, and scalability, and has become the cornerstone of modern data center architecture. The fat tree topology is an improvement over the traditional three-layer tree topology. Its essence is a three-layer cascaded multi-root tree topology with a switch as the core (switch-only). The entire topology can be described using a single parameter k (k represents the number of ports on a single switch). A classic

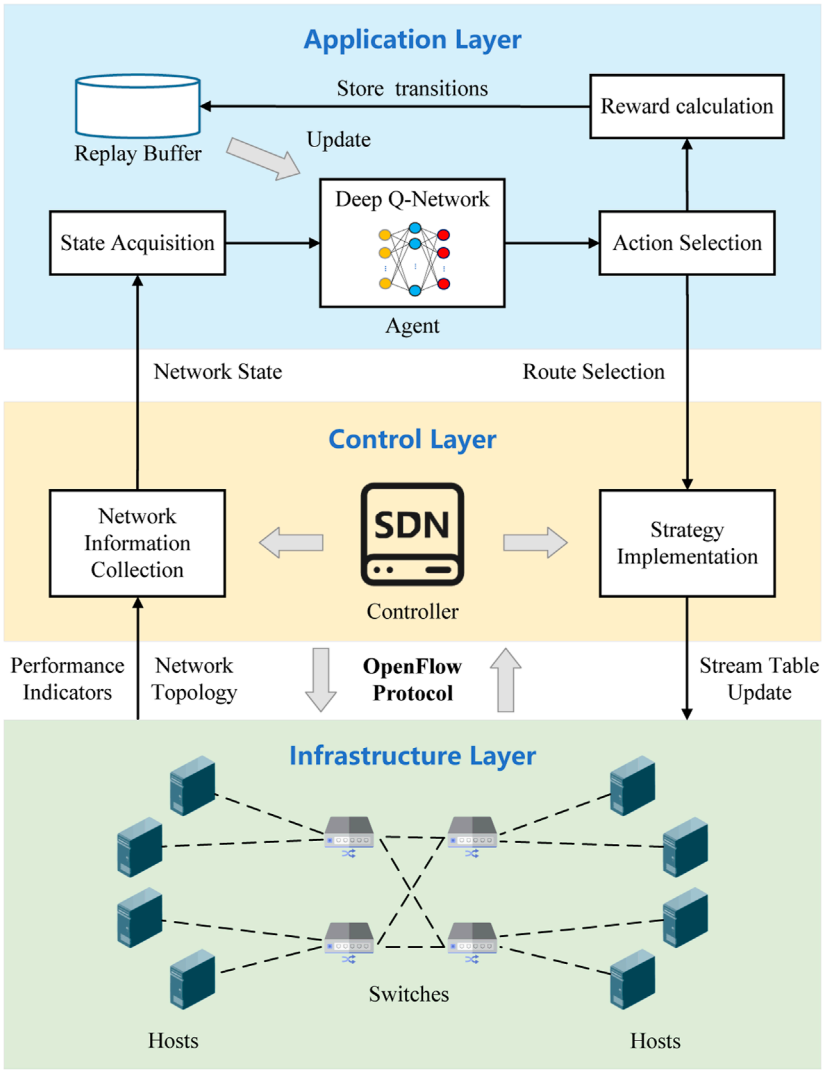

FIGURE 1
Overall system architecture.

quad-tree fat tree topology ($k = 4$) is shown in Figure 2. The three switch layers from top to bottom are the core layer, the aggregation layer and the edge layer.

Core Layer Switches: The core layer serves as the central hub of the fat-tree topology, ensuring high-speed and low-latency connections within the data center. Core layer switches aggregate data from different parts of the data center and are responsible for consolidating and forwarding network traffic. They play a crucial role in facilitating efficient flow of critical information, such as real-time grid monitoring data, between different segments of the data center.

Aggregation Layer Switches: Aggregation layer switches act as an intermediate layer, connecting core layer and edge layer switches. They are pivotal in facilitating communication between various edge switches and the core layer, enabling the efficient flow of data between different sections of the data center. Aggregation switches enhance network scalability and flexibility and are integral to load balancing and redundancy, ensuring optimal data transmission.

Edge layer switches: At the edge layer, network switches are directly connected to end-user devices, sensors, etc. in the power information system. Edge layer switches handle the initial data processing and routing, ensuring that data from different devices can be efficiently transferred to the aggregation layer. These switches play a vital role in managing diverse data sources within the data center, ensuring that incoming data is efficiently directed to its appropriate destination.

Hosts: Host devices include servers, workstations, sensors, and other end-user devices. They are directly connected to edge layer switches and serve as the terminal devices responsible for generating, processing, and consuming data within the power information system. Hosts constitute both the ultimate source and destination of data flows, making them an essential component of the power information system.

As shown in Figure 2, the fat-tree network topology systematically allocates edge layer switches and aggregation layer

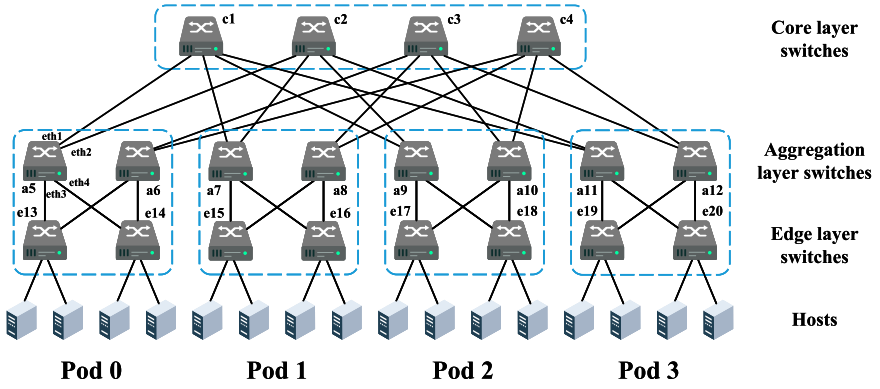

FIGURE 2
Quadruple fat tree topology.

switches to different arrays, referred to as "Pods." A k-fork fat-tree topology consists of k pods, with each pod capable of accommodating $(k/2)^2$ host devices. Each pod's aggregation layer and edge layer contain $k/2$ switches with k interfaces. Each edge layer switch is connected to $k/2$ hosts and $k/2$ aggregation layer switches, while each aggregation layer switch is connected to $k/2$ edge layer switches and $k/2$ core layer switches. Consequently, a k-fork fat-tree structured network can accommodate $k^3/4$ host devices.

Due to its excellent scalability, connectivity, and cost-effectiveness, the fat-tree topology finds widespread application in large-scale system-level network centers, providing high-throughput transmission services for data center networks. This paper focuses on load balancing and flood attack mitigation in the fat-tree topology of data center networks within the power information system, and experimental investigations are conducted using a 4-fork fat-tree topology on a network simulation platform.

4 Proposed method

In this section, we provide a detailed description of our proposed load balancing and dynamic network topology adjustment methodology. Initially, based on the characteristics of the system architecture, a Markov Decision Process is constructed. Subsequently, to determine the action space of each agent, we designed a data flow path acquisition method based on BFS (Breadth-First Search). Following that, we employ Deep Q-Network and Priority Experience Replay techniques to establish the load balancing and topology adjustment algorithm. Lastly, we elaborate on the training process and application details of the introduced algorithms.

4.1 MDP model

Existing SDN routing methods struggle with low efficiency and high computational complexity. Reinforcement learning, known for optimizing strategies, could solve these problems. In reinforcement learning, an agent interacts with an environment over discrete time steps. Reinforcement learning problems are commonly modeled as a Markov Decision Process (MDP), which typically comprises four main components: state, action, transition probability, and a reward function. The state provides a holistic description of the agent's current situation within the environment. An action refers to the decision made by the agent. Transition probability denotes the likelihood of the agent transitioning into a new state after taking a specific action in the current state. The reward function reflects the environment's feedback based on the outcome resulting from an action taken by the agent.

During the interaction with the environment, the expected cumulative reward obtained by the agent in state s_t after taking action a_t according to policy π can be represented by the state-action value function $Q^{\pi}(s_t, a_t)$:

$$Q^{\pi}(s_t, a_t) = E_{\pi}\left[\sum_{t=t_0}^{T} \gamma^{t-1} r_t | s_t, a_t\right] \tag{1}$$

where t_0 denotes the starting time, T represents the ending time, γ is the reward discount factor satisfying the range [0,1], and r_t is the immediate reward obtained at time t.

The primary objective of reinforcement learning is to adjust strategies based on feedback from interactions with the environment, aiming to derive an optimal policy that maps states to the best corresponding actions. This ensures that the value function $V(s)$ for each state s is maximized under this policy. The value function for state s can be represented as:

$$V^{\pi}(s) = E\left[\sum_{t=t_0}^{T} \gamma^{t-1} r_t | s_0 = s\right] \tag{2}$$

Traditional Q-Learning method utilizes tables to store Q-values for state-action pairs. However, such tabular methods are infeasible for problems with large scales, especially for continuous state and action spaces, due to the curse of dimensionality. Integrating with deep learning and using deep neural networks to approximate the Q-value function offers an effective solution to this challenge.

For the fat-tree network in the power information system studied in this paper, our goal is to formulate the optimal traffic

forwarding strategy based on real-time flow requests from each terminal host and to defend against potential flooding attacks. For regular flow requests, the optimal strategy trained using reinforcement learning is used to select forwarding paths. For abnormally high traffic requests (considered as external attacks), communication links between the edge-layer switch and the host should be promptly severed, adjusting the end network topology to prevent attack traffic from entering the power information system. We consider each terminal host as an agent and accordingly train a deep Q-network to guide the flow table policy. For each agent, we construct a Markov decision process, represented by the tuple $\{S, A, P, R\}$, with each element defined as follows:

(1) S represents the state space, with s_t denoting the environment state of the agent at time step t. In the system studied in this paper, considering that each host can access the traffic request information of all hosts. Their acquired state at each moment is identical. The environment state comprises two pieces of information: the target host of each source host and the traffic request. If the target host of the *ith* host is denoted as u_i and the traffic request as w_i, then the environment state at time t can be represented as $s_t = [u_1, \ldots, u_{16}, w_1, \ldots, w_{16}]$.

(2) A denotes the action space, with a_t representing the action taken by the agent at time step t. For each agent, its action space can be represented as $A = [A_0, A_1, \ldots, A_m]$, where A_0 indicates cutting off the communication link with the edge-layer switch (topology adjustment), and $A_1, \ldots, A_m$ represent all shortest paths for forwarding traffic to other hosts.

(3) P indicates the state transition probability. The probability of transitioning from state s_t to s_{t+1} can be represented as $p_t: s_t \times a_t \rightarrow s_{t+1}$.

(4) R signifies the reward function. Through the design of the reward function in this study, we aim to guide agents to achieve load balancing and defend against flooding attacks. Our designed reward function consists of four parts. During the training process, if a host launches a flooding attack, producing an exceptionally high bandwidth request at the current moment, we decide the value of the first part of the reward r_1, based on whether the agent severs the link to adjust the topology. If it successfully adjusts the topology to defend against the attack, then r_1 is 5; otherwise, it is −5. That is:

$$r_1 = \begin{cases} 5, if\ resist\ attacks\ successfully \\ -5,\ else \end{cases} \tag{3}$$

The value of r_1 is designed to incentivize the network's defensive mechanism against flooding attacks. In our model, when a host is under a flooding attack, exhibiting abnormally high bandwidth requests, the network needs to respond effectively. If the network successfully adjusts its topology to mitigate the attack, r_1 is set to a positive value (+5) to reward this effective response. Conversely, if the network fails to adjust and resist the attack, r_1 is set to a negative value (−5) to penalize this failure. This binary reward structure helps in reinforcing the desired behavior of the network in the face of potential threats.

To achieve a regular load balancing effect, we introduce the second part of the reward r_2. Let the designated traffic bandwidth of the host be denoted as $w_{applied}$, and the actual bandwidth of this data stream be w_{actual}. Based on whether the endpoint of the path selected by the agent is the target host, r_2 can be designed as follows:

$$r_2 = \begin{cases} 1 + w_{actual}/w_{applied},\ if\ the\ path\ ends\ at\ destination\ host \\ -3,\ else \end{cases} \tag{4}$$

The value of r_2 is aligned with the goal of achieving optimal load balancing. It is structured to reward actions that lead to efficient traffic distribution across the network. When the endpoint of the path selected by the agent matches the target host (indicating efficient routing), a positive reward (1) is given. This reward is proportionate to the ratio of actual bandwidth to the designated traffic bandwidth, encouraging not only accurate but also efficient bandwidth utilization. In cases where the selected path does not end at the destination host, a negative reward (−3) is assigned to discourage inefficient routing decisions.

Let the data transmission delay be d_j and the packet loss rate be l_p, then the reward for the agent during time interval t can be represented as:

$$r_t = r_1 + r_2 - \beta_1 d_j - \beta_2 l_p \tag{5}$$

where β_1, β_2 are the weight coefficients.

4.2 Data flow path acquisition method

Based on the MDP model established above, determining the action space for each host is the first issue to address. Specifically, the shortest transmission path from a host to all other hosts needs to be defined. Considering the characteristics of the Fat-Tree topology, this paper designs a data flow path acquisition method based on Breadth-First Search (BFS) to construct the action space for each host.

By analyzing the Fat-Tree topology structure, a notable feature emerges: once the data flow reaches the highest node of the transmission path, the downstream path becomes unique. Referring to the Fat-Tree topology in Figure 2, the shortest forwarding paths for different target hosts can be categorized into three types: (1) There is only one shortest path for traffic exchange between two terminal hosts on the same edge switch in the same pod, that is, through the edge switch. (2) For traffic interchange between terminal devices under different edge switches within the same pod, it is only necessary to identify the aggregation switch within that pod, resulting in two shortest paths. (3) For traffic interchange between terminal devices across different pods, identification is needed up to the core switch, leading to four shortest paths. Thus, in the quad-tree Fat-Tree topology shown in Figure 2, we identify 53 shortest paths to all hosts, making the action space dimension 54.

Breadth-First Search (BFS) is an algorithm used for traversing or searching tree structures. We employ BFS to find all potential shortest paths from a source host to other target hosts. When searching within a tree, BFS visits the current node first and then accesses all nodes adjacent to the current node. This approach ensures that during the search

```
Input: source host
Output: action space
Initializes a queue, a collection, and an empty dictionary.
The source host enqueues.
Set collection visited[source] = 1.
while queue is not empty do
    node = queue.dequeue().
    Mark the node as accessed, namely visited[node]=1.
    for neighbor=0 to len(MA[neighbor])-1 do
        if MA[node][neighbor]==1 and visited[i]==0 then
            Add the new path to the queue.
            if neighbor is not source host then
                Adds the path to the shortest path dictionary.
            end
        end
    end
end
```

Algorithm 1. The data flow path acquisition method based on BFS.

process, nodes closest to the source node are visited first. The rationality behind employing BFS in our context stems from its efficiency in identifying all possible shortest paths within a tree-like structure. Given the Fat-Tree topology's hierarchical and layered nature, BFS is particularly adept at systematically exploring this network and identifying optimal paths for data transmission.

The data flow path acquisition method based on BFS proposed in this paper is illustrated as Algorithm 1. Firstly, we define the adjacency matrix M_A for the Fat-Tree network topology. We initialize a queue where each element contains a current node and a path list from the source host to that node. A set is created to keep track of nodes that have already been visited. An empty dictionary is also established to store the shortest paths from target hosts to the source host. Next, the source host is selected as the starting point. In each iteration, a node is popped from the queue, marked as visited, and all its adjacent nodes are traversed. For each adjacent node, if it has not been visited and a connection exists, a new path is appended to the queue. If the adjacent node is not the source host, the found path is added to the shortest path dictionary. This process is repeated until the queue is empty.

We store all discovered paths to construct the action space from the source host to other target hosts. This action space will serve as the foundation for our deep reinforcement learning model, enabling it to efficiently select paths to achieve the objectives of load balancing and network topology adjustment.

4.3 Load balancing and topology adjustment strategies

In electric power information systems, due to the uncertainty of user traffic requests and the potential threat of flooding attacks, network congestion can easily arise, affecting user experience and even system stability. To this end, building upon the established MDP model, we have enhanced the classic DQN algorithm and proposed a strategy based on deep reinforcement learning for load balancing and dynamic network topology adjustment. This strategy dynamically adjusts the data flow forwarding path and network terminal topology in real-time, thereby achieving adaptive and attack-resistant network performance.

Deep neural networks possess powerful representational capabilities. We employ a deep neural network to approximate the Q-function, using the current state of the agent during a given time period as input, and the state-action value $Q(s,a) \approx Q(s,a,\theta)$ as the output, where θ represents the neural network parameters. In each decision-making step, the agent chooses an action based on the current network state, aiming to maximize the expected cumulative reward. By fitting the Q-function using a deep neural network, the agent can handle large-scale and continuous state spaces.

During the training process of the deep Q-network, each interaction with the environment results in an experience transition consisting of the state, action, reward, and next state. The classical DQN algorithm employs a replay buffer to store these transitions. During training, mini-batches of samples are randomly drawn from this buffer to learn, which breaks the temporal correlation between data and stabilizes the learning process. In this paper, we introduce an enhancement by adopting the prioritized experience replay technique, where experiences are drawn based on their importance rather than at random. When storing experience transitions, the temporal difference (TD) error is calculated concurrently. Each experience is assigned a priority based on the magnitude of the TD error, with experiences having larger errors receiving higher priorities. Consequently, when drawing from the replay buffer, experiences with higher priorities are more likely to be chosen. Since priority sampling introduces a bias, importance sampling weights are employed to correct this bias, ensuring that the learning process remains unbiased. Through the prioritized experience replay technique, those experiences that are "challenging" or "unexpected" can be reviewed and learned more frequently, thereby accelerating the learning process and potentially enhancing the network's convergence rate and overall performance.

In addition, we introduce a target network with the same structure as the deep Q-network to solve the correlation and stability problems. Initially, both the deep Q-network and the target network share the same parameters. Throughout the training process, every C steps, the parameters θ' of the target network are updated to θ. After sampling a minibatch of size B from the prioritized experience replay buffer, we can compute the estimated Q-value:

$$Q_{evel} = Q(S_t, A_t, \theta) \tag{6}$$

The target Q-value is:

$$Q_{tar} = r_t + \gamma^* Q\left(s_{t+1}, \arg_{a'} Q(s_{t+1}, a', \theta), \theta'\right) \tag{7}$$

The loss function is calculated based on the difference between the target Q-value and the estimated Q-value. Gradient descent is then applied to update the main network parameters θ. The loss function is defined as:

$$L(t) = \sum_{i=1}^{B} (Q_{tar} - Q_{evel})^2 \tag{8}$$

Through the aforementioned techniques, the agent can more effectively learn the mapping between network states and actions, thereby identifying the optimal strategies for load balancing and topology adjustments. With adequate

Initialize the parameters of deep Q-network θ and target network θ' randomly.
Initialize the prioritized experience replay buffer and the greedy coefficient.
for *episode = 1 to E* **do**
  for *t = 1, 2, , T* **do**
    Initialize the target host and traffic request.
    Receive the initial state s_t.
    Select an action according to the ε-greedy rule.
    The SDN controller sends instructions to update the flow table, then gets the reward r_t and the next state s_{t+1}.
    Calculate the priority and store the transitions $\{priority, (s_t, a_t, r_t, s_{t+1})\}$ in the prioritized experience replay buffer.
    Update the deep Q-network parameters by sampling transitions from the buffer.
    Update the greedy coefficient.
    Set $\theta' = \theta$ every C steps.
  end
end

Algorithm 2. Training Process of the Deep Q-Network.

training, the deep Q-network can provide real-time, dynamic, adaptive, and attack-resistant strategies for load balancing and network topology adjustments in the electric power information system network.

4.4 Training procedure and application

The pseudocode for the training process of the Deep Q-Network is presented in Algorithm 2, which primarily outlines the procedure to update neural network parameters using experience tuples acquired from interactions. We set the total number of training episodes to E. In each episode, interactions are carried out over T discrete time steps. Initially, the target hosts and bandwidth requests of each host are initialized, forming the current state s_t. Subsequently, each agent selects an action a_t from its action space based on the ε-greedy rule. The SDN controller issues commands to update the flow table based on this action, yielding the immediate reward r_t and the subsequent state s_{t+1}. After computing the priority, the transition $\{priority, (s_t, a_t, r_t, s_{t+1})\}$ is stored in the prioritized experience replay buffer. Then, a minibatch of tuples is sampled from the prioritized experience replay buffer for updating the parameters of the deep Q-network. Finally, the greedy coefficient is updated. Every C steps, the parameters of the target network are synchronized with those of the deep Q-network.

After training, each host can make real-time routing planning and topology adjustment decisions based on its own deep Q-network. The application process of the proposed method in this paper is shown in Algorithm 3. In each time interval, the target hosts and bandwidth requests of each host are initialized first, forming the current state s_t. Then, each host selects the optimal action a_t based on the deep Q-network. The SDN controller issues instructions according to a_t to update the flow table, followed by a state transition. Relying on a well-trained deep Q-network, each host can make real-time optimal decisions for load balancing and topology adjustments.

5 Case studies and analysis

In this section, we evaluate the performance of the proposed load balancing and topology adjustment strategy. We first show the experimental settings and metrics. Then, the convergence process of deep Q-networks is present. Finally, we carry out

```
Initialize the well-trained deep Q-networks for
each host.
Receive the target hosts and bandwidth requests of
each host.
Input the state s_t to respective deep Q-network
and get the output a_t with the largest Q-value.
The SDN controller sends instructions to update
the flow table. Load balancing and dynamic network
topology adjustment are completed.
```

Algorithm 3. The application process of the proposed method.

TABLE 1 The hyperparameters for training the deep Q-networks.

Parameter	Value
Number of training episodes	3,000
Time steps in one episode	3
Learning rate	0.001
Discount factor	0.9
Replay buffer size	5,000
Minibatch size	16

simulation experiments under different traffic loads and analyze the results.

5.1 Experimental settings

In the experiments, the computer operating system used is Ubuntu 22.04. The experimental simulation environment utilizes the Mininet simulation software, the Ryu controller, and the OpenFlow 1.3 protocol. We construct a four-fork fat-tree network structure for experimentation, as shown in Figure 2. The number of hosts is 16, with 20 switches, and all link bandwidths are set to 100 Mbps/sec. The simulations were completed by a PC with an Intel Core (TM) i5-12500 CPU @ 3.0 GHz with 16.00 GB RAM, RTX GeForce 2060 SUPER.

During the training of each deep Q-network, we set each episode to consist of three discrete time steps, with a total of 3,000 training episodes. The deep Q-network adopts a uniform fully connected deep neural network. The number of neurons in the input and output layers are 32 and 54, respectively, while the middle layer is a hidden layer with 128 neurons. The learning rate of the deep Q-network is set to 0.001. The reward discount factor is set to 0.9. The exploration coefficient is initially set to 1 and is reduced by 0.0006 after each time step. After decreasing to 0.01, it remains constant. Every 10 steps, the parameters of the target network are synchronized with the deep Q-network. Furthermore, the capacity of the priority experience replay buffer is set to 5,000 with the minibatch size is set to 16. The hyperparameters for training the deep Q-networks are presented in Table 1.

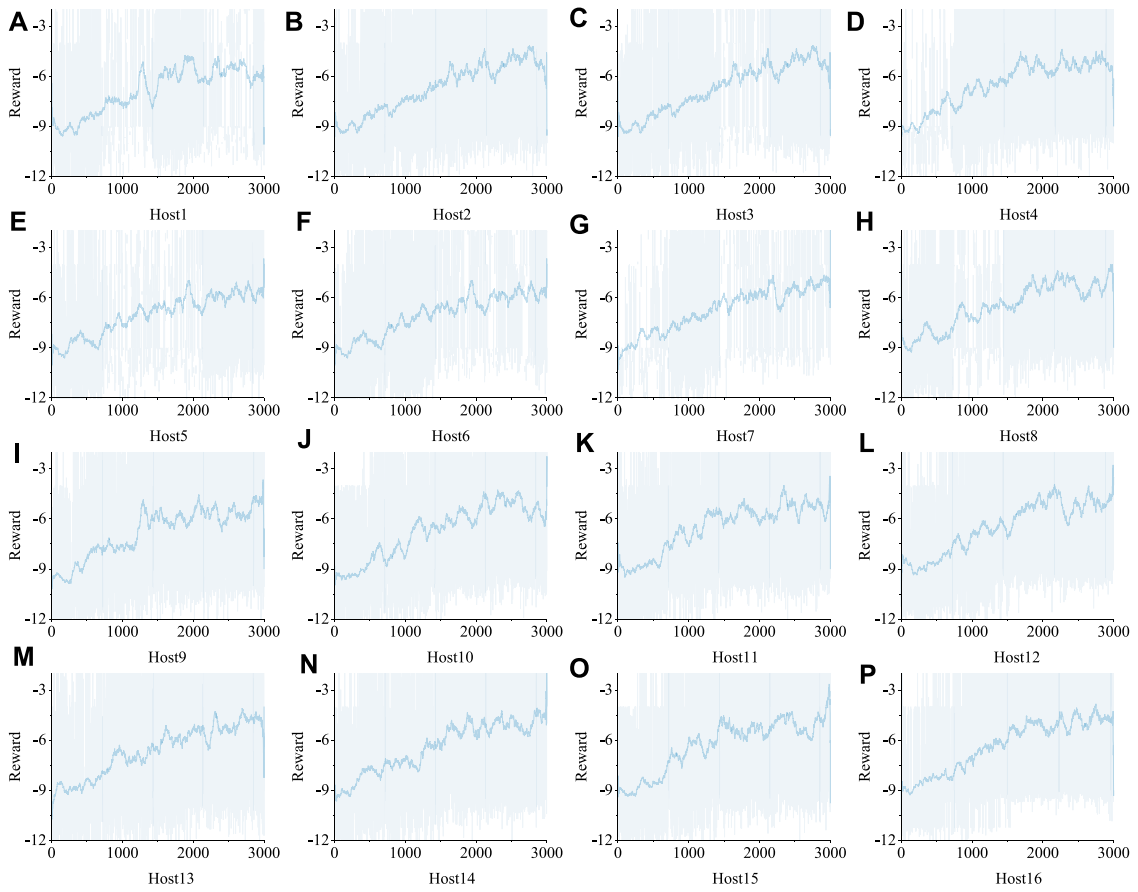

FIGURE 3
The convergence of the DNN training. **A–P** represent the convergence curves of Host1 to 16 respectively.

5.2 Neural network training convergence

First, we evaluate the convergence of the DNN training, with results presented in Figure 3. We have plotted the cumulative rewards for each of the 16 hosts using the deep Q-network over the course of the training episodes. As shown in Figure 3, during the initial phase of training, the cumulative rewards are relatively low, with agents predominantly adopting random strategies for extensive exploration. As training progresses and agents accumulate more experience, the cumulative rewards gradually increase. Notably, after 2,000 episodes, the rewards attained by each agent begin to stabilize and converge. This indicates that during the training process, the agents have learned the optimal strategy. The trained deep Q-networks can thus provide decision-making guidance for load balancing and topology adjustment for each host.

5.3 Load balancing effect analysis

To best restore the actual traffic conditions in the data center network of the power information system, this paper selects the random traffic pattern for simulation experiments. Specifically, each host injects a UDP data stream into any target host in the network with an equal probability. In Mininet, we set the traffic load ratio of each host to the link bandwidth between 0.1 and 0.9 to evaluate network performance under different traffic stress levels. The Iperf tool is used to obtain network testing metrics. Experiments are conducted multiple times under the premise of regular network link allocation, and the average measurement values are taken.

Equal-cost multi-path (ECMP) is a classic routing technique used to forward data traffic along multiple equivalent paths. The path selection strategy of ECMP has various methods such as hashing, polling, and based on path weights. The shortest path for traffic forwarding is usually set with the same routing priority, and each switch makes independent decisions for each hop. To comprehensively evaluate the performance of the algorithm proposed in this paper, we use three metrics: average bandwidth utilization, average jitter, and average packet loss. These metrics are then compared between ECMP and the algorithm proposed in this paper.

(1) Average Bandwidth Utilization: The average bandwidth utilization refers to the ratio of the data volume actually received by the target host to the data volume sent by the source host. The data volume received by the target host varies depending on the network conditions, reflecting the

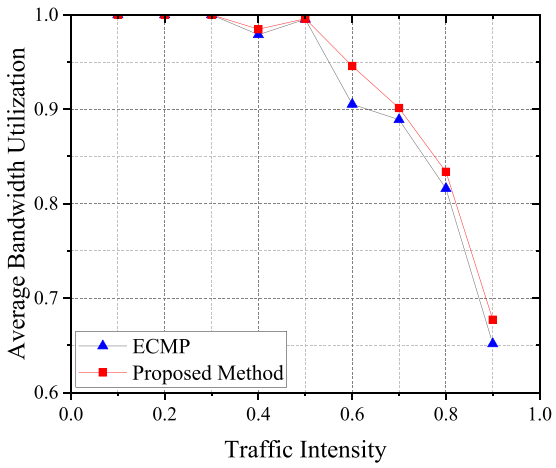

FIGURE 4
Average bandwidth utilization.

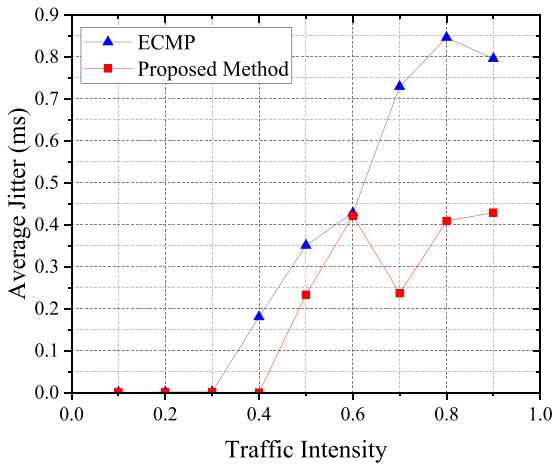

FIGURE 5
The variation of average jitter.

actual bandwidth resources enjoyed by the data stream during network transmission. The data volume sent by the source host corresponds to the designated sending bandwidth. The formula for calculating the average bandwidth utilization is as follows:

$$\eta = \frac{1}{16}\sum_{i=1}^{16}\frac{w_{i,actual}}{w_{i,applied}} \tag{9}$$

Where η represents the average bandwidth utilization, $w_{i,actual}$ denotes the actual bandwidth value of the data transmission for the *ith* host, and $w_{i,applied}$ represents the bandwidth request value of the *ith* host. The average bandwidth utilization is compared to assess the quality of network performance; the larger its value, the better the network performance and the more effective the load balancing.

(2) Average Jitter: The average jitter refers to the average time taken for all data streams in the network to travel from the sender to the receiver. The formula for its calculation is as follows:

$$\varphi = \frac{1}{16}\sum_{i=1}^{16}\left(t_{i,r} - t_{i,s}\right) \tag{10}$$

Where φ is the average jitter, $t_{i,r}$ denotes the reception time of the data stream for the *ith* host, and $t_{i,s}$ represents the starting transmission time of the data stream for the *ith* host. The average transmission delay can be used to measure the degree of link congestion. The smaller its value, the less likely it is for network congestion to occur, indicating a more effective load balancing.

(3) Average Packet Loss: The average packet loss is the ratio of the volume of data that failed to transmit within a unit of time to the volume of data sent. The formula for its calculation is as follows:

$$\delta = \frac{1}{16}\sum_{i=1}^{16}\frac{d_{i,l}}{d_{i,s}} \tag{11}$$

Where δ is the average packet loss, $d_{i,l}$ denotes the volume of data that failed to transmit for the *ith* host, and $d_{i,s}$ represents the volume of data sent by the *ith* host. Serving as a crucial metric for evaluating network performance, the average packet loss rate reflects the system's processing capability. The smaller its value, the better the load balancing effect.

Experiments are conducted at different traffic intensities and the variation of average bandwidth utilization is obtained as shown in Figure 4. As can be seen from the figure, the average bandwidth utilization is close to 1 when the traffic intensity is lower than 0.3. This is due to the relatively low network traffic during this phase, resulting in a minimal probability of link congestion. As the traffic intensity increases, the average bandwidth utilization gradually decreases, indicating a high-load state in the links. Under these circumstances, the method proposed in this paper achieves a higher average bandwidth utilization rate compared to the ECMP algorithm, signifying superior load balancing performance of our method over ECMP.

The variation of average jitter are shown in Figure 5. The graph reveals that at low traffic intensities, both the traditional ECMP algorithm and the method proposed in this paper deliver commendable data transmission outcomes, with the transmission jitter nearly being zero. As the traffic intensity surges, the overall network load increases, potentially leading to link congestion and consequently a general rise in the average transmission jitter. Compared to the ECMP algorithm, our proposed method is adept at selecting the optimal transmission path, effectively minimizing the rate of delay escalation.

Changes in the average packet loss are portrayed in Figure 6. As evident from the graph, in comparison with the ECMP algorithm, the proposed approach effectively reduces the packet loss. This is because the ECMP algorithm evenly distributes the traffic load on each link, overlooking the demands of individual hosts and the network's condition, which can easily lead to network congestion. In contrast, our proposed approach dynamically selects routing paths by comprehensively considering the traffic requests of each host, thereby reducing the probability of link congestion. Summing up

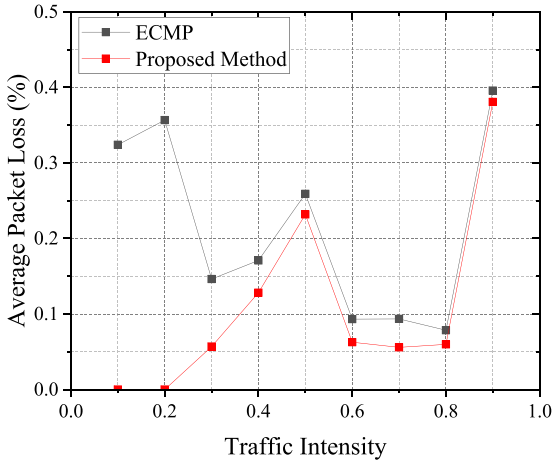

FIGURE 6
The convergence of the DNN training.

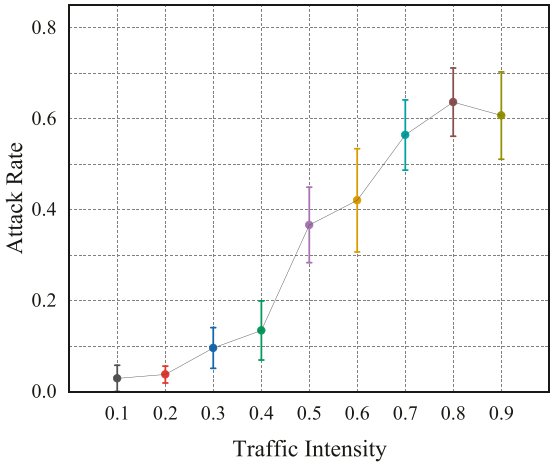

FIGURE 8
The attack rate of the proposed method.

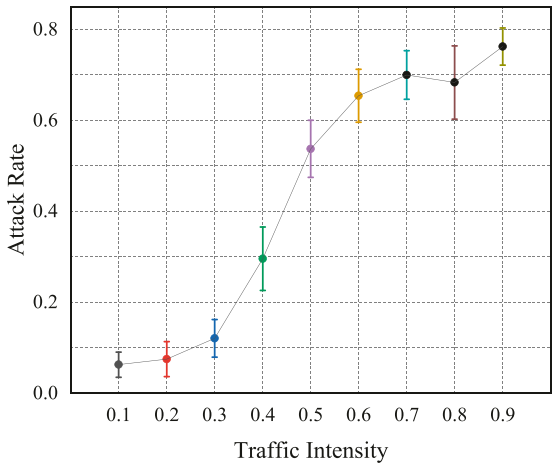

FIGURE 7
The attack rate of the ECMP algorithm.

the analysis, it is evident that the method proposed in this paper achieves superior load balancing results compared to the traditional ECMP algorithm.

5.4 Topology adjustment effect analysis

In addition to load balancing, another significant objective of the method proposed in this paper is to defend against potential attacks by dynamically adjusting network topology. To assess its defense capabilities against flood attacks, we conduct experiments under various traffic intensities. We select a single host as the attacker and adjust the topology using our method, subsequently comparing the success rates of flood attacks before and after the topological adjustments. The success rate of a flood attack is defined as the ratio of the number of other hosts experiencing a decline in average bandwidth utilization to the total number of other hosts. Each host is selected in turn as an attacker for the experiments under different traffic intensities. Experiments were conducted using both the ECMP algorithm and the method proposed in this paper, with results depicted in Figures 7, 8.

In Figures 7, 8, the circular markers represent the average success rate of flood attacks at that particular traffic intensity, while the box area indicates the 95% confidence interval for the flood attack success rate. Comparing the results from Figures 7, 8, it becomes apparent that, at identical traffic intensities, our proposed method effectively reduces the flood attack success rate. This is because our study employs a topology dynamic adjustment strategy based on deep reinforcement learning, which allows for real-time decision-making in accordance with the current network traffic request status. This timely cuts off links connected to terminal nodes that might be under attack, preventing flood attacks from affecting other hosts in the network at the source. Calculations revealed that the average probability of being affected by a flood attack using the ECMP algorithm is 43.244%, while it is 32.159% when employing our method. In comparison to the conventional ECMP algorithm, our method reduces the probability of being attacked by over 11%. This result validates the effectiveness of the approach proposed in this paper in defending against flood attacks.

6 Conclusion

This paper models the problem of load balancing and network topology dynamic adjustment as a Markov decision process, and proposes a method based on deep reinforcement learning to improve the performance, availability and attack resistance of power information systems. According to the characteristics of fat tree topology, we design a data flow path acquisition method based on breadth-first search to construct the action space of each host. Subsequently, we introduce deep Q-network to provide guidance for the optimal strategy of each host. In addition, we also introduce priority experience replay and target network to improve the convergence speed and overall performance of deep

Q-network training. In order to verify the effectiveness of our proposed strategy, we used Mininet to build a typical power information system fat-tree network topology, and conduct a series of case studies and analyses. Experimental results show that our strategy can significantly improve system performance and quickly adjust network topology in the face of network attacks to maintain system stability and availability. The proposed method holds significant promise for broader application in various network infrastructures, particularly in sectors demanding high levels of reliability and attack resilience, thus paving the way for its future adoption in more complex and dynamic network environments.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary Material, further inquiries can be directed to the corresponding author.

Author contributions

XL: Conceptualization, Data curation, Formal Analysis, Methodology, Visualization, Writing–original draft. BB: Investigation, Project administration, Resources, Supervision, Validation, Writing–original draft. WC: Funding acquisition, Resources, Software, Writing–review and editing. DL: Funding acquisition, Project administration, Resources, Supervision, Writing–review and editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. The work was supported in part by State Grid Information and Telecommunication Group scientific and technological innovation projects “Research on Power Digital Space Technology System and Key Technologies” (SGIT0000XMJS2310456).

Conflict of interest

Authors XL, BB, WC, and DL were employed by State Grid Information and Communications Industry Group Co., Ltd.

Publisher’s note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Chakravarthy, V. D., and Amutha, B. (2022). A novel software-defined networking approach for load balancing in data center networks. *Int. J. Commun. Syst.* 35, e4213. doi:10.1002/dac.4213

Choi, H., Kim, M., Lee, G., and Kim, W. (2019). Unsupervised learning approach for network intrusion detection system using autoencoders. *J. Supercomput.* 75, 5597–5621. doi:10.1007/s11227-019-02805-w

Dash, B., Ansari, M. F., Sharma, P., and Ali, A. (2022). Threats and opportunities with ai-based cyber security intrusion detection: a review. *Int. J. Softw. Eng. Appl. (IJSEA)* 13, 13–21. doi:10.5121/ijsea.2022.13502

Ejaz, S., Iqbal, Z., Shah, P. A., Bukhari, B. H., Ali, A., and Aadil, F. (2019). Traffic load balancing using software defined networking (sdn) controller as virtualized network function. *IEEE Access* 7, 46646–46658. doi:10.1109/ACCESS.2019.2909356

Fang, C., Cheng, C., Tang, Z., and Li, C. (2019). Research on routing algorithm based on reinforcement learning in sdn. *J. Phys. Conf. Ser.* 1284, 012053. doi:10.1088/1742-6596/1284/1/012053

Fanlin, M., and Wei, Y. (2020). “Summary of research on security and privacy of smart grid,” in 2020 International Conference on Computer Communication and Network Security (CCNS), Xi’an, China, 21-23 August 2020 (IEEE), 39–42. doi:10.1109/CCNS50731.2020.00017

Gunduz, M. Z., and Das, R. (2020). Cyber-security on smart grid: threats and potential solutions. *Comput. Netw.* 169, 107094. doi:10.1016/j.comnet.2019.107094

Haghnegahdar, L., and Wang, Y. (2020). A whale optimization algorithm-trained artificial neural network for smart grid cyber intrusion detection. *Neural Comput. Appl.* 32, 9427–9441. doi:10.1007/s00521-019-04453-w

Hamdan, M., Hassan, E., Abdelaziz, A., Elhigazi, A., Mohammed, B., Khan, S., et al. (2021). A comprehensive survey of load balancing techniques in software-defined network. *J. Netw. Comput. Appl.* 174, 102856. doi:10.1016/j.jnca.2020.102856

Jamali, S., Badirzadeh, A., and Siapoush, M. S. (2019). On the use of the genetic programming for balanced load distribution in software-defined networks. *Digital Commun. Netw.* 5, 288–296. doi:10.1016/j.dcan.2019.10.002

Khalid, W., Ahmed, N., Khalid, M., Din, A. U., Khan, A., and Arshad, M. (2019). Frid: flood attack mitigation using resources efficient intrusion detection techniques in delay tolerant networks. *IEEE Access* 7, 83740–83760. doi:10.1109/ACCESS.2019.2924587

Li, B., Wu, Y., Song, J., Lu, R., Li, T., and Zhao, L. (2020). Deepfed: federated deep learning for intrusion detection in industrial cyber–physical systems. *IEEE Trans. Industrial Inf.* 17, 5615–5624. doi:10.1109/TII.2020.3023430

Mendonça, R. V., Teodoro, A. A., Rosa, R. L., Saadi, M., Melgarejo, D. C., Nardelli, P. H., et al. (2021). Intrusion detection system based on fast hierarchical deep convolutional neural network. *IEEE Access* 9, 61024–61034. doi:10.1109/ACCESS.2021.3074664

Nisar, K., Jimson, E. R., Hijazi, M. H. A., Welch, I., Hassan, R., Aman, A. H. M., et al. (2020). A survey on the architecture, application, and security of software defined networking: challenges and open issues. *Internet Things* 12, 100289. doi:10.1016/j.iot.2020.100289

Pontes, C. F., De Souza, M. M., Gondim, J. J., Bishop, M., and Marotta, M. A. (2021). A new method for flow-based network intrusion detection using the inverse potts model. *IEEE Trans. Netw. Serv. Manag.* 18, 1125–1136. doi:10.1109/TNSM.2021.3075503

Priyadarsini, M., Mukherjee, J. C., Bera, P., Kumar, S., Jakaria, A., and Rahman, M. A. (2019). An adaptive load balancing scheme for software-defined network controllers. *Comput. Netw.* 164, 106918. doi:10.1016/j.comnet.2019.106918

Rupani, K., Punjabi, N., Shamdasani, M., and Chaudhari, S. (2020). “Dynamic load balancing in software-defined networks using machine learning,” in Proceeding of International Conference on Computational Science and Applications: ICCSA 2019 (Springer), 283–292. doi:10.1007/978-981-15-0790-8_28

Tufail, S., Parvez, I., Batool, S., and Sarwat, A. (2021). A survey on cybersecurity challenges, detection, and mitigation techniques for the smart grid. *Energies* 14, 5894. doi:10.3390/en14185894

Wazirali, R., Ahmad, R., and Alhiyari, S. (2021). Sdn-openflow topology discovery: an overview of performance issues. *Appl. Sci.* 11, 6999. doi:10.3390/app11156999

Xu, Y., Cello, M., Wang, I.-C., Walid, A., Wilfong, G., Wen, C. H.-P., et al. (2019). Dynamic switch migration in distributed software-defined networks to achieve controller load balance. *IEEE J. Sel. Areas Commun.* 37, 515–529. doi:10.1109/JSAC.2019.2894237

Xue, H., Kim, K. T., and Youn, H. Y. (2019). Dynamic load balancing of software-defined networking based on genetic-ant colony optimization. *Sensors* 19, 311. doi:10.3390/s19020311