Computer Networks 236 (2023) 110034

Contents lists available at ScienceDirect

Computer Networks

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

A path selection scheme for detecting malicious behavior based on deep reinforcement learning in SDN/NFV-Enabled network☆

Man Li, Shuangxing Deng, Huachun Zhou *, Yajuan Qin

School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China

ARTICLE INFO

Keywords:
Software defined network (SDN)
Network function virtualization (NFV)
Deep reinforcement learning (DRL)
Service function chain (SFC)

ABSTRACT

The SDN/NFV network is prone to different types of attacks. The Distributed Denial of Service (DDoS) attack has the most severe impact as it can overwhelm the critical components of SDN/NFV to degrade its performance. We propose a closed-loop security architecture (SFCSA) and virtualize detection methods as network service functions in this article. Combining the detection methods forms detection paths, in which different detection paths affect security performance differently. Further, we model the path selection problem as a Markov Decision Process, where the reward balances the malicious traffic detection capability and end-to-end latency. Then, an integrated deep reinforcement learning and convolution neural network path selection algorithm (CNNQ) is proposed. Furthermore, we define a total path malicious traffic detection capability metric. The defined metrics and common metrics are applied to evaluate the building prototype, with the corresponding experimental results demonstrating that the detection performance when combining multiple detection modules outperforms a single detection-based module. Besides, we verify the effectiveness of the CNNQ method under various DDoS attacks scenarios and present the fine-grained classification results of the selected detection modules.

1. Introduction

The software defined network (SDN) is a promising network paradigm that decouples the data and control planes [1]. This technology has the global visibility of the network and can manage networks dynamically. The network function virtualization (NFV) separates network service functions (NSF) from dedicated hardware devices and runs them in commodity servers [2]. The integration of SDN and NFV aims to optimize network resource allocation, automate network management tasks. This can lead to more agile and scalable network infrastructures capable of adapting to users needs. However, this network is still prone to different types of attacks. The DDoS attack has the most severe impact as it can overwhelm the critical components of SDN/NFV to degrade its performance [3].

In 2021, the DDoS threat report [4] indicated that the frequency of DDoS attacks presented a rapid growth tendency over four years, during which the attack growth rate increased by 50%. Thus, DDoS remains an ever-growing tendency to affect enterprises, governments, and personal life. Hence, there is a dire need for a malicious traffic detection method against cyber risks and security issues [5].

The SDN/NFV technology can connect different service functions in a sequenced manner, steering network traffic through a predefined set of service functions to process the traffic, which is known as Service Function Chaining (SFC). Faced with dynamic security threats, the SFC architecture satisfies various security demands according to Internet Engineering Task Force (IETF) RFC 7665 [6]. RFC 8329 [7] describes the Interface to Network Security Functions (I2NSF) framework and analyses how to instantiate and deploy the NSFs. Combining I2NSF and SDN/NFV can deploy NSFs, combine NSFs, and enforce security policies, extending the traditional security mechanisms, thus posing a promising closed-loop cyber security management solution.

A single detection module that cannot detect diverse security threats. This cyber security management solution turns multiple detection modules virtualized into multiple NSFs, in which different NSFs have different effects on security performance. This article combines NSFs and forms detection paths, which can improve the network detection performance. Different detection paths make an obvious difference in detection performance. The path selection against DDoS is an active research field [8]. Hence, we propose a path selection method for the closed-loop security architecture that solves the three challenges described in the following paragraphs.

The first challenge is that detection modules are integrated into the I2NSF and SDN/NFV framework as separate NSFs. Hence, we develop

☆ This research is supported by the National Key R&D Program of China under Grant No. 2018YFA0701604, and NSFC, China under Grant No. 62341102.
* Corresponding author.
E-mail addresses: li_man@bjtu.edu.cn (M. Li), 21120038@bjtu.edu.cn (S. Deng), hchzhou@bjtu.edu.cn (H. Zhou), yjqin@bjtu.edu.cn (Y. Qin).

https://doi.org/10.1016/j.comnet.2023.110034
Received 10 March 2023; Received in revised form 28 July 2023; Accepted 13 September 2023
Available online 20 September 2023

Table 1
A summary of applying statistical methods for DDoS attack detection.

REF	Attacks type	Detection methods	Advantages	Shortcomings
[9]	TCP SYN	Entropy	Proved the availability of the method	The static threshold were not adaptable to dynamic network traffic, leading to misclassify TCP attack traffic as normal traffic.
[10]	HTTP-Get flood	Threshold	Attacking traffic were accurately identified.	This research was only limited to attacks due to HTTP Get requests.
[11]	SYN	Probability model	Verified the efficiency of detection method	The probability model detected SYN packets based on TCP flag distribution, which not adaptable to different traffic speed in SDN networks.
[12]	DDoS	The entropy variation of the destination IP	Had a reliable detection rate	The threshold is the lowest possible rate of attack traffic. However, the traffic rate varies during different time periods in SDN networks. Thus, the threshold requires dynamic adjustments, and its value can impact the detection performance.
[13]	low-rate DDoS	Generalized entropy	Detection method could improve accuracy	The false positive rate of this method is slightly high. Besides, this article was only limited to low-rate attacks.

a closed-loop security architecture (SFCSA) in the SDN/NFV network. The SDN/NFV technology turns detection modules into NSFs, providing diverse security effects. Furthermore, by combining the SDN/NFV network and the I2NSF framework, network administrators can instantiate and reconfigure NSFs to provide timely and effective security effects. Thus, this closed-loop architecture combines security functions into a path, which can be used to detect security threats.

The second challenge is how to generate a detection path. We use Markov Decision Process (MDP) to capture network state transitions and construct state as the current network state (i.e., bandwidth utilization), traffic features, and detection results. We also define action as the selection and combination of NSFs, and the combination of NSFs is also named the detection path.

The third challenge is how to optimize the MDP formulation. Inspired by an emerging approach, deep reinforcement learning (DRL) [14], we use DRL to optimize MDP formulation. DRL allows agents to explore the environment and learn from experience without human heuristics [15]. This study proposes a convolution neural network (CNN) and deep Q network (DQN) algorithm (CNNQ) to select an optimum detection path. The developed algorithm balances the end-to-end latency and the malicious traffic detection capability. Besides, we model both objectives as reward functions, defined as the weighted malicious traffic detection capability minus the weighted end-to-end latency. The main contributions of this research are as follows:

1. We design an SDN/NFV-enabled closed-loop security architecture (SFCSA) comprising physical, data, control, and knowledge layers. The SDN/NFV technology virtualizes detection modules into NSFs. Besides, the SFCSA combines NSFs and forms detection paths that can detect different types of DDoS.
2. The path selection problem is formulated as a Markov Decision Process (MDP) to capture malicious behavior and a CNNQ-based path selection algorithm is proposed. This CNNQ algorithm can generate an optimum detection path according to the traffic features, detection results, and bandwidth utilization.
3. We define a total path malicious traffic detection capability metric. The proposed metric and common metrics are used to evaluate the building prototype, with the corresponding experimental results proving that combining multiple detection modules is better than relying on a single detection module. Besides, the results demonstrate the effectiveness of our approach in different DDoS scenarios.

The remainder of this article is organized as follows. Section 2 introduces the related work. Section 3 describes the proposed closed-loop security framework SFCSA and the work diagram. Section 4 discusses the system state and the problem formulation, and Section 5 describes the path selection algorithm. Section 6 provides the performance analysis of the proposed approach, and Section 7 concludes this article.

2. Related work

In the integrated SDN/NFV network, many studies have proposed various traditional solutions for DDoS attack detection. This section summarizes the relevant traditional works in detecting attacks in the SDN/NFV environment, focusing on statistical and machine learning.

2.1. Statistical method

Several researchers utilized statistical methods for detecting network behavior. Table 1 lists several works that employ statistical methods for DDoS attack detection. Kumar et al. [9] proposed the entropy-based detection and mitigation method against TCP SYN flooding attacks (SAFETY). In this scheme, the SDN controller received destination IP addresses and TCP flags entropy. Then, the entropy and threshold were compared. The experimental results proved the availability of the SAFETY method. Sanjeetha et al. [10] proposed a solution to detect and mitigate the Hyper Text Transfer Protocol (HTTP)-Get flood attack on the web server in the SDN network. The attacks were detected by checking the number of requests arriving at the web server for some duration. If the number of requests received exceeded a particular threshold, the hosts would be blocked for the attack duration once they generated this attack. The experimental results also showed that the attacking hosts and traffic were accurately identified. Prasenjit Maity et al. [11] introduced a probabilistic technique relying on the central limitation theorem to identify DDoS. The DARPA dataset was used to test the probabilistic model, with the corresponding results revealing the efficiency of this approach. However, this method only detected SYN attacks. Mousavi et al. [12]

presented an effective solution for detecting DDoS attacks based on the entropy variation of the destination IP address in SDN networks. This method provided a reliable detection rate but involved a single attack type. Sahoo et al. [13] deployed generalized entropy to detect attacks at the controller. The test results highlighted that their detection method could improve accuracy. However, their work only focused on low-rate DDoS.

In summary, these current methods [9–13] used common entropy and probability model statistical approaches in DDoS detection, which have been effectively validated and evaluated. However, statistical methods are typically based on the analysis and detection of historical data. Network traffic has the characteristics of diversity, dynamic and uncertainty in SDN networks. In addition, SDN's programmability and centralized control architecture may make it easier for attackers to modify network topology and use SDN controllers to carry out attacks. Hence, statistical methods were not adaptable to dynamic security threats in the SDN environment. Therefore, an intelligent and adaptive detection approach that effectively deals with DDoS threats in SDN/NFV networks must be proposed.

2.2. Machine learning

Accordingly, traditional Machine Learning-based (ML) techniques have also been applied to detection and classification problems [16]. Table 2 presents several works applying ML methods for DDoS attack detection. In [17], the Self Organizing Map (SOM) was used to classify network states without attack labels. However, as an unsupervised learning algorithm, the SOM algorithm can incur high time consumption when classifying DDoS attacks. Tan et al. [18] proposed an ML algorithm based on the K-Nearest Neighbor algorithm (KNN) for detecting DDoS attacks in the SDN, effectively improving the detection accuracy. However, the performance analysis did not include the overall system performance. Li et al. [19] exploited Random Forest (RF) by altering sample weights using the weighted voting mechanism to classify attacks. The experiments revealed that this method improved its detection ability with a lower overhead than existing solutions. Nevertheless, this scheme imposed a large processing burden. Ahuja et al. [20] proposed a hybrid model of support vector classifier with random forest (SVC-RF) which classifies traffic using an SDN dataset. Experimental results show that the hybrid model classifies the traffic with an accuracy of 98.8% and a low false alarm rate. However, this paper did not mention the specific type of DDoS attacks. T. AlMasri et al. [21] presented a method that combines ANOVA and naive Bayes for feature selection and detection of DDoS and probe attacks. The experimental results attained a high accuracy of 86.9% for DDoS attacks and 93.5% for probe attacks. However, the accuracy of detecting DDoS attacks is low.

These solutions [17–21] proposed various ML-based algorithms for DDoS detection in SDN/NFV networks, which have been tested using security datasets. Nevertheless, all these methods involve a single detection method. A single detection method cannot detect all attacks because intruders can use different techniques and methods to avoid detection, making it difficult to identify and defend against multiple types of attacks effectively. Therefore, combining multiple detection methods can achieve higher network security.

As an extension of ML, deep reinforcement learning (DRL) allows agents to explore the environment without human guidance or pre-designed rules, and iteratively improve their behavior strategies through interactions with the environment [15]. DRL has recently become important in cyber security [27]. However, detecting DDoS attacks with DRL technology has not been thoroughly investigated. Therefore, it is necessary to investigate the applicability and effectiveness of DRL in autonomous detection threats in SDN/NFV networks. Table 3 lists several works applying RL methods for DDoS attack detection. Noe et al. [22] suggested an autonomous DRL-based mitigation solution to solve low-rate DDoS threats. The solution was ready for

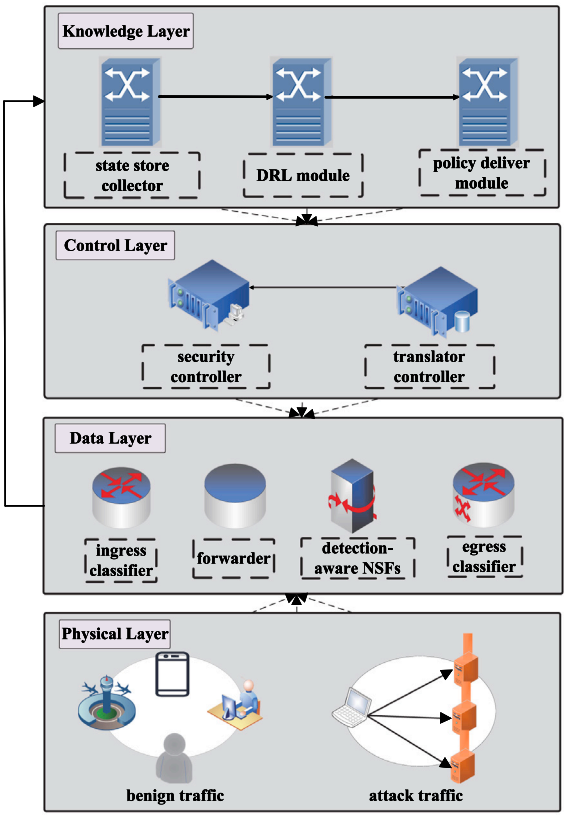


Fig. 1. Closed-loop security architecture SFCSA.

implementation in a medium-sized data center and was tested with security datasets. Nevertheless, it did not consider high-rate attacks, e.g., UDP and SYN floods. Besides, Akbari et al. [23] defined a framework to implement a reinforcement learning (RL)-based solution for mitigating Advanced Persistent Threats (APT) attacks in SDN. The experimental results showed promising potential using this framework to mitigate APT attacks.

Liu et al. [24] proposed an actor-critic deep reinforcement learning approach to detect and defend against DDoS attacks, including SYN, UDP, and ICMP attacks. The state included each port feature of each OpenFlow switch and flow characteristics. Traffic was dropped when the rate exceeded the maximum bandwidth, and the reward included the percentage of benign traffic reaching the victim server and attack traffic reaching the server. The experimental results verified that this approach can effectively mitigate DDoS attack. However, these methods [22–24] were designed and optimized for specific attack types. Due to each attack type's different features and behaviors, it was not easy to classify them accurately using the same model. Therefore, these methods [22–24] are difficult to generalize to other types of attacks.

Zolotukhi et al. [25] proposed an RL agent that processed network state and took actions to redirect certain network traffic in a virtual network. The states comprised the number of ports, flows, and security alert numbers. This method identified and blocked malicious attacks, including SSH password brute-force, DNS tunneling, and slowloris DDoS. However, the specific actions supported by the agent were not described in detail. Malialis et al. [26] deployed multiple RL agents on routers, and the agents were trained to send rate-limit attack traffic to a node in the SDN environment. It is important to note that each agent could not see the whole state, so finding an optimal solution is harder. In contrast, we utilize the global view of SDN to gain an global optimal global policy.

In the past few years, researchers applied DRL to cyber security [22–26], limiting their feasibility and scope. In most cases, they only

Table 2
A summary of applying ML methods for DDoS attack detection.

REF	Attacks type	Detection methods	Advantages	Shortcomings
[17]	DDoS	SOM	Without attacks label	1.all the features were selected manually resulting in increased time for selecting features. 2. only used a single detection method.
[18]	DDoS	KNN	Improved the detection accuracy	The single detection method was not suitable for detecting larger-scale network traffic, resulting in a low efficiency.
[19]	DDoS, probe, U2R and R2L	RF	Improved detection ability with lower overhead	The effectiveness of the detection method was not validated in a prototype system.
[20]	DDoS	SVC-RF	High accuracy and a low false alarm rate	only used a single detection method
[21]	DDoS and probe	naive bayes	High accuracy for probe attacks	low accuracy for DDoS attacks

Table 3
A summary of applying DL methods for DDoS attack detection.

REF	Attacks type	Detection methods	Advantages	Shortcomings
[22]	low-rate DDoS	DRL	Applicable in medium-sized data center	The authors only focused on low-rate attacks.
[23]	APT attacks	RL	This method could mitigate APT attacks	The authors only focused on detecting APT attacks.
[24]	SYN, UDP,and ICMP	actor-critic DRL	Effectively mitigated DDoS	This paper not used NFV technology.
[25]	SSH password brute-force, DNS tunneling, and slowloris DDoS	RL	Redirect certain network traffic	The proposed framework lacked scalability.
[26]	Rate-limit attack traffic	Multiple RL	Utilize the global view of SDN	The method was difficult to generalize to high-rate attack traffic.

focused on detecting certain attacks and not using NFV technology. On the opposite, this study turns detection modules into NSFs. Furthermore, we propose a closed-loop security framework that combines NSFs and form detection paths. When the mixture attack traffic arrives, the SDN/NFV network provides a detection path on demand with the assistance of DRL.

3. System architecture

The SFCSA is a closed-loop architecture, which can detect different types of DDoS. The SFCSA virtualizes detection modules into NSFs and combines NSFs to form detection paths. The path comprises the number and types of detection modules. The excessive number of detection modules can increase latency, and additional types of detection modules decrease detection performance. Thus, this proposed architecture generates an optimum detection path according to the traffic features, detection results, and bandwidth utilization.

This section is divided into two parts: the closed-loop security architecture SFCSA (Section 3.1) and work diagram (Section 3.2). Section 3.1 introduces the functions of each layer in the closed-loop security architecture. Section 3.2 explains the relationship between the SFCSA's three layers and introduces the functions of the NSFs.

3.1. The closed-loop security architecture SFCSA

Based on the SFC and I2NSF framework, the closed-loop security architecture SFCSA is proposed (Fig. 1). Logically, the SFCSA framework comprises a physical layer, data layer, control layer, and knowledge layer.

Physical layer: This layer is located at the bottom of the proposed framework, aiming to generate benign and attack traffic. To this end, we simulate 5G benign traffic and diverse attack categories. The Third Generation Partnership Project (3GPP) and International Telecommunication Union (ITU) model traffic for 5G usage scenarios includes video streaming, home automation, and gaming [28]. Thus, we generate attacks using various tools and scripts, i.e., Hulk, GoldenEye, and Hping3, which can flood the target's resources by overwhelming the network bandwidth with massive request packets or sending a small number of attack packets. In this article, common and sophisticated DDoS attacks are classified as botnet attacks (Botnet), e.g. Ares and Byob, network attacks (NetDDoS), e.g. SYN and UDP, low-rate attacks (LDDoS), e.g. slow body and slow read, distributed reflection denial of service attacks (DRDoS), e.g. Chargen, and Simple Server Discovery Protocol (SSDP), and application layer attacks (AppDDoS), e.g. HTTP-Flood and HTTP-Post. When DDoS attacks enters the network, the detection-aware NSFs will be activated to detect DDoS attacks in the data layer.

Data Layer: The data layer is between the physical and control layers. The data layer contains the NSFs with an ingress classifier, forwarders, detection-aware NSFs, and egress classifiers. In the ingress classifier, the flow table guides the packets passing through a path, and the forwarders are responsible for forwarding the traffic. In this article, the detection-aware NSFs are deployed at the container, i.e., docker, including low-rate attack detection module (LADM), application attack detection module (AADM), botnet attack detection module (BTDM), network attack detection module (NADM), and DRDoS attack detection module (DADM). More details on the detection modules are described in the next subsection. In the egress classifier, the flow table decides which packets arrive at which destination host.

The NSFs receive the low-level configuration policy from the translator controller and the low-level path policy from the security controller via the southbound interface, i.e., OpenFlow. Specifically, the low-level configuration policy first instantiates and configures the NSFs. Then, according to the low-level path policy, incoming benign and attack traffic pass through the selected path, and the benign packets are forwarded to the target host. Finally, after detecting the attack traffic, the security controller regularly records the source IP (SIP), destination IP (DIP), source port (SP), destination port (DP), and protocol (Pro) in the blacklist. When matching the blacklist, the corresponding traffic can be automatically dropped by blocking rules while the others will

be transmitted to the target host. The blacklist is updated every two minutes, which promptly adds the corresponding SIP, DIP, SP, DP, and Pro features of new malicious traffic to the blacklist. Moreover, when traffic is no longer considered malicious traffic, the corresponding features will be promptly removed from the blacklist to maintain the blacklist's accuracy and effectiveness.

Control Layer: The control layer is the second layer of the proposed framework. Logically, the control layer comprises the translator and security controller. The translator controller receives the high-level policy via a secure protocol, i.e., Netconf that can be derived from the policy delivery module located at the knowledge layer. Further, the translator controller converts the high-level policy into low-level policy. The low-level policy comprises detailed data (i.e., path policy) and operation capabilities (i.e., configuration policy), in which the path policy includes the type and number of NSFs. Inspired by the I2NSF framework, the translator controller sends the path policy via a channel supported by the network protocol (i.e., OpenFlow) to the security controller. According to the path policy, the security controller decides how to modify the original rules for incoming new traffic to guide attack traffic detection. The translator controller sends the configuration policy to the data plane via protocols (i.e., Netconf), in which the configuration policy can instantiate and deploy NSFs.

Knowledge Layer: As the top layer of the SFCSA framework, the knowledge layer comprises three parts: state store collector (SSC), DRL module, and policy delivery module. The main purpose of this component is to generate a security path based on the collected traffic statistic reports.

Accordingly, the SSC is a repository in which this database gathers traffic statistic reports from the NSFs in the data layer, including the characteristics of DDoS attacks, bandwidth utilization, and detection results. A summary of information is timely and periodically updated at the SSC before being delivered to the DRL module.

The DRL module solves the path selection problem efficiently. First, the path selection problem is expressed as an MDP process presented in Section 4. Second, to balance the malicious traffic detection capability and the end-to-end latency, the CNNQ algorithm generates the optimal path based on the traffic statistic reports under attack scenarios. Finally, the policy delivery module provides the optimal security path to the control plane.

In summary, the data plane gives the traffic statistic reports to the knowledge plane. Based on the DRL module, the traffic statistic reports adjust the detection path to improve the malicious traffic detection capability and reduce latency. Thus, the framework can form a closed-loop architecture between the data and knowledge planes.

3.2. Work diagram

The proposed workflow is illustrated in Fig. 2, which can be achieved in a learning environment. In the first step, when the tools or scripts generate DDoS in the physical layer, the SSC sends a subscription command to the NSFs and collects monitoring data from NSFs. Based on the CNNQ method presented in Section 5, the DRL module processes and analyzes such data to provide an optimal path. The DRL module delivers the corresponding results, i.e., a path to the policy deliver module. The latter converts the results into high-level policy and sends them to the control layer's translator controller by security channels or protocols (Netconf).

In the second step, the policy extractor extracts data from the high-level policy in the control layer. Then, the policy mapper maps the data to the corresponding path and configuration data. Based on the path and configuration data, the policy constructor generates a low-level policy, i.e., XML file. The path information of low-level policy is transferred to the security controller for updating the flow table. Then, the policy constructor sends the configuration information of the low-level policy to the NSFs in the data layer.

The data plane can understand and configure the configuration policy, and the security controller receives the path policy and updates the flow table. Then, the flow table can be delivered to the data plane, in which traffic is forwarded to the selected NSFs (i.e., path 1 to 3 in Fig. 2) based on the flow table. The same operation flow of each detection-aware NSF is exhibited in the right part of Fig. 2 whereas their detection methods are different.

Further details on the detection-aware NSFs are described as follows:

1. The parsing module is responsible for listening to transmitted traffic. Additionally, Tcpdump [29] periodically collects raw traffic using a pcap file.
2. We utilize an open-source software called CICFlowMeter [30] to extract flow-based features based raw traffic, including 84 statistical attributes, e.g., timestamp, source port, destination port, source IP, destination IP, flow duration, max, mean, and min values of packet's size.
3. To ensure data quality, data preprocessing is responsible for cleaning flow-based features, including normalization and standardization. Except that AADM applies to the standardization method, the four modules use the normalization method.
4. The next step involves feature selection. Feature selection aims to extract and gather the most representative network features beneficial for detection in each detection module. LADM exploits XGBoost and Random Forest (RF) [31], while AADM utilizes randomized trees (ET), XGBoost, and RF [32]. Moreover, BTDM uses RF, ET, XGBoost, boruta, Gradient Boosting Decision Tree (GBDT), spearman, and kendall [33], and NADM uses malicious behavior analysis [34]. DADM leverages Recursive Feature Elimination Cross Validation (RFECV) [35].
5. The selected features are extracted into the well-trained model to finely classify the traffic. A well-trained model is a machine learning or deep learning model trained on sufficient historical attack traffic and can accurately classify new attack traffic. It should be noted that the well-trained models employed separately, i.e., LADM, AADM, BTDM, NADM, and DADM, are Convolution Neural Network and Random Forest (CNN-RF) [31], stacking ML [32], stacking DL [33], CNN-Attention [34], and XGBoost [35].
6. The well-trained model can automatically learn the nonlinear relationship between the selected features, which can quickly complete coarse-grained and fine-grained detections. Coarse-grained detection refers to all detection modules distinguishing attack traffic from benign traffic, and fine-grained detection is that attack traffic should be classified as specific types.
7. The well-trained model's evaluation metrics are precision, recall, malicious traffic detection capability (MTDC), and F1-score [34].
8. If SIP, DIP, SP, DP, and Pro traffic features are in the blacklist, the malicious traffic will be dropped, in which normal traffic has not interfered with attack traffic, and benign traffic can smoothly reach the destination hosts.

4. Markov decision process

The knowledge layer is the core of the SFCSA framework, where the DRL module solves the path selection problem. In this section, the path selection problem is modeled as an Markov Decision Process.

This section comprises system state (Section 4.1) and Markov Decision Process Formulation (Section 4.2). Section 4.1 defines the system state in the Markov Decision Process. Section 4.2 defines the actions and rewards in the Markov Decision Process.

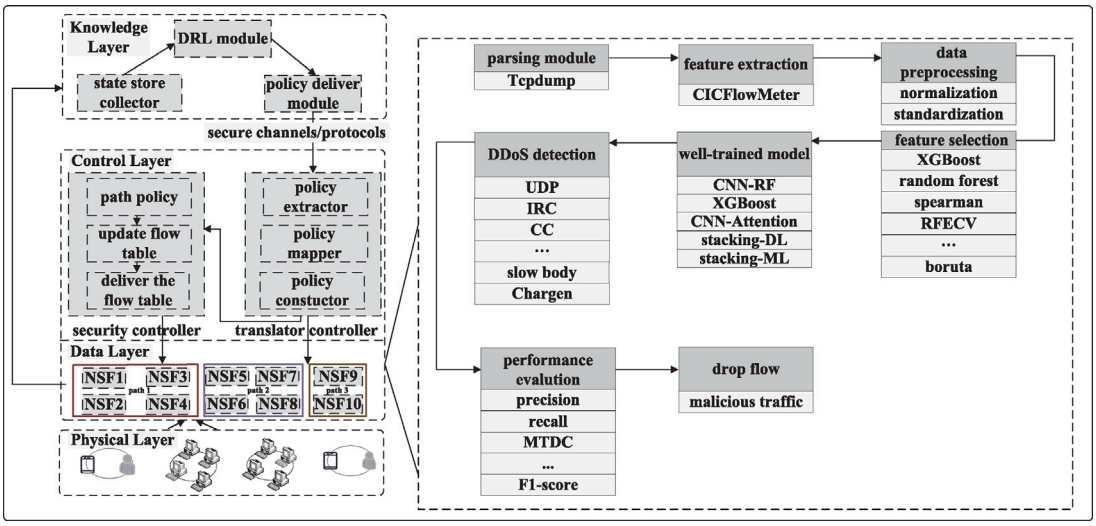


Fig. 2. Workflow diagram of the SFCSA architecture.

4.1. System state

We collect the traffic statistics reports at time t and define the general state of the network S_t as follows:

$$S_t = \{\Psi, \xi, Y\}_t, \tag{1}$$

where Ψ_t represents the features of DDoS attacks, ξ_t^g represents detection results of the gth detection-aware NSF, and Y_t stands for bandwidth utilization. Next, the details of every component are described as follows:

(1) The features of DDoS attacks Ψ

The IETF lists well-known protocols, e.g., User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The malicious hosts execute attack tools or scripts based on these protocols to generate attack traffic in the SDN/NFV network. The attack packets incorporate an IP packet header and payload, with the packet payload including either data or a higher-level protocol (e.g., HTTP).

The packet features are used to build traffic profiles, efficiently classifying diverse traffic [36]. Moreover, since malicious traffic behavior differs from benign traffic behavior, many researchers extract the packet features and model traffic behavior [37]. Hence, we should check the inter-packet relationship to identify attack behavior efficiently. Inspired by [38], we define a structure storing the packet of the necessary information as follows:

$$P = \langle \mathrm{SIP}, \mathrm{DIP}, \mathrm{Pro}, \mathrm{PKT}_{\mathrm{src}}, \mathrm{PKT}_{\mathrm{dir}}, \mathrm{Timestamp} \rangle, \tag{2}$$

The features in the structure are at a packet level, capable of describing the packet characteristics. P represents the required information set of each packet received at the ingress classifier, SIP is the IP of the source host, and DIP is the IP of the destination host. Pro denotes the protocol of each packet, and PKT stands for the direction of packets. Hence, $\mathrm{PKT}_{\mathrm{src}}$ is a forward packet from the source host to the destination host, and $\mathrm{PKT}_{\mathrm{dir}}$ is a back packet from the destination host to the source host. Timestamp is the current time at which Tcpdump captures a packet at the ingress classifier.

Let the network have w source hosts and z destination hosts. $b_{w,z}$ represents a connection between the source hosts w and the destination hosts z. The host connection set B as follows:

$$B = \{\mathrm{NC}, \mathrm{AC}\} = \left\{b_{1,1}, b_{1,2}, \ldots, b_{w,z}\right\}, \tag{3}$$

where NC and AC separately denote the normal connection set and the attack connection set.

For a more detailed network traffic analysis, we extract data packets that conform to packet structure P from the host connection set B at the ingress classifier. This process helps filter out corrupted packets caused by network faults.

The extracted packets form a packet set, denoted as listpacket. We investigate the listpacket and select common features of different DDoS attacks to represent the difference between benign and malicious traffic effectively. They are introduced as follows:

(1) Packet size: Packet size varies for different types of attack traffic. Packets are divided into several segments according to their size. For Botnet, massive HTTP-Post messages are generated from bot machines and sent to the servers, which exhausting network resources. Thus, their sizes belong to the same segment. When botnet attacks happen, the communication behavior between the bot hosts and servers will affect the packet size.

For LDDoS, the attack hosts send a burst of small requests to web server, resulting in malicious behavior with similar characteristics. When legitimate requests are launched and sent to the target servers, the packet size of the transmitted packets varies between the legitimate hosts and web servers to gain different requests.

For DRDoS, the amplification impact of attacks derives from the fact that small requests can initiate many UDP packets, harming the network. These attacks can be classified into two categories: (1) amplification with varied queries containing different contents and (2) amplification with duplicated queries with the same content. However, under two circumstances, the size of the response packets may be analogous to producing prominent amplification. Thus, the packet size should be a representative feature.

For NetDDoS, the attackers consume target resources, e.g., port, memory, and bandwidth, to achieve denial of service. In our case, we leverage attack tools to craft attack packets by setting the same packet size. For AppDDoS, attackers send many incomplete HTTP requests to target servers or websites, in which attackers prevent the benign host from exchanging useful real-time information with their corresponding servers. During this process, the size of packets is similar to archive a good level of attack. Normal users establish legitimate connections with different requests to generate legitimate traffic, and the transmitted packet size differs. Thereby, packet length is chosen as one of the distinctive features. The set of packet sizes is constructed as follows:

$$\mathrm{size}(t) = \{size_n^t | n = 1, 2, \ldots, N\}, \tag{4}$$

where $size_n^t$ is the length of the nth packet in the time window t, and N is the number of packets received at the ingress classifier.

(2) Time To Live (TTL): Each application generating benign traffic adjusts the TTL value independently. In [39], the authors showed that the number of TTLs observed differs, whereas the TTL value is usually

between 40 and 255 under a DDoS attack. For AppDDoS, LDDoS, and NetDDoS, we simulate these three malicious types with tools or scripts, and thus, their TTL usually belongs to a similarity value.

For Botnet, DDoS programs are generally pre-built, in which each infected bot follows an almost identical attack pattern. For DRDoS, attack hosts send a small number of requests to victims so that victims can generate many responses. Therefore, the TTL values of attack traffic simulated by all attack hosts are determined. The set of TTL is expressed as follows:

$$\mathrm{TTL}(t) = \{ttl_n^t | n = 1, 2, \ldots, N\}, \tag{5}$$

where ttl_n^t is the TTL value of the nth packet in the time window t.

(3) Destination Port: In benign traffic, legitimate hosts send different queries to servers. For example, the port number for web page browsing is 80, the port number for transferring files is 21, and the port number for email service is 110.

For DRDoS, hosts send much attack traffic to the target servers, in which protocol often corresponds with a fixed port number. For instance, the port number of the SSDP protocol is 1900, and the port number of the Chargen protocol is 19. Thus, the destination port number is more concentrated in the attack traffic so that the destination port number can be used as a distinct packet feature.

The AppDDoS, Botnet, and LDDoS are generated based on HTTP protocol. The port number for HTTP protocol is 80, and the NetDDoS hardness the weaknesses of TCP protocol. The port number for NetDDoS can be set with the attack tools to maintain consistency with other attacks. Moreover, many researchers extract destination ports to distinguish benign traffic from malicious traffic [40]. The set of ports is designed as follows:

$$\mathrm{port}(t) = \{port_n^t | n = 1, 2, \ldots, N\}, \tag{6}$$

where $port_n^t$ is the port value of the nth packet in the time window t.

(4) Packet interval time: In the typical application, the packet arrival time varies depending on the service type. For example, the arrival time of VoIP is short to avoid the undesirable effects of jitter [41]. When the users intend to browse the web page, the normal users will send requests to the server to browse the web page, and the users will stay on the web page over some time, at which the packet interval time will slow down.

For AppDDoS, the attack machines quickly send many requests to the target servers. Accordingly, the packet interval time should regularly change during the attack. For LDDoS, the essence of this attack is to read the fragmented packets or establish incomplete HTTP connections slowly. During this process, there is a fixed number of packets in the connections, and the packet interval time can change fixedly, eventually achieving an LDDoS attack.

For NetDDoS, the attackers exhaust the network resources of the destination hosts by sending many fixed-length packets. A direct consequence of this attack is that the useful information cannot reach its destination successfully or completely. For Botnet, the zombie hosts establish a three-way handshake with a server in the first phase. Then, in the second phase, botnet hosts construct a command and control channel by updating a library file at regular intervals. In the third phase, the zombie hosts execute a command to perform malicious activity. Packet interval time can regularly change in all the phases when preparing malicious activities.

For DRDoS, attackers send forged request packets to servers, and the servers return a large number of response packets with the analogous packet size to victims, so the packet interval is also fixed. Therefore, the packet interval time of benign traffic and attack traffic has an obvious difference. The set of packet interval arrival time (IAT) is defined as follows:

$$\mathrm{IAT}(t) = \{inter_n^t | n = 1, 2, 3..., N\}, \tag{7}$$

where $inter_n^t$ is the packet interval arrival time value of the nth packet in the time window t.

Based on the above discussion, to distinguish between benign traffic and attack behavior, we choose the most promising features as follows:

$$\Omega = [\mathrm{size}, \mathrm{TTL}, \mathrm{port}, \mathrm{IAT}] \tag{8}$$

However, entropy is widely used to help detect DDoS attacks [42]. Considering the packet size as an example, most new-coming packets are transmitted from the source hosts to the destination hosts, concentrating on the packet size. Nevertheless, the packet size is dispersed under normal conditions. Thus, we use entropy to represent the randomness of properties. A low entropy value indicates a concentration distribution, whereas a high entropy value indicates a dispersed distribution. Assuming that the total number of received packets at time t at the ingress classifier is NM_t, NS_n^t denotes the number of the nth packet's packet size value at t. The possibility of the nth packet size is $P_{\mathrm{size}(n)}(t) = \{p_n^t | n = 1, 2, 3, \ldots, N\}$, which can be calculated based on Eq. (8).

$$p_{\mathrm{size}(n)}(t) = \frac{NS_n^t}{NM_t}, \tag{9}$$

Then, we define the entropy value of packet size during time interval t as follows:

$$H_t(\mathrm{size}) = -\sum_{n=1}^{N} p_{\mathrm{size}(n)}(t) \log(p_{\mathrm{size}(n)}(t)), \tag{10}$$

Similarly, the entropy of TTL, port, and IAT can also be calculated. Eventually, in the time window t, the features of the DDoS attacks Ψ are constructed as follows:

$$\Psi_t = [H_t(\mathrm{size}), H_t(\mathrm{TTL}), H_t(\mathrm{port}), H_t(IAT)], \tag{11}$$

The main pseudo code is elaborated in **Algorithm 1**. When attack traffic enters the network, Tcpdump extracts the complete header information from connection $b_{w,z}$ in the time window t. If packets c and d are in the connections, the packets are classified as list packets (Lines 1–7). The SDN-based network exits background traffic, resulting in a difference in its packet structure. These packets should be dropped (Line 9). Then, *rdpcap*() extracts the packet-level features (Line 12). *entropy*() is used to calculate the entropy value of each feature (Line 13) and *normalized*() is used to scale the entropy value. Finally, the entropy value Ψ_t is obtained (Line 15). The packet features are extracted from each packet at the ingress classifier, in which this method is scalable for all packets.

Algorithm 1 Feature Extraction

Input: time window t, traffic
Output: The feature entropy of DDoS Ψ_t
1: Initialize the list of connections B
2: while True
3: listpacket = []
4: st=start time
5: while $(currenttime - st) < t$
6: **if** c and d in the connection $b_{w,z}$ **then**
7: add c and d to the listpacket
8: **else**
9: Drop packets c and d
10: **end if**
11: **if** a packet in the listpacket **then**
12: call *rdpcap*() to extract the feature
13: call *entropy*() to calculate the value
14: **end if**
15: Call *normalized*() to scale the entropy value Ψ_t
16: **return** Output

(2) Detection results of the detection-aware NSFs ξ The F ($F = \{f_1, f_2, \ldots, f_{|F|}\}$) denotes the set of NSFs. $f_{|F|}$ is the egress classifier. Other NSFs with detection-aware include LADM, AADM, BTDM, NADM, and DADM. In each detection-aware NSF, the performance evaluation

metrics include accuracy (Acc), precision (Pre), recall (Rec), and F1-score (Fsc). The Acc is the fraction of correctly classified attack traffic. The Pre indicates the proportion of correctly classified attack traffic out of the total attack traffic predicted. The Rec represents the proportion of correctly detected attack traffic out of the actual attack traffic. The Fsc is the harmonic mean of pre and rec.

Incoming packets enter the selected detection path in the data plane, and the detection results will be updated in the SSC of the knowledge layer. The detection results can guide the DRL module to decide the appropriate path. In the time window t, the detection result of each detection-aware NSF ξ_t^g is calculated as follows:

$$\xi_t^g = \{Acc_t^g, Pre_t^g, Rec_t^g, Fsc_t^g\}, \tag{12}$$

(3) Bandwidth utilization Y

When malicious users send a large number of attack packets to generate abnormal access, if there is no matching flow table entry, PACKET_IN messages will be sent to the security controller to request the processing method for waiting for the path policy, and reply messages are received at the ingress classifier. A great number of reply information may consume a huge amount of bandwidth resources. Thus, we select bandwidth utilization as a state.

Assume that the link's maximum capacity is β (packets/seconds). The ingress classifier periodically samples the received packets (i.e., l_u ($l_u \in [1,+\infty)$) for each connection $b_{w,z}$). By comparing the values received at two instants (i.e., t_1 and $t_1 + t$), we calculate the related the variation of bandwidth utilization $ll_{w,z}(t)$ (packets/seconds). After the security controller receives the request messages, the controller sends reply messages to the ingress classifier, and the ingress classifier receives the number of packets at the time t_1 (i.e., $l_u(t_1)$). Then, after a period t, the ingress classifier receives another reply message and the number of packets at the time $t_1 + t$ (i.e., $l_u(t_1 + t)$). Accordingly, the related variation of bandwidth utilization $ll_{w,z}$ of a connection $b_{w,z}$ is expressed as follows:

$$ll_{w,z}(t) = \frac{l_u(t_1 + t) - l_u(t_1)}{\beta}, 0 \le t_1 \le t_1 + t, \tag{13}$$

The meaning of Eq. (13) is the relative variation of bandwidth utilization rate within every two time windows compared to the maximum bandwidth utilization rate.

In DDoS attacks, attackers send a large volume of attack packets to the SDN networks, which can overwhelm the network resources by flooding the SDN controller with numerous PACKET_IN messages to request flow table installations. As a result, the network's bandwidth utilization rate increases significantly. Therefore, this paper specifically focuses on the case where the related variation of bandwidth utilization $ll_{w,z}$ is rising, namely when $l_u(t_1 + t) > l_u(t_1)$.

According to Eq. (13), at time window t, the corresponding bandwidth utilization set of connections is expressed as follows:

$$Y_t = \left\{ll_{1,1}(t), ll_{1,2}(t), \ldots, ll_{w,z}(t)\right\}, \tag{14}$$

4.2. Markov decision process formulation

The formulation of the MDP includes four tuples: state S, action A, probability P, and reward R. In what follows, the details of each part will be described.

State (S): At time window t, state parameter is defined as:

$$S_t = \left\{\Psi_t, \xi_t^1, \xi_t^2, \ldots, \xi_t^g, Y_t\right\}, \tag{15}$$

where Ψ_t, Y_t, and ξ_t^g are introduced in Section 4.1.

Action (A): The action set involves a finite set of actions. In the proposed framework, action A denotes one of the NSFs. Accordingly, the action set is the same as the set of NSFs. That is to say, $A = \left\{f_1, f_2, \ldots, f_{|F|}\right\}$. The agent executes an action and adds the corresponding detection-aware NSF to the path list. The construction of path list is completed when the action is egress classifier, which means that an episode is terminated.

Probability (P): The probability P is the state transition probability from the current state $s \in S$ to the next state $s' \epsilon S$ when action $a \in A$ is performed.

Reward (R): In our proposed framework, the reward R represents the network performance. The reward set includes malicious traffic detection capability (MTDC), and end-to-end latency. The primary reward function in episode e is shown below.

$$r_e = \rho_c \mathrm{MTDC}_e - \rho_l \mathrm{latency}_e, \tag{16}$$

where ρ_c and ρ_l are the weight parameters used to denote the importance of the two components. The selection of weight parameters is presented in Section 6.

The first part is designed for selecting combined NSFs (i.e., a path) to improve detection performance. Thus, we also add the corresponding MTDC of every detection module in the reward, defined as follows.

$$\mathrm{MTDC} = \sum_{w=1}^{|W|}\sum_{z=1}^{|Z|} u_w^z e_w^z \frac{(\mathrm{AC}_w^z + \mathrm{NC}_w^z)_{Before}}{\mathrm{AC}_{w\,Before}^z - \mathrm{AC}_{w\,After}^z}, \tag{17}$$

where u_w^z is used to indicate whether there is a connection between a source host w and a destination host z. If $u_w^z = 1$, it indicates that a connection is established between a source host w and a destination host z. If $u_w^z = 0$, it indicates that there is no communication between a source host w and a destination host z. e_w^z is used to indicate whether there is malicious traffic between a source host w and a destination host z. If $e_w^z = 1$, it indicates that attack tools or scripts are activated. If $e_w^z = 0$, it indicates that a source host w crafts no attack packets. $(\mathrm{AC}_w^z + \mathrm{NC}_w^z)_{Before}$ represents the summation of attack and normal packets before detecting attack traffic using selected detection-aware NSFs between a source host w and a destination host z. $(\mathrm{AC}_w^z)_{Before}$ is the number of attack packets between a source host w and a destination host z before the selected detection-aware NSFs are conducted. $(\mathrm{AC}_w^z)_{After}$ represents the number of attack packets not correctly detected after the selected detection-aware NSFs are executed between a source host w and a destination host z.

The second part of the objective is end-to-end latency. The end-to-end latency is the time it takes for packets to traverse all NSFs in a detection path, which includes transmitted latency, propagation latency, queue latency, and processing latency. d_1 denotes the transmission latency of the physical link, measured by the real-time network, d_2 represents a propagation latency. The propagation latency is the time it takes for data packets to propagate over the physical link from the source to the destination nodes, which is the ratio of the end-to-end path length to the propagation speed of electromagnetic waves in the channel.

d_3 indicates a queue latency.Under attack traffic, the number of attack packets should increase quickly, resulting in packets accumulation in the queue. According to the first-in-first-out principle, the classifier and forwarder process the packets in the queue. However, their processing capacity is limited. As a result, the packets wait for transmission in the queue, causing an increase in queuing latency.

Moreover, d_4 is a processing latency. During packets traversing the path, each packet will be received by the virtual machine's network card and should be inserted into a network service header (NSH). Then, packets enter an NSF-based docker and remove the NSH. Meanwhile, the packets are decapsulated and detected by an NSF. To forward to the next NSF, packets are encapsulated and added to the NSH. The packets will be forwarded to the next network card and then processed in a loop until the packets are forwarded to the target host. As the number of attack packets increases, the processing latency increases.

In a real network, the distance between the source and destination nodes is much smaller than the propagation speed. Therefore, this study ignores propagation latency. Thereby, the end-to-end latency is represented as follows:

$$\mathrm{latency} = d_1 + d_3 + d_4, \tag{18}$$

Based on the reward function Eq. (16), the future discounted return R at episode e is redefined as follows:

$$R = \sum_{e'=e}^{E} \lambda^{e'-e} r_e, 0 \le \lambda \le 1, \tag{19}$$

The reward function comprises a discount factor λ and immediate reward r_e, where λ balances the value of current and future rewards. Note that the selected NSFs are sorted in descending order of their rewards, with one NSF having the highest reward at the front of the path list.

To find the optimal path, we consider the impact of the types of detection modules on detection performance and the influence of their number on latency in the path. Thus, based on reward function, the objective function of the path selection algorithm is defined as Eq. (20). The objective function is expressed as finding the optimal path corresponding to the maximum reward, which balances between detection performance and latency, aiming to ensure that the optimal path has high detection performance and low latency.

$$F = \max(R) \tag{20}$$

Specifically, we formulate the path selection as an optimization problem, balancing detection performance and latency to find the optimal path. To achieve this objective, we first optimize the types of detection modules to improve detection performance. Secondly, we optimize the number of detection modules to reduce latency. We aim to find the path which maintains the detection performance and low latency. The details of the path selection algorithm will be described in the next subsection.

5. Path selection algorithm

With the MDP model aforementioned, we characterize the system state, action, and reward. Then we need to design an efficient path selection scheme, which can choose the optimal path. Thus, we apply DRL framework to select the detection path.

This paper combines DQN and CNN method (CNNQ) for path selection. The CNNQ method has the perceptual property of deep learning and the decision-making ability of reinforcement learning [43]. Besides, the CNNQ method adopts CNN to approximate the Q-function of state–action pairs, which can extract the data features with convolutional layers and pooling layers and provide automatic learning from data structure [44]. Thus, CNNQ interacts with the environment to select the optimal path that maximizes future reward.

5.1. CNNQ path selection algorithm

The CNNQ path selection algorithm includes a learning process and a detection process. In the former process, we first collect plenty of traffic data under a variety of DDoS attacks, including traffic features, detection results, and bandwidth utilization, to create an offline dataset. Next, we construct a DRL environment based on this dataset, in which the learning agent selects an action using the $\epsilon-$greedy method [45]. Then the agent executes an action and gains a reward. The agent obtains enough experience in a learning loop to train the policy network with optimal network performance. The trained policy network is a well-established model. For the detection process, when incoming traffic enters the network, the SSC monitors and gathers the network state (i.e., s), then state–action values (i.e., $Q(s,a)$) are generated based on the well-established model, and the agent selects a corresponding action to the maximum value. The learning process can be time-consuming, and the learning time is proportional to the number of learning samples. The more the learning samples, the better the well-established model stabilizes. The well-established model can be long-term optimized based on sufficient data. Then, the agent with the well-established model can make an optimal decision to adjust the selected NSFs by observing the network state. In summary, the CNNQ model can generate the detection path.

Fig. 3 illustrates the proposed architecture of CNNQ in the SDN/NFV-Enabled network. The learning process is illustrated in the upper part of Fig. 3. By observing the current system state s_j (step 1), the agent achieves an action with the exploration and exploitation method. Hence, the agent exploits known information to select action a_j with probability ϵ, and explores unknown environments to select action a_j with probability $1-\epsilon$ based on the $\epsilon-$greedy policy. (step 2). The current state s_j moves to the next state s_{j+1}. Meanwhile, the agent gets a reward r_j from the environment and generates state transition (s_j, a_j, r_j, s_{j+1}) (step 3). All the transition experiences are generated by repeating steps 1–3 and stored in the reply memory. When the stored examples are enough, the agent randomly chooses the batch size for evaluation and target network, where batch size is used to specify the number of experience samples per training (step 4). For each learning episode, the gradient descent method updates the target network (step 5). During this process, the parameters of the target Q network should be updated at every certain step (step 6). After learning, the well-trained model will be deployed at the DRL module to adjust the path (step 7). Then, in the detection process, by inputting the state S_t and using the well-trained Q network (step 8), the well-trained model generates multiple Q values and selects the action corresponding to the maximum Q value (step 9). Eventually, the trained model can generate the optimal combination of NSFs (step 10). Next, according to the optimal combination of NSFs, path and configuration information will be created (i.e., attack detection policy) in the policy delivers module. The policy delivers module sends this policy to the environment to verify the performance of the optimal combination of NSFs (step 11). Observing a state in the SDN/NFV environment to deliver an adjusted policy to the environment aligns with the SFCSA architecture's closed-loop characteristic.

5.2. CNNQ learning process

Due to the various network traffic characteristics, the CNNQ method can describe network states comprehensively and in detail, considering the attack behavior features, detection performance, and bandwidth utilization, encouraging the agent to make favorable decisions. In this section, we detail the learning process. Fig. 4 illustrates that the learning agent requires adequate experience to train the model and optimize its performance in the learning process. After learning, we ran this well-trained model to gain the selected NSFs and achieve detection. However, to increase the data utilization ratio and avoid non-convergence, CNNQ leverages experience replay and two network techniques [43] to improve the overall performance of the CNNQ model.

Experience replay: The experience replay strategy is that the agent randomly extracts a small batch size of empirical samples from the experience replay pool. The experience replay pool is a data structure comprising experiences (i.e. $e_j = \langle s_j, a_j, r_j, s_{j+1} \rangle$), that can form a replay memory sequence $D = \{e_1, e_2, \ldots, e_j\}$. The agent learns from the batch size empirical samples and the model parameters are updated using the loss function until the model converges. Thus, the experience replay breaks the correlation between data and increases the data utilization ratio by reusing historical data.

Two networks: The same network evaluation Q value and target Q value will lead to a loop between evaluation values and target values, making convergence challenging. Therefore, the target and evaluation networks approximate the value function using CNN. Our study's CNN structure includes two convolution layers, two pooling layers and two dense layers. At time j, the Q evaluation value calculated by the evaluation network is expressed as $Q(s_j, a_j; \theta)$, where θ denotes the parameters (i.e., bias and weight parameters of CNN) to be updated in the evaluation network. Then, the parameter θ of the Q evaluation network should periodically be copied to the parameter θ^- of the target Q network. To improve the performance of the Q network, the loss

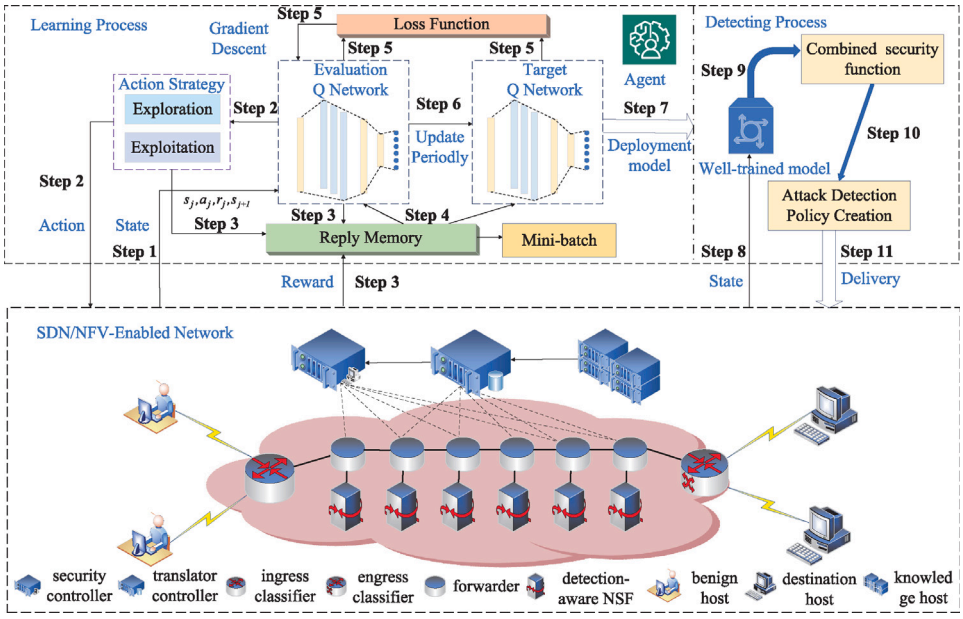


Fig. 3. Proposed architecture of CNNQ in the SDN/NFV-Enabled network.

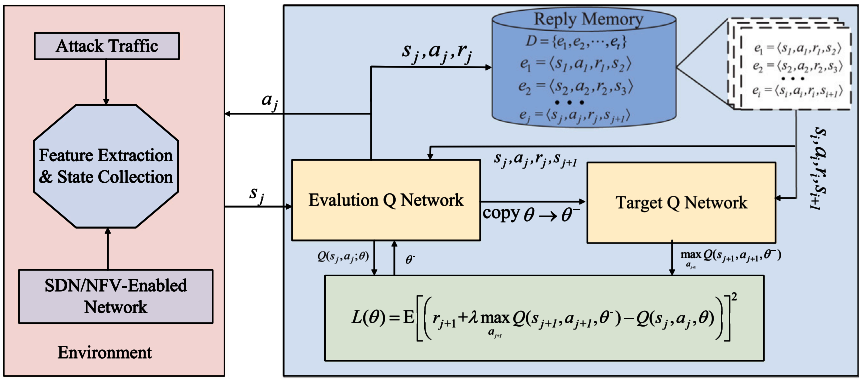


Fig. 4. The learning process of CNNQ.

function is applied to train the CNN parameters. The loss function is expressed as follows:

$$L(\theta) = \mathrm{E}\left[\left(r_{j+1}+\lambda \max_{a_{j+1}} Q(s_{j+1}, a_{j+1}, \theta^{-}) - Q(s_j, a_j, \theta)\right)\right]^2, \tag{21}$$

where $r_{j+1}+\lambda \max_{a_{j+1}} Q(s_{j+1}, a_{j+1}, \theta^{-})$ is the calculated target Q value with parameter θ^{-}. $Q(s_j, a_j, \theta)$ is the calculated evaluation Q value with parameter θ.

The stochastic gradient descent (SGD) is used to update parameters θ, which is applied to optimize the loss function. The SGD is expressed as follows.

$$\begin{aligned}\theta' = \theta + \alpha \big[&(r_{j+1} + \lambda \max_{a_{j+1}} Q(s_{j+1}, a_{j+1}, \theta^{-})) \\ &-Q(s_j, a_j, \theta) \big] \nabla Q(s_j, a_j, \theta),\end{aligned} \tag{22}$$

where α is the learning rate. The learning rate reflects the effect of the evaluation Q value on the current Q value.

The main pseudo code of the learning process is presented in **Algorithm 2**. The path list, evaluation network, target network, and replay memory are initialized in lines 2–5. In an episode, the agent selects an action a_j using the $\varepsilon-$greedy method under the current state s_j (lines 7–8). When the selected action (i.e., NSF) is recorded in the path list L, the agent will again select an action based $\varepsilon-$greedy (lines 10–11). When the selected action is not recorded, the reward is calculated according to Eq. (19), and the replay memory pool D will store experience (s_j, a_j, r_j, s_{j+1}) for future training. A mini-batch is randomly sampled from the replay memory pool D (lines 13–14), and then the SGD method is used to minimize the loss function to update the evaluation network and target network (lines 15–16). We acquire the path list L with unrepeated NSF, and an episode is terminated until the egress classifier is reached (lines 18–19).

By considering the objective of ensuring detection capability and low latency, the agent selects and combines high-performance NSFs to form an optimized detection path. Consequently, the optimization of path selection can improve the system's detection performance and enable it to effectively counteract various DDoS attacks.

5.3. CNNQ detection process

Based on the generated well-trained model, the main pseudo code of the CNNQ-based detection process is presented in **Algorithm 3**.

The input of the evaluation network is the current system state $s_{t'}$ (line 4), and the output is an optimal path (line 5). Then, the performance of the path will be verified in the inner loop (lines 7–15). Based on the selected NSFs, the policy delivery module can generate the high-level policy and forward it to the control plane (line 7). In the control plane, the translator controller translates the high-level policy into low-level policy and delivers it to the security controller

Algorithm 2 CNNQ-based learning process

Input: state set S, action set A, path list L, Episode E
Output: the trained CNNQ model
1: **for** all episodes **do**
2: Initialize the path list L
3: Initialize the evaluation network $Q(\theta)$
4: Initialize the target network $Q(\theta^{-})$
5: Initialize replay memory pool D
6: **while** $done = False$ **do**
7: the state transfer to the current state
8: choose a_j based on the $\varepsilon-$greedy method
9: **if** a_j exits in L **then**
10: the state transfer to the previous state
11: choose a_j based on the $\varepsilon-$greedy method
12: **else**
13: calculate the reward r_j according to Eq.19
14: store experience (s_j, a_j, r_j, s_{j+1}) in D and select mini-batch samples
15: Update the evaluation network $Q(\theta)$ with SGD
16: Copy the parameters of the evaluation network $Q(\theta)$ to the target network $Q(\theta^{-})$
17: **end if**
18: **if** $a = egressclassifier$ **then**
19: done = True
20: **end if**
21: **end while**
22: **end for**
23: **return** Output

Algorithm 3 CNNQ-based detection process

1: Current state $s_{t'}$
2: Call for the trained model
3: **for** t′ = 1, 2, … , t **do**
4: Input $s_{t'}$ to the evaluation network
5: Output the optimal path
6: Loop
7: the policy delivery module can generate the high-level policy-based selected action and forward it to the control plane
8: the translator controller translates the high-level policy into low-level policy and delivers the security controller
9: Update flow rule based on the low-level policy in the security controller
10: Delivery of the reconfigure command to the data plane with the translator controller
11: Reconfigure the selected NSFs in the data plane
12: Excute flow rule in the data plane
13: Activate the mitigation stage
14: Update the next state $s_{t'+1}$ in the state collector
15: End Loop
16: **end for**

(line 8). The security controller will update the flow rule based on the path information of low-level policy (line 9). Based on the configuration information of the low-level policy, the translator controller sends a command to the data plane (line 10) and the selected NSFs are reconfigured (line 11). Afterward, the classifier forwards incoming traffic according to the flow rule. The selected NSFs also detect traffic in sequence (line 12). Then, activated by the mitigation module, the relevant traffic feature will be recorded in the blacklist. Traffic will be dropped when these incoming traffic features match the blacklist (line 13). If the selected NSFs provide detection results, the time t' will be moved to t'+1. The system enters the next system state $s_{t'+1}$, and the state collector will synchronously update the state (line 14). The data plane gives feedback results to the knowledge plane, which guides the control plane to update the flow table.

When facing unknown attack traffic, the agent selects the optimal NSF combination based on the features of malicious behavior Ψ and bandwidth utilization Y. This approach ensures that the system can effectively detect unknown attacks and keep latency acceptable. Therefore, the closed-loop SFCA architecture can adjust the detection path to improve malicious traffic detection capability and reduce latency. More importantly, the architecture has universality and can be applied to detection various types of attacks.

6. Experimental results

This section evaluates the detection and latency performance of the selected paths by the CNNQ method proposed in Section 5, which is divided into five parts: experimental environments (Section 6.1), evaluation metrics (Section 6.2), choose reward weights (Section 6.3), choose training parameters (Section 6.4), and performance evaluation (Section 6.4).

Section 6.1 describes the setup of the experimental environments, the topology structure and used datasets. Section 6.2 introduces the various metrics which are used to measure the model performance. Section 6.3 explains the process of choosing the weights of the reward function. Section 6.4 selects the important parameters of the CNNQ model, including discount factor, epsilon, and learning rate by comparing with existing methods. Finally, Section 6.4 compares the performance before and after applying the CNNQ model in the prototype system and provides fine-grained detection results of the selected detection modules by the CNNQ model.

6.1. Experiment environments

Fig. 5 illustrates the SFCSA topology. The prototype is run on virtual machines (i.e., Ubuntu 18.04 LTS) and is integrated with the SSC, DRL, and policy deliver modules on a knowledge host. The security controller uses OpenDayLight (ODL) and I2NSF as the translator controller. We use Docker containers to implement NSFs (see Section 3) and manage the security resources. We deploy ingress and egress classifiers and forwarders with OpenvSwitch (OVS) in the virtual machines. The ingress classifier uses Python scripts to perform feature extraction, with the computation of feature entropy taking approximately 0.8 s. This study uses virtual machines as attack and benign hosts. We deploy a variety of scripts and tools to simulate 19 attacks in the attack hosts and generate 5G benign communication traffic with scripts in the benign host [34].

The closed-loop security architecture SFCSA aims to feedback and improve the total path malicious traffic detection capability based on traffic statistic reports. This architecture can provide the corresponding detection path (e.g., green and purple lines in Fig. 5) and reduce unnecessary latency under different attack scenarios.

For the CNNQ algorithm, the size of the replay memory D is set to 2000. The batch size is set to 128 [43]. The number of episodes is set to 20,000. We choose the weights in the reward function and select training parameters compared CNNQ with the Q-learning [45], DQN [46], and QMIND [47] approaches. Then, we plot the experimental results of total path malicious traffic detection capability and end-to-end latency for the proposed framework. Q-learning and DQN are basic reinforcement learning algorithms, and this work states that Q-learning, DQN, and CNNQ algorithms have the same state, action, and reward. For the QMIND algorithm, the states are precision, recall, F1-score, accuracy, and false alarm rate. The action is one of the five detection-aware NSFs. The reward function is the sum of weight factors and evaluation criteria. Then, we select the best parameters according

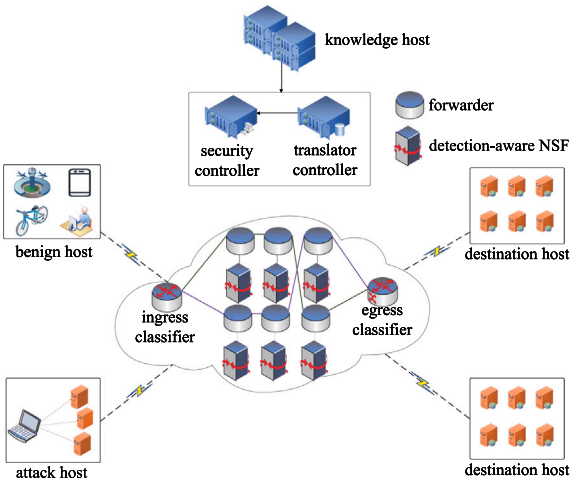


Fig. 5. SFCSA topology.

to the results. Finally, we test the performance of CNNQ with the selected reward and training parameters under different attack scenarios on its capability to provide a detection path to classify mixture traffic. Moreover, we also evaluate the selected detection modules separately to provide fine-grained results.

6.2. Evaluation metrics

The evaluation metrics include end-to-end latency, precision, recall, and F1-score, as described in Section 4. Additionally, we rely on the total path malicious traffic detection capability (TMTDC), which measures the total path malicious traffic detection capability, defined as follows:

$$\mathrm{TMTDC} = \frac{\sum_{c=0}^{C} \mathrm{MTDC}(NSF_c)}{C} + \frac{\mathrm{Average}\left[\mathrm{MTDC}(NSF_c)\right] \times \left[\frac{\sum_{c=0}^{C} \mathrm{MTDC}(NSF_c)}{\mathrm{Average}(\mathrm{MTDC}(NSF_c))} - C\right]}{C}, \tag{23}$$

where C is the number of selected detection-aware NSF in the path list, $\mathrm{MTDC}(NSF_c)$ denotes the detection capability of the cth detection-aware NSF, $\sum_{c=0}^{C} \mathrm{MTDC}(NSF_c)$ is the detection capabilities summation of selected detection-aware NSFs, and $\mathrm{Average}\left[\mathrm{MTDC}(NSF_c)\right]$ is the average detection capability of selected detection-aware NSFs.

6.3. Choose reward weights

The first attack scenario includes DRDoS and AppDDoS. We plot the results under different weight parameters in the first attack scenario. The reward weight parameter ρ_c and ρ_l [45] will result in different TMTDC and end-to-end latency results, as shown in Fig. 6(b). For the extreme case of $\rho_c = 0$ and $\rho_l = 1$, only latency is considered. The latency of the CNNQ approach (i.e., red line) has the smallest value compared with other settings. However, the TMTDC is not performing well in Fig. 6(a) because that reward function only considers latency, and the selected detection-aware NSFs of the CNNQ agent tends to be one of the five detection-aware NSFs. Nevertheless, the single NSF can only identify a part of the traffic attack, and thus the selected path will result in low latency and MTDC over a single NSF. For the extreme case of $\rho_c = 1$ and $\rho_l = 0$, where only MTDC is considered, the end-to-end latency of the CNNQ approach is the poorest, whereas the TMTDC performs well (Fig. 6(a)). This is because we ignore latency and only consider MTDC in the reward.

In the other case, where $\rho_c \neq 0$ and $\rho_l \neq 0$, increasing the weight parameter ρ_c (i.e., $\rho_c = 1$) can improve the proportion of MTDC in the reward, resulting in a slightly larger TMTDC value. However, as illustrated in Fig. 6(b), increasing the proportion of latency in the reward model leads to a larger latency especially for $\rho_l = 0.8$, compared with that of $\rho_l = 0.2$. This reward aims to provide a suitable path that guarantees MTDC and low latency. Thereby, we choose the weight parameter $\rho_c = 0.8$ and $\rho_l = 0.2$.

This section considers the second attack scenario, including DRDoS, AppDDoS, and Botnet. We test the performance of the different training parameters in the second attack scenario.

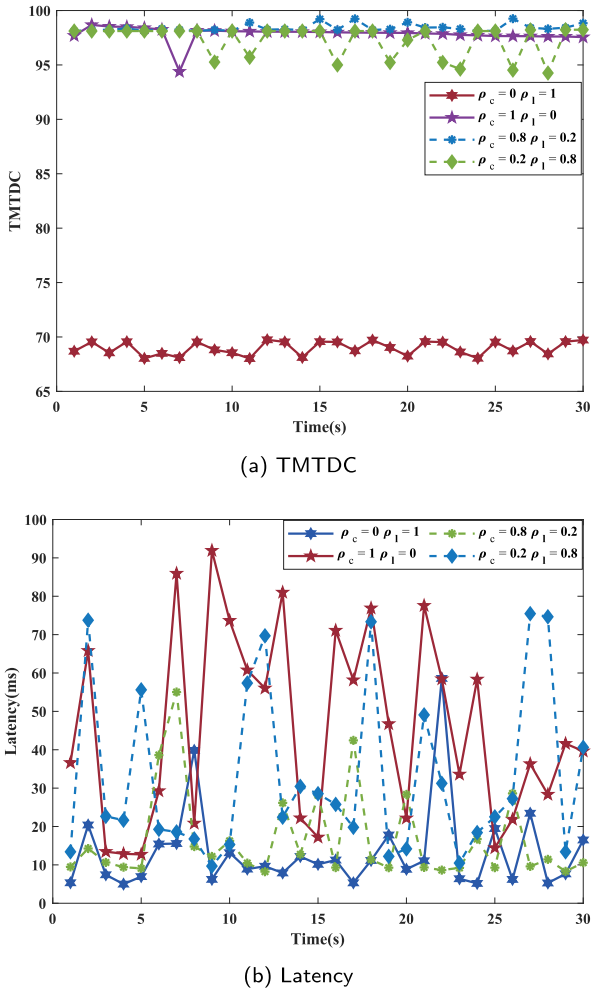


Fig. 6. Performance of the proposed CNNQ method under various ρ_c and ρ_l.

Table 4
Average latency with different discount factor parameters.

Time	Methods	Average latency		
		0.5	0.7	0.9
10 s	DQN	23.46	21.62	24.31
	CNNQ	20.38	18.77	11.48
	QMIND	25.21	12.84	11.04
	Q-learning	16.22	10.76	19.62
20 s	DQN	20.24	26.62	24.32
	CNNQ	16.68	18.62	16.65
	QMIND	12.74	16.85	13.62
	Q-learning	16.79	16.82	15.26
30 s	DQN	22.35	17.71	24.38
	CNNQ	17.82	18.49	14.59
	QMIND	11.84	14.94	10.59
	Q-learning	16.14	15.48	12.49

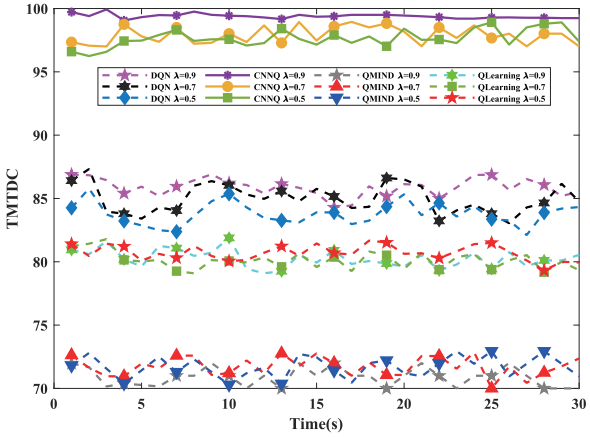


Fig. 7. TMTDC with different discount factor parameters.

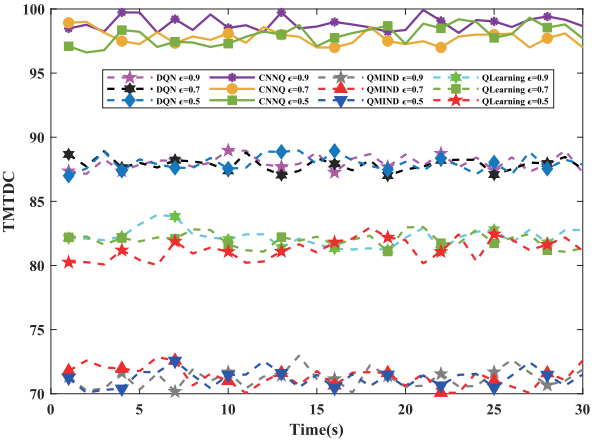


Fig. 8. TMTDC with different epsilon parameters.

(1) Choose the Discount Factor λ: The discount factor λ reflects the balance of the current reward and the cumulative future reward from the next state. Different discount factors impose a different reward impact for the current state and the estimated cumulative reward from the next state. As depicted in Fig. 7, the QMIND agent does not consider the MTDC and the end-to-end latency. The QMIND method has the lowest TMTDC, whereas the CNNQ method has the highest TMTDC under different discount factors. The experimental results infer that TMTDC reaches 99 for $\lambda = 0.9$. Table 4 reports the average latency of the CNNQ method compared to the other RL models. The average latency of CNNQ is lower than that of DQN, but the average latency of CNNQ is slightly higher than that of Q-learning and QMIND at most times. However, the increased latency is acceptable because the CNNQ method has the highest TMTDC. For the CNNQ model, the average latency for $\lambda = 0.9$ is the lowest at different times. Based on the previously mentioned analysis, we choose the CNNQ model, and the discount factor λ is 0.9.

(2) Choose the Epsilon ε: Section 5 explained the $\varepsilon-$greedy method. The epsilon value ε reflects whether the model can find the optimal global solution. Fig. 8 reveals that the TMTDC of CNNQ achieves the highest value (i.e., 99) for epsilon $\varepsilon = 0.9$, whereas the corresponding value of QMIND is less than 75 under various epsilons. The ranking of the corresponding value of the DQN and Q-learning methods is middle. Table 5 presents the corresponding average latency of different epsilon parameters in different models. Owing to dynamic change in the network, the QMIND model has a narrow range of latency fluctuation compared with other methods at different times. The average latency

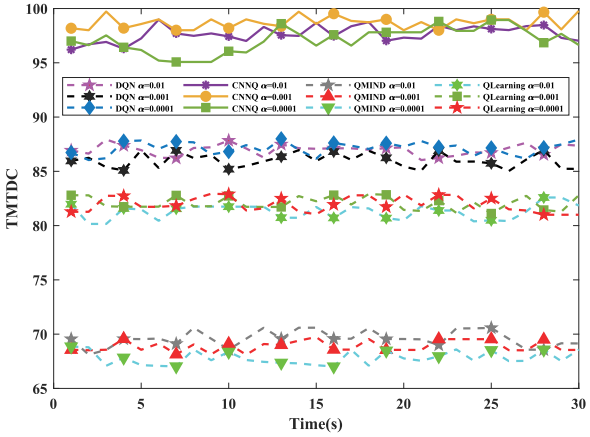


Fig. 9. TMTDC with different learning rate parameters.

of Q-learning and DQN is slightly slower than that of CNNQ in some time. However, the chosen model aims to balance MTDC and latency. Thereby, we choose the CNNQ method. For the CNNQ model, the average latency of $\varepsilon = 0.9$ is the lowest at different times. Finally, we choose $\varepsilon = 0.9$.

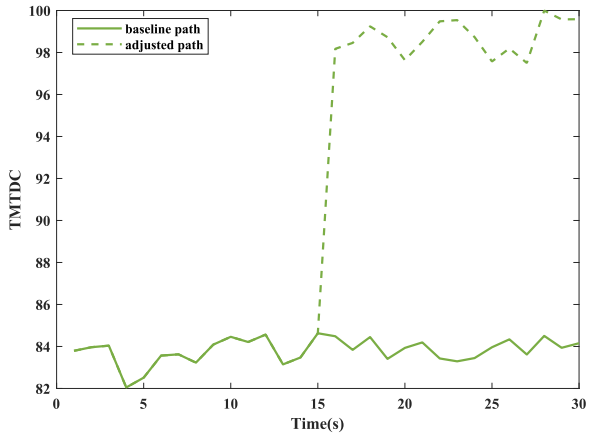
(3) Choose the Learning Rate α: The learning rate α controls the process, so the model learns new state information. If the learning rate is too low, it is difficult for the agent to learn the new reward information, resulting in slow convergence. On the opposite, if the learning rate is too large, the agent always favors new information over the existing experience, causing an unstable loss in the learning process. Fig. 9 demonstrates that our proposed CNNQ approach has the highest TMTDC when the learning rate is 0.001. The DQN and QMIND cannot fully use the state information whose results are next to that of the CNNQ method.

Table 6 reveals that the corresponding latency of QMIND has a small value under a learning rate of 0.01, 0.001, 0.0001, whereas one NSF cannot detect all DDoS attacks. The corresponding average latency of CNNQ is higher than that of Q-learning and DQN. The maximum difference in latency between CNNQ and other methods is 12.33 ms in 30 s, in which a minor increase in latency can affect normal service, e.g. real-time communication. However, the difference between the corresponding latency of CNNQ and the other models is acceptable in most cases. Considering both latency and MTDC, we choose the learning rate 0.001 of the CNNQ method.

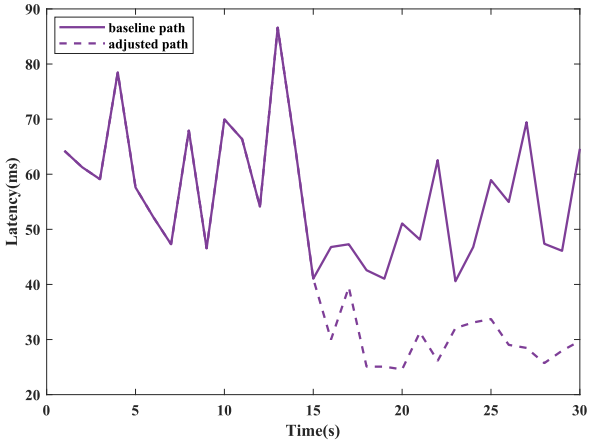
Based on the previously mentioned experimental results analysis, we conclude that the detection performance when combining detection modules is better than relying on a single detection module. Besides, we analyze the total path malicious traffic detection capability and latency under different parameters of different models. Finally, we train the CNNQ model with the best parameters. After measuring 20,000 episodes, the training process takes 133.40 s.

6.4. Performance evaluation

(1) The performance before and after enabling CNNQ: To examine the effectiveness and intelligence of CNNQ, we train the CNNQ model offline with the parameters selected in the above analysis and then deploy it in the DRL module. First, a security path (i.e., NADM, AADM, DADM, and LADM) is preset as a baseline path, and then we launch the third attack scenario (i.e., LDDoS, NetDDoS, and DRDoS) for the 30 s. Fig. 10(a) shows the TMTDC before and after enabling CNNQ. Under mixed attack traffic, the corresponding TMTDC of the baseline path is less than 88 (i.e., solid green line). After 15 s, the trained model selects the adjusted path (i.e., DADM, NADM, and LADM) according to the state. The adjusted path will deliver to the translator controller, which



(a) TMTDC



(b) Latency

Fig. 10. Performance before and after enabling CNNQ.

can translate high-level policy into low-level policy and send it to the security controller. Then, the flow entry is modified and delivered to the data plane, forming a service function chain to complete detection. The TMTDC of the adjusted path (i.e., green dotted line) is up to 99, increased by 18 compared to the baseline path.

Fig. 10(b) depicts the latency before and after enabling CNNQ. When mixed illegitimate traffic enters the network, the latency of the baseline path fluctuates between 40s and 90s (i.e., solid purple line). After adjusting the path, the latency range fluctuates between 20s and 40s (i.e., purple dotted line). Compared with the baseline path, the corresponding latency of the path selected by the CNNQ model slightly decreases while the path greatly increases the TMTDC. Therefore, CNNQ can detect the attack with high detection capability and decrease latency.

(2) Performance evaluation enabling CNNQ: We deploy detection modules as NSFs to detect mixed attack traffic. The detail of each detection module is presented in Section 3. Fig. 11 illustrates the evaluation topology before and after enabling CNNQ. The baseline path is the source host, ingress classifier, NADM, AADM, DADM, BTDM, egress classifier, and destination host. When the fourth attack scenario (i.e., LDDoS, Botnet, AppDDoS and DRDoS) is launched, we first activate SSC to gather traffic statistic reports from the data plane. Then, traffic statistic reports are input into the established CNNQ model in the DRL module. Eventually, the model outputs the selected detection-aware NSFs. The selected detection-aware NSFs are DADM, BTDM, AADM, and LADM.

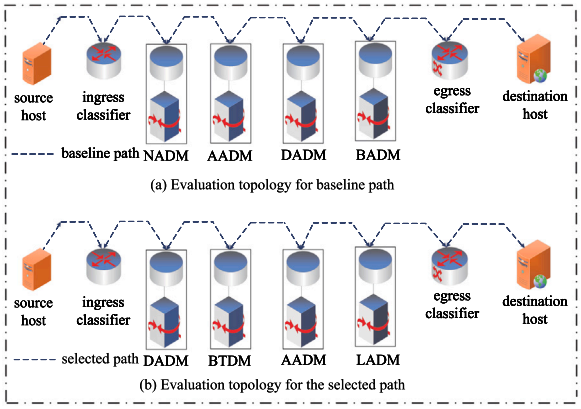


Fig. 11. Evaluation topology before and after enabling CNNQ.

Table 5
Average latency with different epsilon parameters.

Time	Methods	Average latency		
		0.5	0.7	0.9
10 s	DQN	22.41	36.52	26.69
	CNNQ	25.32	29.64	21.54
	QMIND	15.11	12.87	11.87
	Q-learning	18.37	36.58	27.41
20 s	DQN	28.24	22.78	34.82
	CNNQ	28.48	29.34	29.24
	QMIND	15.16	12.61	11.57
	Q-learning	20.98	25.25	22.23
30 s	DQN	26.52	22.81	22.21
	CNNQ	29.28	27.69	25.42
	QMIND	14.6	13.93	10.86
	Q-learning	15.42	17.42	21.32

Table 6
Average latency with different learning rate parameters .

Time	Methods	Average latency		
		0.01	0.001	0.0001
10 s	DQN	13.48	12.48	20.49
	CNNQ	25.64	20.83	21.37
	QMIND	12.71	16.05	11.17
	Q-learning	14.88	19.01	25.73
20 s	DQN	15.32	11.20	14.25
	CNNQ	14.42	15.57	26.43
	QMIND	13.06	15.27	13.92
	Q-learning	16.77	17.19	19.49
30 s	DQN	17.83	18.87	20.49
	CNNQ	23.51	22.4	27.35
	QMIND	12.17	10.07	11.8
	Q-learning	22.62	29.42	38.79

The path consists of an ingress classifier, detection modules, and an egress classifier described as a high-level policy. The policy deliver module delivers high-level security policy to the translator controller, which translates high-level security policy into the corresponding low-level security policy. Then, the translator controller sends the path rule to the security controller and a reconfiguration policy to the data plane. In the data plane, the security controller parses the path rule of the low-level security policy forming flow tables and delivering flow tables to classifiers and forwarders to implement the detection path. Meanwhile, the classifiers and forwarders receive and execute the reconfiguration policy. Finally, LDDoS, Botnet, AppDDoS and DRDoS traffic are detected with a selected detection path.

The detection results of each detection module are reported in Table 7, demonstrating that each detection module achieves fine-grained classification. For DADM,each attack's precision, recall, and F1-score

Table 7
Detection results.

Module	Attacks type		Precision	Recall	F1-score
DADM	DRDoS	Chargen	0.99	0.99	0.99
		TFTP	0.99	0.99	0.99
		SSDP	0.99	0.99	0.99
		NTP	0.99	0.99	0.99
BTDM	Botnet	Ares	0.99	0.98	0.99
		Byob	0.32	0.66	0.43
		Mirai	0.98	0.93	0.95
		Zeus	0.99	0.99	0.99
		IRC	0.99	0.99	0.99
AADM	App DDoS	CC	0.95	0.85	0.92
		HTTP-Flood	0.68	0.7	0.69
		HTTP-Get	0.99	0.99	0.99
		HTTP-Post	0.99	0.5	0.67
LADM	LDDoS	slow read	0.77	0.98	0.86
		slow body	0.96	0.85	0.9
		slow headers	0.99	0.77	0.87

are almost 0.99. For BTDM, each attack's precision, recall, and F1-score are over 0.90, except for the Byob attack. For AADM, all metrics on Challenge Collapsar (CC) and HTTP-Get attacks exceed 0.85 and 0.99. However, the HTTP-Post recall only is 0.5, and HTTP-Flood precision is only 0.68. This is because HTTP-Flood and HTTP-Post attack tools send many request packets to the destination hosts. Thus, the attacks have similar malicious behavior, making it challenging to identify both attack types accurately. For LADM, the slow read precision and slow headers recall are 0.77, whereas other attack metrics exceed 0.85.

Overall, most attack types are correctly classified into only a few attacks may be misjudged as other attack types, resulting in low precision and recall. Therefore, we further adjust and optimize the parameters to enhance the fine-grained classification performance of the detection modules.

7. Conclusion

Due to facing diverse attack types, we combine detection modules and form detection paths and propose a closed-loop security framework that integrates DRL, I2NSF, and SDN/NFV technology and applies them to detect DDoS attacks. Specifically, first, we formulate the path selection problem as MDP. Then, the proposed integrated CNNQ algorithm ensures the malicious traffic detection capability and low end-to-end latency. Furthermore, we define a metric for evaluating the total path malicious traffic detection capability. Finally, we implement and evaluate the developed framework over our developed prototype system.

Compared with the existing RL algorithms, the experimental results demonstrate that the detection performance when combining multiple detection modules outperform the single detection module. Besides, our proposed algorithm can find an appropriate path considering the end-to-end latency and malicious traffic detection capability. It is worth noting that in this article, we only construct the detection module for known attacks. In the future, we will verify the performance of detection zero-day attacks with the SFCSA architecture. Besides, we will explore the multiagent reinforcement learning method that cooperates with detection results among multi-domain to enhance cyberspace security capability.

CRediT authorship contribution statement

Man Li: Conceptualization, Methodology, Formal analysis, Writing – original draft, Visualization. **Shuangxing Deng:** Investigation, Software, Data curation. **Huachun Zhou:** Funding acquisition, writing – review & editing. **Yajuan Qin:** Supervision.

Declaration of competing interest

The authors declared that they have no conflicts of interest to this work.

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Data availability

The data that has been used is confidential.

References

[1] N.M. Yungaicela-Naula, C. Vargas-Rosales, J.A. Perez-Diaz, SDN-based architecture for transport and application layer DDoS attack detection by using machine and deep learning, IEEE Access 9 (2021) 108495–108512.

[2] X.C. Wu, K.Y. Hou, X. Leng, X. Li, Y.B. Yu, B. Wu, Y. Chen, State of the art and research challenges in the security technologies of network function virtualization, IEEE Internet Comput. 24 (1) (2020) 25–35.

[3] M. Imran, M.H. Durad, F.A. Khan, H. Abbas, DAISY: A detection and mitigation system against denial-of-service attacks in software-defined networks, IEEE Syst. J. 14 (2) (2020) 1933–1944.

[4] NSFOCUS, Ddos threats report in 2021, 2021, http://blog.nsfocus.net.

[5] G.C. Amaizu, C.I. Nwakanma, S. Bhardwaj, J.M. Lee, D.S. Kim, Composite and efficient DDoS attack detection framework for B5G networks, Comput. Netw. 188 (2021) 107871.

[6] J. Halpern, Ed. Ericsson, Ed C. Pignataro, RFC7665: Service function chaining, 2015, https://www.rfc-editor.org/rfc/rfc7665.txt.

[7] D. Lopez, Telefonica I+D, E. Lopez, Curveball Networks, L. Dunbar, J. Strassner, RFC 8329: Framework for interface to network security functions draft-ietf-i2nsf-framework-08, 2017, https://www.ietf.org/archive/id/draft-ietf-i2nsf-framework-08.txt.

[8] Keshav Sood, Mohammad Reza Nosouhi, Dinh Duc Nha Nguyen, Frank Jiang, Morshed Chowdhury, Robin Doss, Intrusion detection scheme with dimensionality reduction in next generation networks, IEEE Trans. Inf. Foren. Sec. 18 (2023) 965–979.

[9] Kumar P., Tripathi M., Nehra A., SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN, IEEE Trans. Netw. 15 (2018) 1545–1559.

[10] R. Sanjeetha, K.N.A. Shastry, H.R. Chetan, A. Kanavalli, Mitigating HTTP GET FLOOD DDoS attack using an SDN controller, in: Proc. IEEE Int. Conf. Recent Trends Electronics, Inf., Commun. Technol., 2020, pp. 6–10.

[11] P. Maity, S. Saxena, S. Srivastava, K.S. Sahoo, A.K. Pradhan, N. Kumar, An effective probabilistic technique for DDoS detection in OpenFlow controller, IEEE Syst. J. 16 (2022) 1345–1354.

[12] S. M. Mousavi, M. St-Hilaire, Early detection of DDoS attacks against SDN controllers, in: Proc. IEEE Int. Conf. Comput., Netw. Commun., 2015, pp. 77–81.

[13] S.S. Kshira, P. Deepak, T. Mayank, J.P.C. R. Joel, S. Bibhudatta, D. Ratnakar, An early detection of low rate DDoS attack to SDN based data center networks using information distance metrics, Future Gener. Comput. Syst. 89 (2018) 685–697.

[14] Xu shaoyi, Zheng shanshan, Multi-agent reinforcement learning based resource allocation for M2M communication, J. Beijing Jiaotong Univ. 42 (2018) 1–9.

[15] D. L. Gu, W. Zeng, S. Li, A. Guo, Y. Zomaya, H. Jin, Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2020) 279–291.

[16] Rami J. Alzahrani, A.S. Alzahrani, Survey of traffic classification solution in IoT networks, Int. J. Comput. Appl. 183 (2021) 37–45.

[17] T.M. Nam, P.H. Phong, T.D. Khoa, T.T. Huong, P.N. Nam, N.H. Thanh, L.X. Thang, P.A. Tuan, V.D. Loi, Self-organizing map-based approaches in DDoS flooding detection using sdn, in: Proc. IEEE Int. Conf. Inf. Netw., 2018, pp. 249–254.

[18] L. Tan, Y. Pan, J. Wu, J. Zhou, H. Jiang, Y. Deng, A new framework for DDOS attack detection and defense in SDN environment, IEEE Access 8 (2020) 161908–161919.

[19] J. Li, Z. Zhao, R. Li, H. Zhang, AI-based two-stage intrusion detection for software defined IoT networks, IEEE Internet Things J. 6 (2019) 2093–2102.

[20] N. Ahuja, G. Singal, D. Mukhopadhyay, N. Kumar, Automated DDoS attack detection in software defined networking, J. Netw. Comput. Appl. 187 (2021) 103108.

[21] T. AlMasri, M.A. Snober, Q.A. Al Haija, IDPS-SDN-ML: An intrusion detection and prevention system using software-defined networks and machine learning, in: Proc. 1st Int. Conf. Smart Technol., Applied Inform. Eng., 2022, pp. 133–137.

[22] N.M.Y.N., Cesar Vargas-Rosales, Jesús Arturo Pérez-Díaz, Diego Fernando Carrera, A flexible SDN-based framework for slow-rate DDoS attack mitigation by using deep reinforcement learning, J. Netw. Comput. Appl. 205 (2022) 103444.

[23] E. Tahoun Akbari, M.A. Salahuddin, N. Limam, R. Boutaba, Atmos: Autonomous threat mitigation in SDN using reinforcement learning, in: Proc. IEEE/IFTP Netw. Oper. Manage. Symp., 2020, pp. 1–9.

[24] Y. Liu, M. Dong, K. Ota, J. Li, J. Wu, Deep reinforcement learning based smart mitigation of DDoS flooding in software-defined networks, in: Proc. 23th Int. Conf. Workshop Comput. Aided Model. Des. Commun. Links Netw., 2018, pp. 1–6.
[25] M. Zolotukhin, S. Kumar, Reinforcement learning for attack mitigation in SDN-enabled networks, in: Proc. 6th Int. Conf. Netw. Soft, 2020, pp. 282–286.
[26] K. Malialis, D. Kudenko, Distributed response to network intrusions using multiagent reinforcement learning, Eng. Appl. Artif. Intell. 41 (2015) 270–284.
[27] T.T. Nguyen, V.J. Reddi, Deep reinforcement learning for cyber security, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–17.
[28] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J.J. Ramos-Munoz, J.M. Lopez-Soler, A survey on 5G usage scenarios and traffic models, IEEE Commun. Surv. Tutor. 22 (2020) 905–929.
[29] The tcpdump group, TCPDUMP, 2013, https://github.com/the-tcpdump-group/tcpdump.
[30] A.H. Lashkari, CicFlowMeter, 2018, https://github.com/ahlashkari/CICFlowMeter.
[31] Li Lijuan, Li Man, Bi Hongjun, Zhou Huachun, Multi-type low-rate DDoS attack detection method based on hybrid deep learning, Chin. J. Netw. Inf. Secur. 8 (2022) 73–85.
[32] Li Yingzhi, Li Man, Dong Ping, Zhou Huachun, Multi-type application layer DDoS attack detection method based on integrated learning, J. Comput. Appl. 42 (2022) 1–9.
[33] Shen Qi, Tu Zhe, Li Kun, Qin Yajuan, Zhou Huachun, Online botnet detection method based on ensemble learning, Appl. Res. Comput. 39 (2022) 1–9.
[34] M. Li, H. Zhou, Y. Qin, Two-stage intelligent model for detecting malicious DDoS behavior, Sensors 2022 (2022) 2532.
[35] Tianqi Yang, Weilin Wang, Ying Liu, Huachun Zhou, Multi-class DRDoS attack detection method based on feature selection, Res. Briefs Inf. Commun. Technol. Evol. 7 (2021) 1–15.
[36] M.E. Ahmed, H. Kim, M. Park, Mitigating DNS query-based DDoS attacks with machine learning on software-defined networking, in: Proc. Int. Mil. Commun. Conf., 2017, pp. 11–16.
[37] S. Almutairi, S. Mahfoudh, S. Almutairi, J.S. Alowibdi, Hybrid botnet detection based on host and network analysis, J. Comput. Netw. Commun. 2020 (2020) 1–16.
[38] M. Hajimaghsoodi, R. Jalili, RAD: A statistical mechanism based on behavioral analysis for DDoS attack countermeasure, IEEE Trans. Inf. Foren. Sec. 17 (2022) 2732–2745.
[39] M.E. Ahmed, S. Ullah, H. Kim, Statistical application fingerprinting for DDoS attack mitigation, IEEE Trans. Inf. Foren. Sec. 14 (2019) 1471–1484.
[40] R. Wang, Z. Jia, L. Ju, An entropy-based distributed DDoS detection mechanism in software-defined networking, in: Proc. IEEE Int. Conf. Trust, Secur., Privary Comput. Commun., 2015, pp. 310–317.
[41] W. Guo, J. Xu, Y. Pei, L. Yin, C. Jiang, N. Ge, A distributed collaborative entrance defense framework against ddos attacks on satellite internet, IEEE Internet Things J. 9 (2022) 15497–15510.
[42] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley-Blackwell, New Jersey, NJ, 2006.
[43] J. Chen, J. Chen, H. Zhang, DRL-QOR: Deep reinforcement learning-based QoS/QoE-aware adaptive online orchestration in NFV-enabled networks, IEEE Trans. Netw. Serv. Manag. 18 (2021) 1758–1774.
[44] Peilun Wu, Hui Guo, Lunet: A deep neural network for network intrusion detection, in: IEEE Symposium Series on Computational Intelligence, 2019, pp. 617–624.
[45] J. Chen, X. Cheng, J. Chen, H. Zhang, A lightweight SFC embedding framework in SDN/NFV-Enabled wireless network based on reinforcement learning, IEEE Syst. J. 16 (2022) 3817–3828.
[46] J. Wang, Y. Liu, W. Zhang, X. Yan, N. Zhou, Z. Jiang, Relfa: Resist link flooding attacks via renyi entropy and deep reinforcement learning in SDN-IoT, China Commun. 19 (2022) 157–171.
[47] T.V. Phan, T.M.R. Gias, S.T. Islam, T.T. Huong, N.H. Thanh, T. Bauschert, Q-MIND: Defeating stealthy DoS attacks in SDN with a machine-learning based defense framework, in: Proc. 6th Int. Global Commun. Conf., 2019, pp. 1–6.

Man Li is currently a Ph.D. candidate at the National Engineering Laboratory for Next Generation Internet Interconnection Devices, Beijing Jiaotong University, Beijing, China. Her research interests includes network security and network function virtualization.

Shuangxing Deng is currently studying for the M.S. degree in information and telecommunications engineering. His research interest is network security.

Huachun Zhou received the B.S. degree from the People's Police Officer University of China in 1986. He received the M.S. in telecommunication automation and Ph.D. degrees in telecommunications and information system from Beijing Jiaotong University in 1989 and 2008, respectively. Now, he is a professor in National Engineering Lab for Next Generation Internet Interconnection Devices at BJTU. His main research interests are in the area of mobile and secure computing, network management and satellite network.

Yajuan Qin received her B.S. and M.S. degrees in Radio Technology from the University of Electronic Science and Technology of China in 1985 and 1988, respectively,and Ph.D. degree in communication engineering from Beijing University of Posts and Telecommunications in 2003. She was a research associate in 2002 at CRL of Japan. She joined Beijing Jiaotong University in 2004, where she is now a professor. Her research interests are in the areas of computer networks and wireless communications.