DDoS Mitigation while Preserving QoS: A Deep Reinforcement Learning-Based Approach

Shurok Khozam, Gregory Blanc, Sébastien Tixeuil, Eric Totel

DDoS Mitigation while Preserving QoS: A DDQN-Based Approach

1st Shurok Khozam
SAMOVAR
Télécom SudParis
Palaiseau, France
shurok.khozam@telecom-sudparis.eu

2nd Gregory Blanc
SAMOVAR
Télécom SudParis
Palaiseau, France
gregory.blanc@telecom-sudparis.eu

3rd Sébastien Tixeuil
LIP6
Sorbonne Université
Paris, France
sebastien.tixeuil@lip6.fr

4th Eric Totel
SAMOVAR
Télécom SudParis
Palaiseau, France
eric.totel@telecom-sudparis.eu

***Abstract*—The deployment of 5G networks has significantly improved connectivity, providing remarkable speed and capacity. These networks rely on Software-Defined Networking (SDN) to enhance control and flexibility. However, this advancement poses critical challenges including expanded attack surface due to network virtualization and the risk of unauthorized access to critical infrastructure. Since traditional cybersecurity methods are inadequate in addressing the dynamic nature of modern cyber attacks, employing artificial intelligence (AI), and deep reinforcement learning (DRL) in particular, was investigated to enhance 5G networks security. This interest arises from the ability of these techniques to dynamically respond and adapt their defense strategies according to encountered situations and real-time threats. Our proposed mitigation system uses a DRL framework, enabling an intelligent agent to dynamically adjust its defense strategies against a range of DDoS attacks, exploiting ICMP, TCP SYN, and UDP, within an SDN environment designed to mirror real-life user behaviors. This approach aims to maintain the network's performance while concurrently mitigating the impact of the real-time attacks, by providing adaptive and automated countermeasures according to the monitored network's situation.**

***Index Terms*—Reinforcement Learning, Distributed Denial of Service, Quality of Service, Software-Defined Networking**

I. Introduction

The advent of 5G networks has ushered in a new era of connectivity, promising unprecedented speed, reliability, and capacity [1], [2]. These networks rely on SDN as the backbone architecture, serving as the infrastructure that enhances both control and flexibility [1]. However, with this technological leap comes the inevitable challenge of securing these networks against evolving cyber threats. Traditional cybersecurity approaches, such as signature-, blockchain- and rule-based systems, are unable to address the evolved techniques of modern cyber threats as they rely on static and rigid rules, which can result in their inability to effectively deal with unseen attack patterns. Additionally, they may introduce latencies that are unacceptable for real-time applications. In response to these challenges, there is a growing interest in leveraging AI and DRL techniques to enhance the security of 5G networks [3] as they can autonomously discover effective defense strategies in dynamic and uncertain environments. Additionally, RL can iteratively improve its performance over time by adjusting its strategies based on the environment's feedback, enhancing the resilience of the 5G networks against emerging threats. To provide a comprehensive understanding, we survey relevant literature on the role of ML and DRL in enhancing the security of 5G networks including various perspectives and methodologies. In particular, we focus on defenses against distributed denial-of-service (DDoS) attacks given their severe threat to 5G infrastructures. These attacks can control a large number of compromised devices to amplify their malicious impact, making it harder for defenders to simply drop or disconnect the attacker hosts [4], [5]. Furthermore, in recent years, the rise of IoT devices with low-cost, high-bandwidth connections has exacerbated these attacks by providing a vast pool of vulnerable devices for exploitation.

In this paper, we propose a DRL-based approach that employs a Double Deep Q-Network (DDQN) algorithm to dynamically adapt its mitigation strategies in response to DDoS attacks in SDN with the objective of mitigating their impact on the network and maintaining its performance for honest users.

II. Related work

Given the importance of 5G, many authors argued that the role of machine learning (ML), beside DRL applications, can be used to create complex and effective defense systems instead of traditional cybersecurity approaches such as signature-based, blockchain-based, and rule-based systems, which are not sufficient to address the rapidly evolving nature of recent cyber attacks [6]. Mihoub et al. have employed advanced ML techniques for DoS/DDoS detection and mitigation [7]. Their approach adopts a fine-grained perspective, identifying different subcategories within broader classifications of DoS and DDoS attacks, achieving promising accuracy of 99.81% with a Random Forest classifier empowered by the "Looking-Back" concept. The mitigation component involves the application of rate-limiting measures on specific traffic or rejection of packets originating from specific IP addresses. However, the evaluation of the proposed system is conducted on the Bot-IoT dataset. While this dataset provides a basis for testing, it may not fully represent the diverse range of real-world IoT scenarios and attack variations [8].

Furthermore, Dake et al. introduced an innovative multi-agent RL framework tailored for SDNs [9]. This framework

employs Multi-Agent Deep Deterministic Policy Gradient algorithm, focusing on enhancing multipath routing efficiency and robust detection and prevention of malicious DDoS traffic. Another DRL approach, DeepAir was proposed for adaptive intrusion response in SDN [10], where a DDQN algorithm was developed to train the agent on a dataset of network traffic to learn an optimal intrusion response policy in SDN networks. The authors considered two types of DoS attacks, TCP SYN flood and Link layer flood. By learning this dynamic policy, their approach achieved a significant drop in attack packets and a remarkable 70% reduction in quality of service (QoS) violations. However, the evaluation of the proposed approach is limited to specific scenarios and is not assessed in real-world. On a different note, Akbari et al. presented a framework named ATMoS (Autonomous Threat Mitigation using RL in SDN) to simplify the process of developing RL applications in the context of network security management with SDN [11]. In the proposed framework, a Neural Fitted Q-learning agent is employed to mitigate an Advanced Persistent Threat (APT) and DDoS attacks. The convergence results demonstrate the efficacy of the proposed approach for active threat mitigation. However, the number of hosts and their ordering must remain fixed during training.

III. Attacker Model

A. Assumptions

We are considering an SDN network with the following assumptions:

- Hosts exchange various volumes of traffic.
- The attacker's sending rate exceeds that of legitimate hosts.
- The attacker takes control of the host associated with the network connection possessing the highest bandwidth.
- Attacks are applied periodically for specific durations, which are deemed sufficient to carry out a successful attack.

B. Attack Scenarios

In this article, we consider an SDN network where different users are communicating with a server through controlled switches in a provider network, accomodating varied traffic volumes across various ingress points. We suppose that the network under study is composed of n ingress points, m controlled switches with the condition that $n \geq m$, each switch is connected to all other controlled switches, and finally a server represented by h_s accessible through switch s_0. Such a network composition is depicted in Fig. 1 where the small colored boxes, denoted by switches' numbers $10x$, indicate that all switches are interconnected with each others.

The main focus of our problem revolves around an attacker attempting to compromise a server's availability by initiating various DDoS attacks including TCP SYN, UDP and ICMP floods. This adversarial activity is triggered from a subnet interfacing with the protected network, producing a mixture of traffic with varying proportions of legitimate and malicious flows. The attacker in our approach is smart enough to compromise the host with the highest bandwidth as a starting point for its malicious activities towards the server, triggering its attacks for a specific duration, using a substantial quantity of high-priority threads. Each of them opens its own connection to the server and executes the predefined attack in autonomy in order to achieve the highest impact on the server. The duration of each attack thread varies highly and is randomly defined.

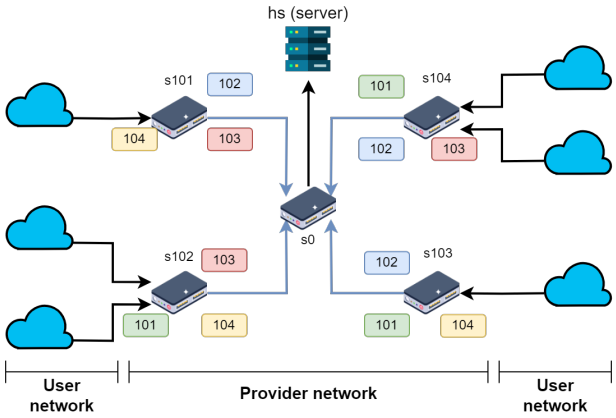

Fig. 1. Network Architecture.

IV. Deep Reinforcement Learning

Reinforcement Learning (RL) is a dynamic paradigm within machine learning where agents learn to make sequential decisions by interacting with an environment to maximize cumulative rewards. The fundamental essence of RL revolves around the core concept known as Markov Decision Processes (MDP), a mathematical framework that models decision-making scenarios with states, actions, and rewards. Agents adapt their strategies through a process of exploration and feedback to learn optimal policies [12]. In reinforcement learning, an agent learns to interact with an environment by selecting actions to maximize cumulative rewards. The agent observes the current state of the environment, selects an action according to its policy, and receives feedback in the form of rewards. The key components include *the agent*, responsible for decision-making based on a policy π; *the environment*, which provides states s and rewards r in response to actions; and *the reward signal*, denoted by r_t, representing the immediate reward at time t for an action a_t. The agent's goal is to learn an optimal policy that maximizes the expected cumulative reward over time, as formulated by the expression $\max_\pi \sum_{t=0}^{\infty} \mathbb{E}[r_t]$ [13]. In this approach we use one variation of RL algorithms, which is DDQN algorithm explained in the following subsection.

A. Double Deep Q-Network

Double Deep Q-Network (DDQN) is an improvement to the Deep Q-Learning (DQN) algorithm aimed at mitigating the overestimation issue inherent in DQN. In DQN, a single neural network is employed to compute the estimated values of possible actions within a given state, denoted as action values, and the desired Q-values, represented by target values. However, this mechanism can lead to overestimation of action

values, particularly in scenarios with noisy or high variance environments [14].

DDQN addresses this problem by utilizing two separate neural networks: one for action selection and one for action evaluation. The online network, denoted as Q', is responsible for selecting actions, while the target network, denoted as Q, is used for action evaluation. By decoupling the action selection and evaluation, DDQN reduces the likelihood of overestimation [15]. The DDQN algorithm updates the action-value function using the following formula:

$$Q^*(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, \arg\max_{a'} Q'(s_t, a_t))$$

Here, $Q^*(s_t, a_t)$ represents the estimated optimal action-value function for the current state s_t and action a_t. The term $\arg\max_{a'} Q'(s_t, a_t)$ selects the action with the highest value according to the online network Q'. This selected action's value is then evaluated using the target network Q to reduce overestimation.

V. Adaptive RL for DDoS Mitigation

A. Methodology

In this section, we present our methodology for DDoS attack mitigation. We focus on an SDN network with an attacker attempting to compromise a server by launching a diverse set of DDoS attacks, including TCP SYN, UDP, and ICMP floods, with varied traffic intensity. This malicious traffic poses a threat for benign users by disrupting their services. To address this problem, we employ a real-time monitoring module to continuously observe the network's situation during the attack.

This module collects network metrics' information and transmits them to the RL module through the controller's southbound interfaces. The RL module, selects suitable countermeasures based on its trained neural network to enhance the network's performance. These chosen countermeasures are sent through the controller's northbound interfaces. The controller interprets the RL algorithm's (i.e., the agent's) decisions and transforms them into flow rules applied in the data plane. These measures include the manipulation of traffic volumes to reduce the volume of malicious traffic sent by attackers, or conversely, to maximize traffic for legitimate users on different hosts. Other measures include traffic redirection across multiple switches, which is explained in Section V-C.

The agent taking these measures is trained using the DDQN algorithm since it is able to deal with continuous state spaces, to learn and adapt to dynamic environments by continuously refining its strategies based on feedback from network performance and attack mechanism's severity.

B. RL Model

During training, the RL model learns by interacting with its environment's state to optimize actions based on received rewards. In our approach, the state of the RL agent encompasses a diverse set of traffic metrics. These metrics are selected to offer a comprehensive, holistic overview of the network environment, enabling the agent to make informed decisions to optimize performance and enhance the overall efficiency. The state presented in Table I is defined for each ingress point and extracted at the controlled switches' interfaces based on information computed from captured flows that are deemed to be originating from mostly honest networks, considering that the distinction by legitimate and malicious flows is provided by a third party intrusion detection system.

State metric	**Extraction method**
Total packets in the forward direction (ingress point to server)	Tshark & CICFlowMeter
Total packets in the backward direction (server to ingress point)	Tshark & CICFlowMeter
Total size of packets in the forward direction	Tshark & CICFlowMeter
Total size of packets in the backward direction	Tshark & CICFlowMeter
Number of received bytes	Controller through OVS commands
Number of transmitted bytes	Controller through OVS commands
Bandwidth	Controller through OVS commands
Number of lost packets	Manually computed
Number of delivered packets	Manually computed
Number of transmitted bytes per second	Manually computed
Number of transmitted packets per second	Manually computed
Controlled switches of the provider network along ingress point's path to the server	Manually computed
Bandwidth between controlled switches	Controller through OVS commands
Bandwidth between ingress points and provider's network	Controller through OVS commands
Average latency between ingress points and the server	Manually computed
Average packet transmission delay of an ingress point	Manually computed
Average throughput between ingress points and the server	Manually computed
Average jitter between ingress points and the server	Manually computed

TABLE I
NETWORK STATE METRICS

In Table I, the metric representing controlled switches of the provider network along the ingress point's path to the server is a boolean vector of size N_s, i.e., the number of controlled switches in the provider network. Performance metrics are calculated based on the data obtained from traced packets during flow generation phase. These metrics are standard network ones [16]: throughput, latency, delay, and jitter.

The agent's actions consist of three options: (1) controlling bandwidth on the provider side (either increasing or decreasing it); (2) redirecting flows within the provider network (taking into account alternative paths between switches); and (3) taking no action. In the context of our experimental infrastructure, that has 4 controlled switches, the total number of actions is 45.

In our methodology, we define the parameters for bandwidth control through a systematic experimentation process designed to meet the characteristics of our network. Specifically, the minimum bandwidth threshold was established at 0.01 Mbps, while ingress links and intra-provider connections had maximum bandwidth thresholds of 3.1 Mbps and 9.1 Mbps, respec-

tively. Additionally, the bandwidth's increment and decrement steps were determined as 0.3 Mbps.

In this study, we use an asymmetric reward function (illustrated in Fig. 2) as experiments revealed that greater penalty for the RL agent's incorrect actions significantly reduces training time [17]. Our primary objective is to minimize end-to-end latency and jitter in the network as they directly impact quality of service and end-user experience. More precisely, the reward function is composed of sub-rewards, each of them ranges from -3 to $+3$, considering the effect of jitter and latency related to benign flows between the ingress points and the server, as detailed in Figure 2, where "attribute" represents either jitter or latency. Notice that if the agent encounters any other case not illustrated in the mentioned figure, it is assigned a minimal penalty of -0.1.

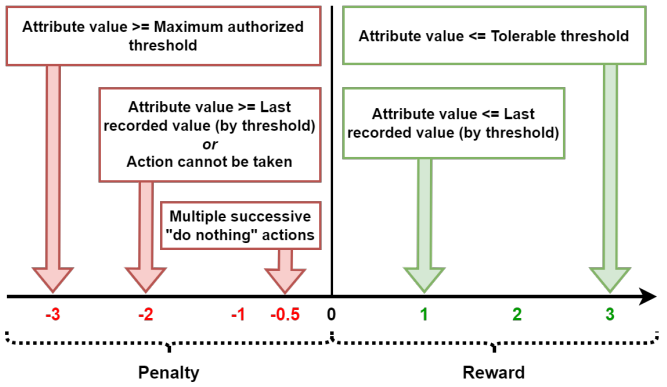

Fig. 2. Performance-dependent reward values for RL agent algorithm.

C. Adaptive Response

In the context of adaptive response in SDN networks, we present a possible scenario of the RL agent's behaviour during an attack exemplified in Fig. 3. In this illustrative case, both legitimate and malicious traffic originate from source r_1, connected to switch s_2. Upon occurrence of a disturbance in network performance, the RL agent dynamically reacts by reducing bandwidth on the link between s_2 and s_0. During this adaptation, if legitimate users' quality of service degrades, which is detected through state measurements conducted by the RL model in section V-B, the agent reroutes their traffic to an alternative path (green link between s_2 and s_3) with enhanced bandwidth, which may involve increasing bandwidth for the new path (link between s_3 and s_0) if needed. Throughout this process, careful consideration is given to the availability and potential congestion of the selected network path. The proposed scenario serves as a depiction of the adaptive capabilities of the RL agent, showcasing its effectiveness in dynamically optimizing network performance under varying conditions.

D. Model Implementation

We train the agent over 50 episodes and 100 steps with DDQN algorithm, using a neural network consisting of an input layer of 128 neurons (using ReLU activation function), three intermediate layers comprising a total of 1208 neurons

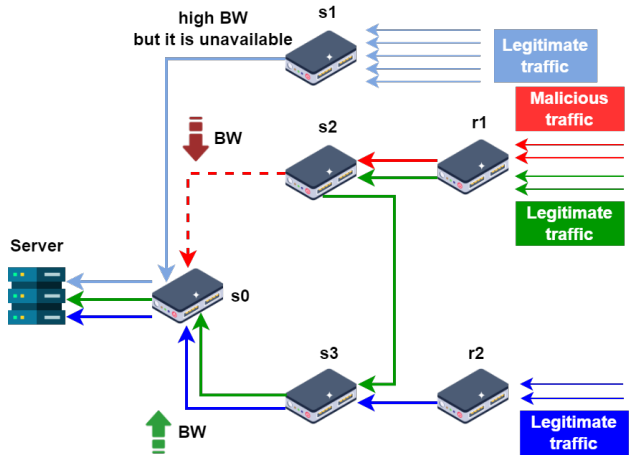

Fig. 3. Actions' selection in our approach.

(the first two layers use ReLU activation function having 256 and 694 neurons respectively, while the final one uses sigmoid activation function with 258 neurons), an output layer composed of 45 output actions along with linear activation functions. For optimization, we employ the mean-squared-error as a loss function, the Adam optimizer, and a stochastic gradient descent method. We use the following key parameters for the DDQN training: epsilon decay is 0.998, discount rate (γ) is 0.85, and the learning rate (α) is 0.01.

VI. Experiments and Results

A. Experimental Settings

1) Software-Defined Network (SDN) Implementation: Our SDN network is implemented using the Mininet emulator [1] with a customized controller as depicted in Fig. 4. Mininet provides a virtualized environment for network emulation, allowing to create and simulate various network scenarios. Meanwhile, the customized controller facilitates efficient control and management of the SDN.

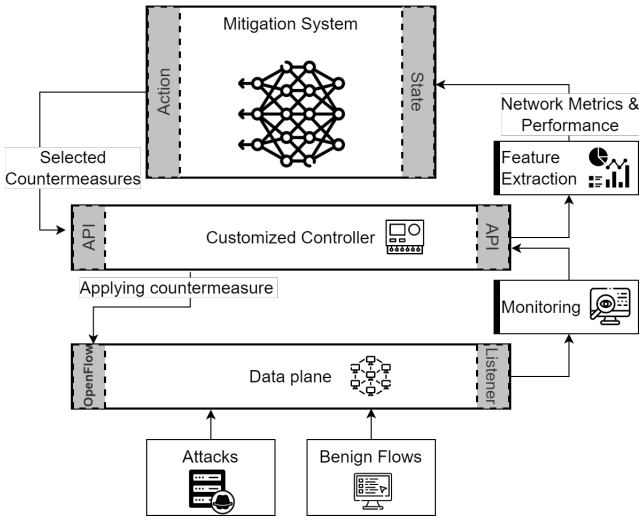

Fig. 4. System Architecture.

[1]https://mininet.org/

2) Traffic Generation: In each step, benign users generate flows using custom benign traffic generation method. These flows persist for a duration of $d_1 = 40s$. Meanwhile, malicious traffic flows generated by the MHDDOS tool [2]) persist for a duration of $d_2 = 30s$.

3) Packet Capture and Network Monitoring: During these flow periods, we use Tshark [3] to capture packets at controlled switches' interfaces, we store the result as a PCAP file. This file is then used by CICFlowMeter [4] to extract features, which are then stored in a CSV file.

4) Controller and State Metrics: The controller obtains state metrics based on the extracted features from the network through its southbound interfaces. Then, it transmits the network's state and performance metrics (calculated in section V-B) to the RL agent using Flask APIs. These APIs enable efficient communication and information exchange, empowering the RL agent to compute action evaluations (rewards).

5) RL Agent: On the RL side, the agent computes the action to be taken, and sends it to the controller. The controller transforms the high-level policy (RL actions) into flow rules configured using Open Virtual Switch (OvS) commands to be applied within the Mininet emulator.

B. Experiment Description

The experiment is conducted on the network depicted in Fig. 1, with the primary objective of assessing the effectiveness of our mitigation strategy under different types of DDoS attacks. The experimental network comprises four controlled switches, a server, an attacker and multiple ingress points. We train our RL algorithm within the Mininet emulator, which operates within a virtual machine operating on Ubuntu 20.04, equipped with a six-core CPU and 2 GB memory, while the physical hosts underlying our experiment operate on an Intel Core i7-12800H CPU. Throughout the training phase, we diversify various parameters such as attack types and traffic volumes to simulate a wide range of network conditions. Additionally, we develop multiple models based on different ingress points within the network under study. Subsequently, we evaluate the efficacy of the trained model by analyzing the agent's behaviour in terms of bandwidth manipulation across different connections between controlled switches. Additionally, we analyze routes' redirections for both malicious and legitimate traffic. This evaluation incorporates a range of network performance metrics such as jitter, throughput, delay, and latency, while considering their impact on honest users.

C. Results

We now report how the agent actions evolved during training iterations, and how it behaved after training completion resulting in remarkable improvements in network service and performance for honest users, as represented in Fig. 5. After training, during the evaluation phase, our approach underwent testing across diverse attack scenarios of one episode of 100 steps each. Initially, upon commencing each scenario with traffic generation, we noticed relatively high average values for latency, jitter, and delay attributed to the attack's intensity. However, as the agent iteratively took actions, notable enhancements were observed. Approximately by the 50th step, the adverse effects of the attack started to diminish, and the QoS steadily improved, reaching its best possible state around the 100th step. At this point, we observed that average jitter and latency for packet delivery decreased to less than $0.2s$, with average flow delay stabilizing at around $0.04s$. These improvements were consistent across various traffic types.

Furthermore, the average throughput for legitimate traffic directed to the server increased to reach approximately $500kbps$ across diverse attack types. In Fig. 6, the proportion of benign

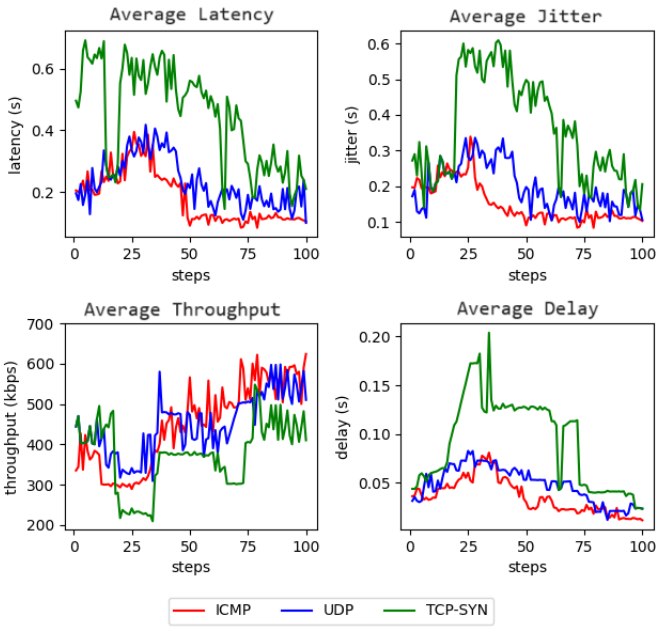

Fig. 5. Average network performance metrics for honest users.

flows reaching the server during different attack types indicated a low percentage at early training stages, not surpassing 28%. However, by the end of training where the algorithm underwent sufficient training and enough convergence, this percentage progressively rose, reaching its maximum value of 80.4% during ICMP attacks.

In Fig. 7, we present the newly directed flow routes of the network (previously represented in Fig. 1 with standard flow routes) after the intervention of our agent and the application of its countermeasures, where red lines represent a majority of malicious flows, while other colors represent a majority of legitimate ones. Notably, our trained agent efficiently mitigated the threat originated from a specific ingress point by rerouting this traffic through a new path passing through the controlled switch (s102). Simultaneously, it constrained the bandwidth for malicious traffic from 4.3 Mbps to 0.4 Mbps, while increasing its allocation for legitimate traffic. These measures were able to mitigate the attack effect on honest users, improving network performance and their service quality.

[2]https://github.com/MatrixTM/MHDDoS
[3]https://www.wireshark.org/docs/man-pages/tshark.html
[4]https://github.com/ahlashkari/CICFlowMeter

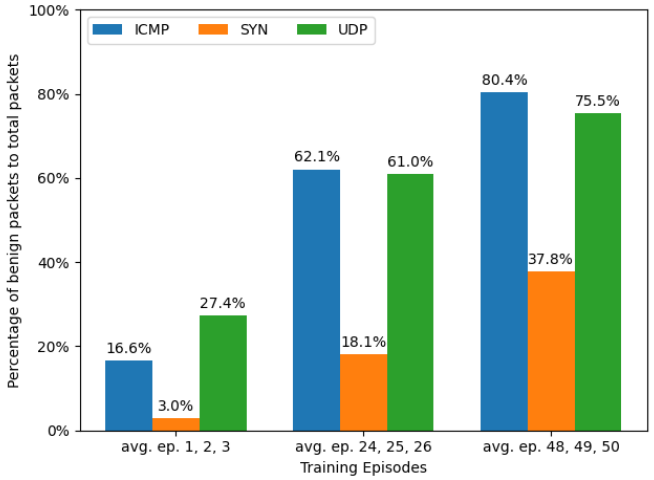

Fig. 6. Average legitimate traffic percentage reaching to the server under different types of DDoS attacks through various training stages.

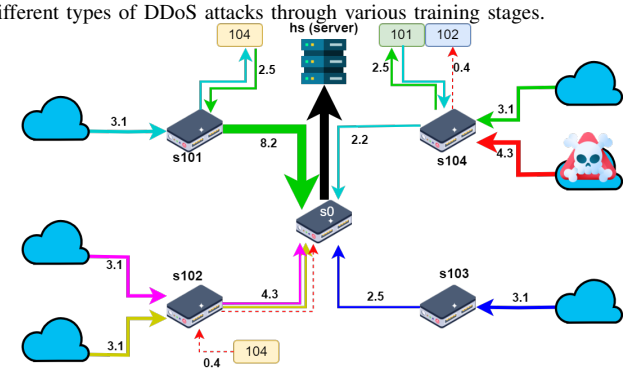

Fig. 7. Network architecture after redirections and bandwidth throttling countermeasures.

To sum up, our proposed mitigation strategy effectively handles diverse traffic volumes while minimizing service disruption to honest users by dynamically adjusting flexible countermeasures based on network performance feedback. However, a limitation lies in occasionally unnecessary redirections, which may raise costs by increasing the number of controlled switches a flow needs to traverse in order to reach its destination, without significantly impacting network performance.

VII. Conclusion

In this study, we proposed a DRL-based framework aimed at mitigating the impact of attacks on network availability, while ensuring optimal network performance for honest users. Our framework employs a DDQN algorithm, trained within a simulated environment reflecting real-world user behaviors and diverse traffic patterns. Notably, our trained agent was able to mitigate various DDoS attacks, including TCP SYN, UDP, and ICMP floods. Moreover, our approach integrates a diverse array of countermeasures, such as traffic throttling and flow redirection. It dynamically adjusts its strategies according to the network situation while preserving the quality of service. Looking ahead, our future direction involves two primary objectives: firstly, enhancing our framework by incorporating a training process that covers all attackers' positions in a single training session and within a feasible training timeframe. Secondly, improving the scalability of the proposed solution, in terms of adding more ingress points and additional controlled switches, without affecting the dimensions of the state representation throughout the training process.

Acknowledgment

This work is funded by the French National Research Agency through the GRIFIN project (ANR-20- CE39-0011).

References

[1] A. Algarni and V. Thayananthan, "Improvement of 5g transportation services with sdn-based security solutions and beyond 5g," *Electronics*, vol. 10, no. 20, p. 2490, 2021.

[2] M. Letourneau, G. Doyen, R. Cogranne, and B. Mathieu, "A comprehensive characterization of threats targeting low-latency services: The case of l4s," *Journal of Network and Systems Management*, vol. 31, no. 1, p. 19, 2023.

[3] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "Ai and 6g security: Opportunities and challenges," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 616–621.

[4] J. Mölsä, "Mitigating denial of service attacks: A tutorial," *Journal of computer security*, vol. 13, no. 6, pp. 807–837, 2005.

[5] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim, "Ddos attack detection method using cluster analysis," *Expert systems with applications*, vol. 34, no. 3, pp. 1659–1665, 2008.

[6] M. Sewak, S. K. Sahay, and H. Rathore, "Deep reinforcement learning in the advanced cybersecurity threat detection and protection," *Information Systems Frontiers*, pp. 1–23, 2022.

[7] A. Mihoub, O. B. Fredj, O. Cheikhrouhou, A. Derhab, and M. Krichen, "Denial of service attack detection and mitigation for internet of things using looking-back-enabled machine learning techniques," *Computers & Electrical Engineering*, vol. 98, p. 107716, 2022.

[8] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Bot-iot dataset," IEEE Dataport, 2018. [Online]. Available: https://ieee-dataport.org/documents/bot-iot-dataset

[9] D. K. Dake, J. D. Gadze, G. S. Klogo, and H. Nunoo-Mensah, "Multi-agent reinforcement learning framework in sdn-iot for transient load detection and prevention," *Technologies*, vol. 9, no. 3, p. 44, 2021.

[10] T. V. Phan and T. Bauschert, "Deepair: Deep reinforcement learning for adaptive intrusion response in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2207–2218, 2022.

[11] I. Akbari, E. Tahoun, M. A. Salahuddin, N. Limam, and R. Boutaba, "Atmos: Autonomous threat mitigation in sdn using reinforcement learning," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[12] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[13] F.-M. Luo, T. Xu, H. Lai, X.-H. Chen, W. Zhang, and Y. Yu, "A survey on model-based reinforcement learning," *Science China Information Sciences*, vol. 67, no. 2, p. 121101, 2024.

[14] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep q-learning," in *Learning for dynamics and control*. PMLR, 2020, pp. 486–489.

[15] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[16] J. Kurose and K. Ross, *Computer Networking: A Top-down Approach*. Pearson, 2022.

[17] A. Bignold, F. Cruz, R. Dazeley, P. Vamplew, and C. Foale, "Human engagement providing evaluative and informative advice for interactive reinforcement learning," *Neural Computing and Applications*, vol. 35, no. 25, pp. 18 215–18 230, 2023.