Challenges in RL based Attack Detection in SDN

Aparna Ganesan
Department of Computer Science
The University of Texas at Dallas
Richardson, USA
Aparna.Ganesan@utdallas.edu

Kamil Sarac
Department of Computer Science
The University of Texas at Dallas
Richardson, USA
ksarac@utdallas.edu

***Abstract*—With the advent of effective data collection and network monitoring techniques, machine learning (ML) algorithms are increasingly deployed to mitigate network attacks. However, traditional ML techniques lack quick adaptability to changing attack nature as they require human involvement between collecting and labeling network data to train ML models. Reinforcement learning (RL) is an online method that enables quick adaptation to changing nature of the observed phenomenon. The main challenge in RL-based network attack detection is to develop an effective reward mechanism that provides useful feedback to guide the RL algorithm in making accurate decisions. In this work, using TCP SYN flood attacks as an example, we explore the possibility of building a reward function that relies solely on network-level metrics to detect attacks using RL. We conduct experiments with our proposed reward function to observe the impact of the agent's actions on the number of packets dropped and the packet retransmission rate in an emulated network environment. We report our findings that show promise as well as challenges in using RL for attack detection.**

***Index Terms*—Attack detection, Reinforcement learning, programmable data planes, security**

I. Introduction

Attack detection for Software Defined Networks (SDNs) requires *effective and scalable* solutions. Several ML-based methods ([1], [2]) have been proposed as solutions for attack detection in SDNs. However, the current ML-based mechanisms are found to be slow in adapting to the changing nature of network traffic as the overall process requires (1) collecting network traffic data, (2) labeling the flows as benign or malicious, (3) re-training the ML algorithm based on this new data, and (4) deploying the new model to do packet classification. We see that with the ever-changing nature of attack traffic, ML-based approaches do not adapt and react quickly when encountering new types of attacks.

Reinforcement learning (RL) models eliminate the need for periodic data collection and labeling for model training. In attack detection with RL, during each episode, the agent observes the network state and chooses an action to perform based on its current policy. After each action is performed, the environment provides feedback based on the quality of the action in the form of a reward. The agent updates its policy based on the reward value. To ensure the timely convergence of the learning algorithm, it is important to design a good quality reward function that can appropriately quantify the impact of the agent's action.

In our previous work [3], we demonstrated a scalable approach to implement an RL-based solution to perform in-network packet filtering. In that work, we capitalized on the centralized architecture of SDN and proposed a hybrid system where the learning happens at the control plane and the enforcement of the models is done at the data plane. One limitation of that work was the use of previously collected and labeled datasets to act as an *Oracle reward* to provide feedback for the agent's actions.

Designing a practical reward function that solely relies on the network signals to provide reliable feedback to an agent is a crucial component of an RL based solution. In this work, using TCP SYN flooding as an example attack on SDN infrastructure, we attempted to develop a practical reward function for the agent to identify attack packet signatures and translate them as rules to the data plane switches in an RL based solution. Please note that, different from traditional networks, in SDN, TCP SYN flood attacks may cause overloading on the controller and saturation on switch-to-controller path. Our focus is on protecting SDN infrastructure from such attacks.

This paper presents our methodology and the learned lessons. More specifically, we (1) demonstrate a way to train an RL agent by interacting with an emulated network environment, (2) develop a reward function to evaluate the performance of the agent's action in mitigating flooding attacks, and (3) discuss challenges associated with automating network attack detection using RL mechanisms. In our experimental work, we gained useful insight and identified various challenges in designing a practical and effective reward function.

In the rest of the paper Section II presents the related work; Section III presents the proposed framework; Section IV elaborates on the implementation; Section V presents the challenges and pitfalls; and, Section VI concludes the paper.

II. Related Work

Significant research has been done on adapting ML solutions for network management. More specifically, the work done in [4] explored the possibility of performing in-network ML at the programmable switches in the data plane using a domain specific language called P4 [5]. Furthermore, several studies ([1], [2], [6], [7]) have focused on offloading different stages of ML training and deployment to the data plane devices. These studies focus on translating supervised ML models onto data plane switches to perform in-network packet

filtering. These works depend on collected and labeled datasets to train the models. In network attack detection, packets are collected and processed before they can be used for attack detection. These ML-based solutions are heavily dependent on datasets used to train the models requiring significant time to collect and label the packets.

RL-based mechanisms train directly on the current network state eliminating the additional time required for labeling. Several RL-based methods ([8], [9]) perform actions that focus on isolating the victim node or perform packet filtering based on *IP addresses*. Such actions in the network either focus on managing the ongoing attack by isolating the nodes or adding a new firewall rule to filter each attack packet. In [3], we focused on translating the actions of the RL agent onto the data plane switches. These actions were designed in such a way that the installed rules partially denote the attack signature of the packet, described by packet features other than IP addresses. The network environment was simulated by replaying a previously collected dataset as the state of the environment and the reward metric was also evaluated with the help of the labeled dataset as an *oracle reward function*. While this method offered significant insights into the feasibility of using RL for network attack mitigation, the reward function is still dependent on a collected dataset.

In this work, we develop an environment that directly observes the changes in network metrics in the emulated network to return the reward value to the agent. Furthermore, the agent does not depend on a previously collected dataset to describe the network state. The agent trains by directly interacting with the network environment.

III. Methodology

The SDN architecture offers network-wide visibility and a centralized platform to implement the RL-based solutions. Here, the agent is implemented on top of the control plane and can observe the events in the network. Based on its current policy and network state, the agent performs an action in the environment. For our experiments, we set up an RL agent for an emulated network with P4 switches in Mininet [10]. We want to develop a framework that facilitates shift towards network automation, specifically for network attack mitigation. We implement our custom data collection pipeline that collects data from the emulated network environment directly and a reward function solely using the network metrics.

Every attack has a varied impact on the network. Therefore, it becomes essential to develop reward functions that are tailor-made for the family of attacks under consideration. In this work, we focus on TCP SYN Flood attack and develop our ideas by considering its impact on the network.While TCP SYN Flood attacks are well studied, we chose this attack to evaluate our proposed methodology. We believe that this same principle may be extrapolated to other families of attacks. We present our methodology in two subsections. One pertains to the network and the considered attack vector and the other pertains to the RL agent and the environment.

A. Network View of a TCP SYN Flood Attack

During TCP SYN Flood attacks, the attacker initiates a connection to the server by sending SYN packets at very high speeds. The server allocates resources for every SYN packet and responds with a SYN-ACK packet to the client. However, the malicious client does not complete the three-way handshake with an ACK packet. If the attacker spoofs the source IP address, it does not result in a final ACK packet. In either case, these connection requests take up memory resources at the victim server, causing legitimate users to be denied service. While this impact is observed at the server level, at the network level, especially in SDN, the use of spoofed source IP addresses in SYN packets causes the switch to send a *PacketIn* message to the controller asking for a flow rule to forward the packet to the next hop. Therefore, the goal is to develop a solution that can isolate malicious SYN requests from legitimate ones with the help of an RL agent at the switch level. The idea is to observe the network and design a reliable reward function for the RL agent, such that the agent does not only denote whether a packet is benign or malicious but also observe malicious packet characteristics, extract a blueprint of the attack packets, and install it in the data plane switches to facilitate packet filtering.

Now that we have established the characteristics of a SYN flood attack, we want the agent to learn the characteristics of an attack packet. During each episode, the RL agent takes the state of the environment as input, and based on the current policy, the agent generates an action to be performed in the environment. The state of the environment is a representation of the current network state in this case. This can be designed in several ways. Some potential state representations may be (1) the links and the state of the links, or (2) the statistics of the number of packets from every possible pairs of source and destination in the network. Whichever state representation is chosen, the selected representation should have the same input dimension during each episode.

The agent takes in the state information and generates an action to be performed at the network based on the current policy. In order to inform the agent about the impact of a particular action, we need to design a reward function that looks at changes in chosen network parameters after the enforcement of the said action. For example, in the case of a SYN Flood attack, the goal is to drop malicious SYN packets before they reach the server. In the presence of an attack, legitimate requests are denied service. If the action successfully drops attack packets, we should see that the benign requests are being served and that no half-open requests are exhausting the server resources. If the action does not correctly identify the attack packets, we will observe the legitimate clients being affected. Therefore, it is important to choose metrics that reflect the impact of the agent's action reliably. For example, congestion in the network links may denote the presence of an active attack in the network. When the congestion in the network links reduces, that indicates that some packets are getting dropped, which may be benign or malicious.

B. Network Environment

We start our experimentation with a simple topology emulated in the Mininet environment. We aim to develop a closed-loop system that has an RL agent running on top of the network that uses real-time network traffic metrics as the state and performs actions in the network without any help from previously collected datasets. Perfecting a system design that can observe the network state and perform appropriate actions for attack detection will help mitigate previously unseen attacks before they are able to bring down the entire network. We will elaborate on individual components of our experimental setup as follows.

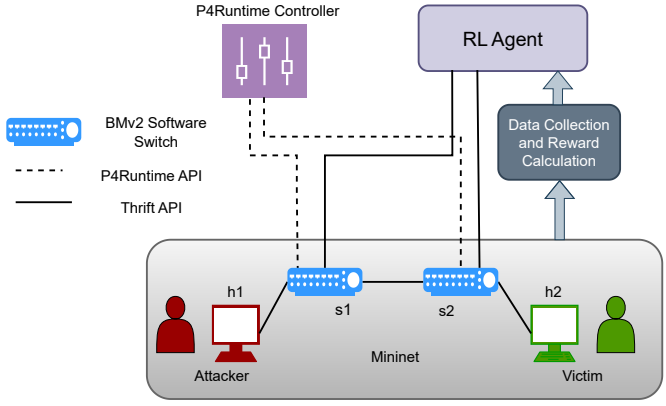

Fig. 1. Topology for In-Network Translation

1) Network Topology: The network topology used for the implementation is denoted in Figure 1. Host *h1* is the malicious host, and host *h2* runs a TCP server application that is under attack. The network link between switches *s1* and *s2* is monitored, and we train the agent to install rules that drop the attack packets generated by *h1* and protect the victim server. We have a Python-based controller that installs routing rules for reachability for the entire network with the help of P4RuntimeAPI. The agent actions are enforced in the network with the help of *Simple_switch_CLI* [11] that is based on ThriftAPI. In our current framework, the controller does not run any routing or security applications to validate the actions of the agent. We believe that action validation and active conflict resolution between the actions of the agent and the controller is an open problem that needs further research.

2) Traffic Generator: Once the topology is up and running, we generate both benign and TCP SYN flood attack traffic in the network. We monitor the link between switches *s1* and *s2* and launch traffic between hosts *h1* and *h2*. For the benign traffic generation, the host *h2* runs a TCP server that waits for connections on a particular TCP port. We launch benign packets from *h1* that emulate a benign TCP client. We run a multi-threaded program that establishes concurrent connections via the TCP three-way handshake protocol and exchanges data packets with packet sizes selected from a predefined range. The application behaves as a TCP client, and in case of duplicate ACKs or time-outs, the client and the server perform packet retransmissions for the missing packets.

For the attack packets, we generate a flood of SYN packets with different source port numbers, targeting the TCP server with different packet payload sizes compared to benign packets. We generate packets at varying packet rates within the limits of the processing speed of the software switch. We launch the attack packets with spoofed IP addresses towards the target. As established earlier, the attacker does not send any packet other than the SYN packet to the client. The attacker does not retransmit lost packets or respond with an ACK packet. The SYN-ACK from the server gets dropped as the packets are launched with a spoofed IP address.

3) Network Statistics Collection: While implementing an RL agent for a data network environment, the agent expects two different forms of information from the environment. During each episode, the agent takes in the network state information and performs an action. After an action is performed in the environment, the agent waits for feedback in the form of a reward value. Therefore, it is important to choose the metrics that carry the maximum information for the agent to make quality updates to the policy.

TABLE I
FEATURES THAT REPRESENT FLOW STATE OF THE NETWORK

Feature List	
Flow Duration	Packet Length Min
Flow Bytes/s	Packet Length Mean
Flow Packets/s	Packet Length Std
Fwd Packets/s	Packet Length Variance
Bwd Packets/s	Active Mean
Total Fwd Packets	Fwd Header Length
Total Bwd Packets	Bwd Header Length
Total Length of Fwd Packets	Fwd Seg Size Min
Total Length of Bwd Packets	Fwd Act Data Pkts
Fwd Packet Length Mean	Flow IAT Mean
Fwd Packet Length Std	Flow IAT Max
Fwd Packet Length Max	Flow IAT Min
Fwd Packet Length Min	Flow IAT Std
Bwd Packet Length Mean	Fwd IAT Total
Bwd Packet Length Std	Fwd IAT Max
Bwd Packet Length Max	Fwd IAT Min
Bwd Packet Length Min	Bwd IAT Total
Packet Length Max	FIN Flag Count
Subflow Fwd Packets	Subflow Bwd Packets

a) State: During each episode, the state of the network environment can be generated in multiple modes. For example, just the link state information of all the links can be used as the state of the network. Similarly, for individual servers in the network, the local server logs can be used to extract the summary of the application state used by the agent. The environment state information can be extracted from the network layer or the application layer. Whether the agent is configured to perform traffic engineering or attack mitigation, the metrics that reflect the appropriate state must be used.

In our design, since the agent gives the packet features as action, we chose the state of the network in the form of flows collected. We develop a data collection application that generates flow data in the link between *s1* and *s2* using Wireshark and CICFlowmeter [12]. The flow features that are

selected to be used as network state are given in Table I. These features were shortlisted based on their relevance and their ability to reflect the difference between malicious and benign packet behaviors for attack mitigation.

b) Reward: Selecting network parameters is very crucial for the successful convergence of the RL agent. The environment should have a module that observes the network after every action of the agent is enforced and responds with a reward value based on the quality of the action. In a real-time network scenario, identifying if a particular packet that is dropped is an attack packet or a benign packet is not straightforward. After each action, we need to observe the changes in the network and decide if the changes seen are for the better or the worse in terms of network stability and evaluate the appropriate reward value. To illustrate, consider a flooding-based attack targeting a server is active in the network. The network links are congested due to the attack packets, causing the benign requests to be dropped. Based on the policy, the agent performs an action on the network that may drop some of the incoming packets. If the action eases up the congestion in the network links, the agent can be given a positive reward value. However, the link congestion parameter cannot discriminate whether the dropped packets are from the attacker or legitimate clients. Thus, the change in network link congestion alone will not be sufficient to judge the action.

During SYN Flood attacks, the attackers do not retransmit SYN packets, only legitimate clients retransmit packets until they are able to establish a connection. Therefore, we could use the retransmission rate to see if the packets dropped are benign or malicious packets. However, using the retransmission rate alone does not indicate if the packet retransmissions are due to the agent's action or due to link or node failure. Therefore, in this work, we utilize the data plane switch buffer state and packet retransmission rate as metrics to evaluate the reward. We use P4 registers at the switch to simulate a virtual queue that keeps track of the number of packets that are dropped due to the agent's actions and the number of packets that are dropped by the switch due to other factors like routing table misses. We use two P4 registers, one to keep track of the number of packets that are dropped due to the agent's action and one to keep track of the number of incoming packets at the switch. After the action is installed in the switch, we observe the link between *s1* and *s2* to identify changes in packet retransmission rate. Thus, the retransmission rate from the link and the number of packets dropped by the agent could identify if a desired action is performed by the agent.

IV. Implementation

In this work, the agent is trained using Proximal Policy Optimization (PPO) Algorithm [13]. We define an OpenAI gym environment that acts as an interface between the emulated Mininet environment and the PPO agent. All the blocks of the framework run concurrently in real-time. That is, once the topology is started, we start sending both benign and malicious traffic in the network between hosts. The data collection unit is also started, and flow statistics are collected.

a) State: The state of the environment is denoted in the form of a 2-dimensional array of size $N \times M$, where N denotes the number of flows and M denotes the number of features chosen to represent a flow. This flow data generated by the CICFlowmeter comprises about 80 features. Out of these 80 features, we removed redundant features and shortlisted only relevant features, as shown in Table I. In addition, the features are shortlisted such that the feature values differ significantly between attack and benign flows. Here, the total number of features used for state representation is 38, that is, M = 38. In addition to the number of features and their characteristics, the number of flows to be included in network state representation is also important. We observed that for smaller values of N, the agent receives fewer flows from the network, which does not denote the complete state of the network. When there are both benign and malicious flows in the network, it is important to choose an N value that is large enough so that the agent is able to see different types of flows in the network during each episode. Thus, we set the number of flows per episode to be 15.

b) Action: The fundamental idea of using RL for network attack mitigation is to ask the agent to describe the characteristics of malicious packets. Rather than asking the agent if a given packet is malicious, we ask the agent to give us the attack signature. The attack signature of a packet may be defined in multiple formats. For example, in IP address-based blacklisting of attack packets, packets arriving from certain ranges of IP addresses are dropped. Thus, the attack signature majorly comprises IP addresses. In other cases, the attack signature may be fine-grained, carrying very specific details of packet features. Ideally, we want the agent to be able to narrow it down to a fine-grained attack signature. In our experiments, we start with two features to define the attack signature. Initially, we start with the packet size and SYN flag of the packet header. We understand that these two features do not completely define the attack signature, but are chosen for this preliminary study.

Given that the benign and attack packet sizes are distinctive, we expect the agent to learn and narrow down on the malicious packet size and install that as a flow rule in the data plane switches. During each episode, 15 flows from the collected flow data are given as input to the agent. Based on the current policy, the agent generates an action which is installed in the data plane switch.

c) Reward: The reward value reflects whether the agent is moving towards performing actions that detect and mitigate the attack. The agent should receive a high positive reward if the attack packets are dropped. However, it is not a trivial task to identify if a particular packet that is dropped is from the attacker or a legitimate user at the victim network. Therefore, the reward function must be designed to provide accurate feedback to the agent solely by observing the network traffic pattern. In this work, we designed the reward function using the statistics from the switch and packet retransmission rate.

The reward function used is shown in Listing 1. If the agent installs an action that matches neither benign nor malicious

packets, then that action will not drop any packets in the network. This indicates that the installed action is not very useful, as the attack is still ongoing, and none of the malicious packets are getting dropped. Even though the action is not dropping malicious packets, it is also not causing any adverse effect in the network by dropping benign packets. Therefore, we configure the reward function to give a low negative reward to the agent. If the installed action starts to drop packets at the switch, we need to determine if the packets that are dropped are benign packets from legitimate clients or malicious attack packets. To determine this, we look at the number of packet retransmissions in the network. If there is a sudden spike in the number of packet retransmissions, that denotes that benign packets are getting dropped. In this case, the agent's action causes further damage in the network, and therefore, the agent receives a high negative reward. If the packets are getting dropped due to the agent's action, but there are no packet retransmissions in the network, that indicates that malicious packets are getting dropped. That is, the agent's current policy is able to achieve the desired behavior. Therefore, a high positive reward value is given to the agent as feedback for the current episode.

In addition to sending appropriate rewards, it is also important to rectify the actions that cause benign packets to be dropped. Similarly, when the installed match-action rules do not change the state of the network by dropping neither benign nor malicious packets, these actions need to be retracted so that the switch memory is not filled with futile rules. Therefore, whenever the reward value is evaluated to be a negative value (low or high), the most recent action is retracted.

```
if number_of_packets_dropped_by_agent < 1:
    return -5
else:
    if number_of_retransmissions > 0:
        return -10
    else:
        return 10
```

Listing 1: Reward Function for the Agent

V. Results and Observations

In this section, we report the results of our experiments along with some observations based on these results. We also elaborate on the challenges and limitations encountered during the implementation of our proposed reward function.

A. Results

In this work, the state is directly extracted from the network environment, and the reward values are given based on the criteria described in Listing 1. We initially train the RL agent for 100 episodes to observe the feasibility of our proposed solution. Even though the agent was trained for a limited training period, its learning behavior provided certain valuable insights. These observations highlight the potential and promise of pursuing research in this direction. To understand the behavior of the agent, we present our results in two parts.

a) Reward: During the initial training episodes, the agent tries to explore the environment and performs random actions. The number of episodes before the agent received a high positive reward is large. That is, the installed rules were futile

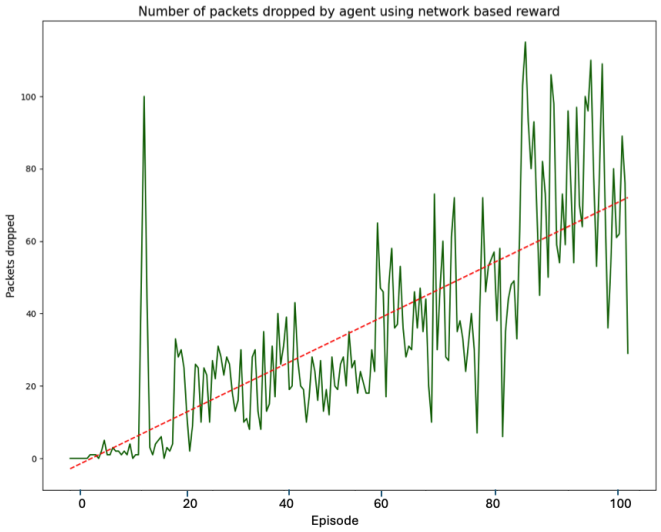

Fig. 2. Packets Dropped by the Agent during Training

for a minimum of the first 20 episodes. We attribute this behavior to the random exploration of the environment by the agent. However, after the initial 20 episodes, we observed that the majority of the actions that the agent installed in the switch received a reward of 10, that is, these actions cause the attack packets to be filtered. The observed reward values demonstrate the expected behavior of random exploration by a new agent.

b) Packets dropped by the agent and Retransmission rate: As the training proceeds, the reward values change and packets get dropped by the agent. Whether the dropped packets are malicious or benign is not known at the victim network. In our experiments, we identify whether malicious packets are dropped using (1) the number of packets dropped at the switch due to the installed rule and (2) the number of retransmissions after each action is performed at the switch. We demonstrate this with the help of the plot in Figure 2 and Figure 3.

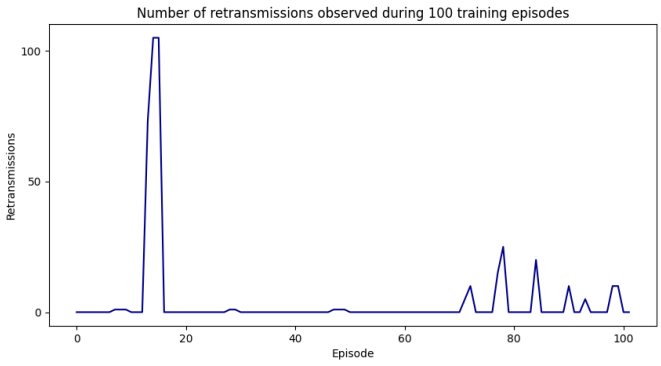

Fig. 3. Retransmissions in the Network during Training

Figure 2 plots the number of packets that are dropped at the switch due to the rules installed by the agent during each episode. During the initial episodes, the number of packets dropped by the agent is zero. As the training progresses and useful rules get added to the match action table, the number

of packets getting dropped gradually increases. We observe that the number of packets dropped spikes suddenly around the 18^{th} episode. We observe that this behavior may be due to the installation of match-action rules that drop benign packets. Once these incorrect rules get deleted, the benign packets are not dropped by the switch, and the count decreases. This can be further validated from Figure 3. Figure 3 plots the number of retransmissions observed in the network during each episode. Even though benign packets are dropped and retransmissions occur at the network, as soon as incorrect rules are deleted the number of retransmissions drops to zero.

Even though the agent was not able to converge to an optimal policy with 100 episodes, these plots demonstrate that with extended training, the RL agents can learn to differentiate benign and malicious traffic patterns in the network and install rules to maintain network stability. Furthermore, the gradual increase in the number of packets being dropped by the agent (Figure 2), along with the number of positive reward values, indicates that the agent is moving towards optimal actions to reduce the impact of the ongoing attack. Therefore, we believe that for the considered SYN flood attack, in the simplest form, this reward function can denote the impact of the agent's actions on the network.

B. Challenges

a) Issues with State Definition: We observed that the state definition used in this work has limited effectiveness for the considered type of network attack. Specifically, we used N-traffic flows from the network traffic collected, characterized by M-features. When an attacker launches a flood of attack packets with different flow IDs, the size of the collected dataset grows drastically and rapidly. However, the agent processes the state information at a constant pace, sequentially from the collected dataset. Therefore as the training progresses, the agent is extracting flow information from older data. In a real-time network scenario, this approach for state definition causes the agent to use past data to perform an action in the present. While this state information is sufficient to evaluate the reward function within the context of our designed experiment, it needs further research for use in real-time production networks.

b) Issues with RL Convergence: The goal for the RL agent is to learn an optimal policy to maximize rewards for every action it performs in the network. The environment must be designed in such a way that the reward value given as feedback accurately reflects the quality of the action. In this work, the number of dropped packets by the agent increased during later episodes of training with the current reward function. However, we observe that the reward function does not effectively discern whether the latest rule is responsible for the packet drops or if one of the older rules is causing packets to be dropped. That is, the first condition merely checks whether any packets are dropped due to the rules installed by the agent. The reward function does not quantify the number of packets dropped by each rule. This is significant because once a valid rule that drops attack packets is installed, it remains persistently in the switch. The switch will drop all the packets matching the rule in the future.

Therefore it is important to develop the reward function with a custom data structure that keeps track of the number of packets dropped by each rule to help the agent to converge at the global optimum. While this is an issue, it is important to note that the proposed reward function immediately removes the rules that drop benign packets from the switch once retransmissions are observed in the network.

VI. Conclusions

In this work, we designed a simple reward function that uses network metrics such as packet retransmissions and switch buffer state to provide feedback to the agent. The agent starts with a random exploration of the environment and slowly learns to install useful rules in the network. In the case of production networks, this random exploration by the agent may lead to an undesirable network state. Therefore, we believe that agents that are pre-trained using previously collected datasets may be able to mitigate known attacks as well as help the agent from having to explore the environment from ground zero. We also observe that the proposed reward function is a rudimentary approach that can send feedback with some reliability only for SYN flood attacks. While it can be tedious to develop different reward functions for individual types of attacks, benign traffic profiling may help in designing a reliable reward function that aims to maintain network stability irrespective of the type of attack. We believe that with a clear characterization of stable network behavior, any action that causes instability in the network may be penalized by the reward function. **Acknowlewdgements**. This work is partially supported by NSF award DGE-1922398.

References

[1] A. Ganesan *et al.*, "Attack detection and mitigation using intelligent data planes in sdns," in *GLOBECOM*. IEEE, 2022.
[2] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021.
[3] A. Ganesan *et al.*, "In-network reinforcement learning for attack mitigation using programmable data plane in sdn," in *ICCCN*. IEEE, 2024.
[4] Z. Xiong *et al.*, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, New York, NY, USA, 2019.
[5] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, 2014.
[6] B. M. Xavier *et al.*, "Programmable switches for in-networking classification," in *INFOCOM*, 2021.
[7] Y. Li *et al.*, "Accelerating distributed reinforcement learning with in-switch computing," in *Intl Symposium on Computer Architecture*, 2019.
[8] I. Akbari *et al.*, "Atmos: Autonomous threat mitigation in sdn using reinforcement learning," in *NOMS*. IEEE, 2020.
[9] M. Zolotukhin *et al.*, "Reinforcement learning for attack mitigation in sdn-enabled networks," in *NetSoft)*. IEEE, 2020.
[10] "Mininet official website." [Online]. Available: http://mininet.org/
[11] "simple_switch_cli." [Online]. Available: https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md
[12] G. Draper-Gil *et al.*, "Characterization of encrypted and vpn traffic using time-related," in *ICISSP*, 2016.
[13] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv preprint*, 2017.