In-network Reinforcement Learning for Attack Mitigation using Programmable Data Plane in SDN

Aparna Ganesan
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas - 75080
aparna.ganesan@utdallas.edu

Kamil Sarac
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas - 75080
ksarac@utdallas.edu

***Abstract*—The development of reinforcement learning (RL) algorithms has created a paradigm where the agents are trained to learn directly by observing the environment and learning policies to perform tasks autonomously. In the case of network environments, these agents can control and monitor the traffic as well as help preserve the confidentiality, integrity, and availability of resources and services in the network. In the case of software-defined networks (SDN), the centralized controller in the control plane has become the single point of failure for the entire network. Reactive routing in SDNs makes such networks vulnerable to denial-of-service (DoS) attacks that aim to overwhelm switch memory and the control channel between SDN switches and controllers. One potential solution to cope with such attacks is to use an intelligent mechanism to detect and block them with minimal performance overhead for the controller and control channel. In this work, we investigate the practicality and effectiveness of a reinforcement learning (RL) approach to cope with DoS attacks in SDN networks that utilize programmable switches. Assuming the existence of a reliable reward function, we demonstrate that an RL-based approach can successfully adapt to the changing nature of attack traffic to detect and mitigate attacks without overwhelming switch memory and the control channel in SDN.**

***Index Terms*—Reinforcement learning, software-defined networks, programmable data planes, SDN security**

I. Introduction

Software-defined networks (SDN) introduced the idea of separating control plane logic from packet forwarding logic, virtually centralizing the control logic, and converting switches to simple packet forwarders. Even though SDN enables better network manageability and ease of functional extensibility, the technology may be vulnerable to potential security threats such as denial-of-service (DoS) attacks. An attacker can send a high volume of novel packets, forcing the switch to send a request to the controller to receive forwarding rules to handle each such packet. This attack can potentially overwhelm the controller, saturate the controller-to-switch control channel, and fill the switch's forwarding table. Defending an SDN environment against these types of attacks becomes an important task to ensure the smooth operation of the network.

The introduction of programmable switches and the P4 language has provided us with the ability to introduce limited control logic to the switches. One can use P4 to implement some limited control logic on the switches to introduce new functionality or improve the performance of existing functions in an SDN network. P4 gives network administrators the flexibility to define the way they want the packets to be processed in the network without being constrained by the protocols supported by the hardware ASIC. P4 allows customization in the packet processing pipeline to include a block that can filter out packets before forwarding them to the controller. In this paper, we use this capability offered by P4 and programmable switches to handle flooding-based DoS attacks without having to rely on the controller for every malicious packet while learning to identify new and previously unseen attack patterns.

Recently, machine learning (ML) algorithms have been extensively used to implement solutions to deal with network attacks such as DoS attacks. It has been shown that supervised ML algorithms [1], [2] can be trained on previously collected and labeled network traffic to effectively classify ongoing traffic as benign or malicious with high accuracy. On the other hand, these algorithms are found to be slow in adapting to the changing nature of the 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. Given that this process requires a non-trivial amount of time and potential human involvement (for labeling), it may have limited effectiveness in network environments where the nature of the traffic changes continuously.

In this paper, we explore a reinforcement learning (RL) based approach to improve attack detection time as the nature of the traffic changes in the network while leveraging available network attack datasets. In RL, the agent builds a policy by interacting with the environment. Initially, the agent explores the environment by choosing actions based on trial and error and receives feedback from the environment as a numerical reward. This feedback helps the agent to continuously update its policy and use the policy to take actions to maintain the desired state of the environment.

When applying RL principles in data networks, we face several challenges. In contrast to certain application scenarios where a relatively small number of variables define the state of the environment, data networks require a larger set of

parameters to fully describe their state. In addition, it is challenging to develop a reliable model of the network that can be used in conjunction with the RL agent. Furthermore, the success of the RL agent depends on the quality of the feedback that it receives from the network. Given these challenges, our main goal in this work is to examine the feasibility of using RL in adapting to the changing nature of attack traffic in the network assuming that we have a reliable feedback (i.e., an *oracle reward*) mechanism available to us. We apply the principles of Offline RL [3] and leverage previously collected and labeled datasets to train the agent. In Offline RL, the authors propose the use of previously collected data for training the RL agents extracting maximum possible utility out of available datasets. In our approach, we use the labeled datasets to provide reliable feedback for the agent to learn optimal policy to filter malicious packets based on dynamic packet and flow features. We consider finding a practical and reliable reward mechanism as a future work.

In our framework, the actions of the agent are translated into the data plane switches as match-action rules. These rules represent the signatures of DoS attacks and allow the switches to detect and drop malicious packets in the data plane. We use Proximal Policy Optimization (PPO) [4] algorithm as our RL agent in an emulated network environment. A custom OpenAI Gym [5] environment is developed as a moderator between the emulated mininet [6] environment and the RL agent. The mininet environment will have a network topology made of P4 programmable switches. We create a pipeline that collects packets in the network as pcap files and use the Cicflowmeter [7] to generate flow-based dataset that is fed back to the agent as the environment state.

In our evaluations of the developed test environment, we see that the match-action rules installed by the agent can filter out malicious packets in an efficient manner. Our contributions in this work include the following: (1) we formulate a flooding-based attack and its mitigation as an RL problem; (2) we develop a custom OpenAI Gym environment that acts as a moderator between the emulated mininet environment and the RL agent; (3) we develop a method to leverage existing datasets to train the RL agent offline to perform actions that mitigate network attacks; (4) we propose a way to install the agent's actions as match-action rules in the P4 programmable software switches without involving the controller for every new attack traffic flow; and (5) we simulate the network scenario by replaying well-known datasets and report the performance of RL in mitigating network attacks.

The rest of the paper is organized as follows. Section II presents the related work. Section III provides a brief discussion on the PPO algorithm and P4 programmability. Section IV presents the theory behind the proposed solution. Section V discusses the implementation and evaluations. Section VI concludes the paper.

II. Related work

With the invention of programmable data plane devices along with domain-specific languages like P4, several solutions have been developed to perform in-network computations on the switches. In [8], the authors present a comprehensive survey of studies that focus on offloading ML operations to network infrastructure devices. In the case of in-network ML, there are several works where different stages of the ML algorithm are implemented on the programmable data plane [1], [9], [10]. Several works off-load training of the ML algorithm onto the data plane where the data plane performs aggregation operations associated with the model updates.

The work done in [9] is an example of utilizing the data planes to perform training and gradient aggregation in the network devices. This work utilizes the features offered by SDNs and uses the programmability of switches to perform the training of the learning algorithms. The authors propose their own protocol and packet format to facilitate direct communication among the switches (workers). In our work, we focus on offloading the RL models onto the data plane to perform inference. In contrast to [9], complex mathematical operations are not carried out at the switches, thereby preserving their high processing speeds. Instead, all the learning and optimizations are run on the control plane and the data plane devices can function as worker nodes, implementing the instructions obtained from the control plane.

In [11], the authors propose a framework to perform both RL training and inference at the data plane, focusing on classical RL methods such as tile coding with algorithms like *Sarsa* [12]. This work primarily addresses the efficient implementation of learning at data plane devices, such as *Netronome* SmartNIC hardware, and presents the effect of such implementation on their performance. The authors also briefly discuss the adaptability of their implementation for DDoS attack mitigation. In our work, we perform RL inference in the data plane by training the RL agent externally to the data plane devices using the *Deep-RL* algorithm (PPO) for the specific task of attack mitigation. We also present the evaluation of our proposed solution for DDoS attack mitigation, demonstrating that the proposed solution efficiently leverages the advantages of both the control (for training) and data plane (in-network packet filtering) of the SDN architecture.

In [13], the authors developed an intelligence system using Deep Q Learning and PPO to perform mitigation against network attacks. The defense framework is implemented on the OpenStack cloud computing platform with Opendaylight controller and Openflow-enabled OpenvSwitches. Similarly in [14], the authors implement an RL-based framework using Deep Q Networks with constraints on exploration performed by the agent. The security constraints are learned in a semi-supervised manner as partial attack signatures. In both [13] and [14], the RL agent learns to install flow rules to drop packets based on the source IP address, requiring a new rule for every new attack flow. In cases where the attacker can launch multiple flows, installing a rule for every new flow may overwhelm the switch memory. In our work, we identify attack signatures based on dynamic packet features like packet count per flow and packet size and add match action rules

in P4-programmable data plane devices. Since we implement attack signatures that are more generic for the considered attack independent of source and destination IP addresses, our method provides a solution against attackers launching DDoS attacks with spoofed source IP addresses.

III. Reinforcement Learning and P4 Programming

The goal of this work is to implement a robust in-network attack mitigation framework that can continuously learn using RL algorithms to filter new types of attacks. The agent runs on top of the controller and installs corresponding match-action rules in the data plane switches to filter out malicious packets. In this section, we summarize the two technologies that form the basic building blocks of the proposed solution, namely, RL and P4 programmability.

A. Reinforcement Learning

In RL, the agent learns its behavior through trial and error by interacting with the environment. It is a closed-loop system where the agent takes an action in the environment based on which the environment moves on to the next state and the agent receives a reward back depending on how useful the action was towards achieving the long-term goal. Based on the interaction between the agent and environment, the collected rewards and environment state are used to perform updates on the policy. In this work, we use a model-free optimization-based algorithm called *Proximal Policy Optimization (PPO)*, a state-of-the-art deep RL algorithm in the class of policy optimization algorithms [4], as our RL agent.

PPO is a policy gradient method. In policy gradient methods, the policy based on which the agent chooses the action is improved at each step directly. The policy $\pi_\theta(a|s)$ is the probability of taking action *a* from state *s*, parameterized by θ, where θ can be the weights of the neural network. The main idea of these methods is to boost the probabilities of actions that earn high rewards while penalizing the actions that lead to low reward values by decreasing their probability. These updates are performed iteratively such that the resulting policy $\pi_\theta(a|s)$ is the optimal policy.

Policy gradient methods are online methods where the agent iteratively learns the optimal policy directly by trial and error based on what it encounters in the environment. This means that policy gradient algorithms do not store past experiences in a replay buffer and learn from stored data. Once the episode is over and the data is used to perform a gradient update, the data from that episode is discarded. This naturally limits the ability of the agent to maximize the sample efficiency. Sample efficiency denotes the amount of training data that a model needs to achieve the desired performance. Since these methods do not store data offline, the agent needs more training episodes to achieve good performance.

Algorithms like PPO were developed to tackle the sensitivity of vanilla policy gradient algorithms with respect to hyper-parameters like learning rate. PPO algorithm tackles these issues by making sure that the new policy update is not too different from the previous policy with the help of simpler first-order methods that can be tuned and implemented at ease while making sure that it has high sample efficiency. PPO achieves this by using clipped policy gradients. That is, the gradient is clipped to a certain range before performing a policy update. If an action is more likely under the current policy than it was under the previous policy, instead of biasing the policy update more and more towards the action, PPO clips the objective function to limit its impact during the gradient update. Similarly, if the action is not returning good rewards with the current policy, PPO makes sure that the action is not penalized to make its probability zero in this current update. This is performed by including the clipping function in the PPO objective function as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\ (r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{1}$$

where $\hat{A}$ is the advantage function, ϵ is a hyperparameter and $r_t(\theta)$ denotes the probability ratio given by

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2}$$

Here, $\hat{A}_t$ is an estimate that is used to measure the quality of a given action in comparison with the average action for a given state. PPO uses a clipped version of the advantage function to prevent unstable policy updates. The PPO objective function makes sure that the new policy update is not too far from the previous policy by taking the minimum of the clipped and unclipped objective. The ϵ is a hyperparameter and is used to lower bound on the unclipped objective.

In addition to advantages like robust policy updates, PPO has been shown to perform better than all the other policy gradient methods for continuous control tasks [4]. That is, the action of the agent could contain several parameters each of which can take any value from a range of values. In our case, since we aim to find the signature of the attack packets based on features of the packet that take continuous values within a range, we use the PPO algorithm to train the agent.

B. P4 Programmability

The main goal of developing the P4 language [15] is to offer flexibility in the data plane switches independent of the target hardware used in them. P4 offers network administrators the freedom to process packets with custom protocols and pipelines. The P4Switch is a software switch that is developed based on a specific switch architecture. The architecture defines the components that are present in the switch with certain components that can be customized and programmed. These programmable components can be modified based on the requirements of the application. In this work, we use *simple_switch_grpc* [16] as the data plane switch. The *simple_switch_grpc* is a variation of *Behavioral Model version2 (bmv2)* [17] P4 software switch, that supports *v1model* architecture.

Among the programmable components, the ingress and egress pipelines have match-action table units. These match-action table units have match keys which can be packet features and action fields such as forward or drop actions. The rules are installed in these tables with key values and corresponding action values by the control plane. For every packet processed by the switch, the key values of the packet are checked against the table and the corresponding action is performed. The data plane switches are programmed with the help of P4 language and the control plane communication is handled by the P4Runtime API [18]. The P4 language offers several externs like counters, meters, registers, etc., which can also be used to customize the packet processing pipeline. The block diagram of the ingress processing with the custom match-action table that is used in this work is depicted in Fig.1.

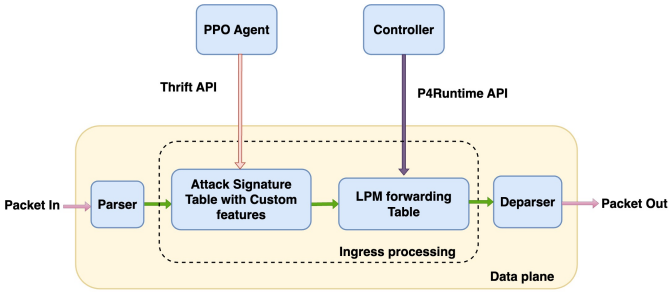

Fig. 1: Ingress Processing with Custom match-action table

IV. Proposed Solution

In this work, we explore the possibility of automating the in-network attack mitigation process using RL. As we previously established, flooding-based network attacks are evolving every day, putting the entire network infrastructure under serious pressure. Therefore, it becomes essential to develop solutions that can evolve and learn new attack signatures in a continuous manner. We propose that with reliable and accurate feedback, the RL-based solutions can learn the optimal policy to filter new types of attacks in a continuous manner. The major contribution of our work is to develop a framework where the RL agent's actions are directly installed in data plane devices to perform in-network mitigation of network attacks. We configure the action parameters as packet and flow features that the malicious packets are expected to take. We use a separate channel using ThriftAPI [19] to install the match-action rules directly into the switch bypassing the controller to protect it from being overwhelmed by the attack traffic.

We develop our solution in two stages. Firstly, we focus on evaluating the efficiency of RL-based methods to learn attack signatures given that they have reliable reward calculation mechanisms. We leverage well-known previously collected datasets to train the agent offline and evaluate its effectiveness in choosing actions that filter malicious packets. Then, we propose a framework to adapt RL-based solutions for real-time network environments by translating the actions of the agent in the form of match-action rules to be installed on data plane switches to perform in-network packet filtering.

a) Stage 1: During the initial stages of learning in RL, the agent has limited knowledge about the environment. Hence, the agent relies on random exploration of the environment by performing actions without any reliable policy. In the case of real-time network environments, these random actions by the agent could potentially cause the dropping of benign packets. Therefore, in this work, we perform initial training by replaying previously collected datasets and evaluating the agent's performance before deploying it in the network environment. Training the agent with pre-existing datasets ensures that the agent starts with a useful initial policy, thereby preventing blind exploration of the network environment from scratch.

In our work, we use the PPO algorithm to perform policy updates in the agent. In RL, the policy is defined as the decision function that decides the action given an input state. During each episode, the agent receives the state of the environment as input and performs an action based on the current policy for the given state. While training with previously collected datasets, batches of flows from the dataset are given as input to the agent during each episode.

Index	Features	Feature type
1	Source port	Packet-level
2	Destination port	Packet-level
3	Total packets in forward direction	Flow-level
4	Total packets in backward direction	Flow-level
5	Average packet length	Packet-level
6	ACK Flag	Packet-level
7	CWE Flag	Packet-level
8	SYN Flag	Packet-level
9	Number of flow packets per second	Flow-level
10	Number of forward packets per second	Flow-level

TABLE I: Features used for defining network state

We replay the packet flows from well-known DDoS datasets namely CICDDoS2019 and CICIDS2017 during the first stage. These datasets have more than 80 features containing both packet-level and flow-level features. In this stage, we perform feature engineering to rank and select the features of the dataset. The purpose of selecting traffic features is twofold: (1) The environment state that is given as input to the agent is given in terms of recent flow information that is described by traffic flow features. The selected features are listed in Table I. (2) The action parameters that are given as output by the agent are also packet/flow feature values that the attack packets/flows are expected to take. These feature values will be used to install match-action rules in the data plane switches to perform in-network packet filtering. Therefore, it is important to select features that can be expressed as match-action rules in data plane switches as well as describe the network state efficiently for the agent. Here, we shortlist features that are most relevant and can be used to train the agent and filter attack packets in the data plane switches. We use subsets of features in different combinations in the match-action table of the switch to describe the attack signature. The feature combinations that are selected to be used as the agent's action

parameters and installed as match-action rules in the data plane switches are listed in Table II.

Index	Feature pairs used as Action parameters
1	SYN flag Count, Destination Port
2	ACK flag Count, Source Port
3	ACK flag Count, Destination Port
4	SYN flag Count, Source Port
5	Destination Port, Packet Length Mean
6	Source Port, Packet Length Mean
7	SYN flag Count, Packet Length Mean
8	ACK flag Count, Packet Length Mean

TABLE II: Feature combinations used as action parameters

We perform the training of the agent in an offline manner. Instead of allowing the agent to obtain network state from the network directly, we leverage previously collected datasets to simulate network scenarios. During each episode, the environment state is provided in batches of ten flows from the dataset, simulating the ten most recently seen flows in the network. The agent receives the state information from the environment as the ten most recent flows and returns the action values based on its current policy. Here, the agent's actions do not merely indicate whether a packet is benign or malicious. Instead, we configure the agent to analyze the most recent flows seen in the network and identify packet/flow feature values that a malicious packet/flow could potentially exhibit. This enables us to install rules at the switch to drop any future packets/flows that enter the network carrying the same feature values.

In a traditional RL scenario, once the action is performed in the environment, the agent receives feedback from the environment in the form of numerical rewards. The agent updates the policy based on the numerical reward for the next episode. Therefore, the numerical reward from the network is very important for the agent to learn the optimal policy. In our case, while training with the previously collected datasets, we create a database of feature values that malicious and benign packets exhibit from the dataset to provide reliable rewards to the agent. For every action given as output by the agent, we check if the match-action rule would filter benign or malicious packets using the lookup dictionary and provide reliable rewards. Given that our reward decision comes from labeled data, we call this reward mechanism as *oracle reward*. Oracle reward is not practical in real-life deployment scenarios but in our work, it helps us evaluate the merit of RL-based solution to the problem. We leave the selection of a practical reward mechanism as our immediate future work.

b) Stage 2: In Stage 1, we proposed an RL-based solution for attack packet detection in data networks. In this step, we present our framework to implement the solution in an SDN environment, as shown in Fig. 2. The major contribution of this work is to develop the framework to impart some intelligence into the switch to filter malicious packets based on dynamic packet features instead of relying on the static 4-tuple header information of the packet. We propose that utilizing dynamic packet features in match-action tables to filter malicious packets optimizes the switch memory

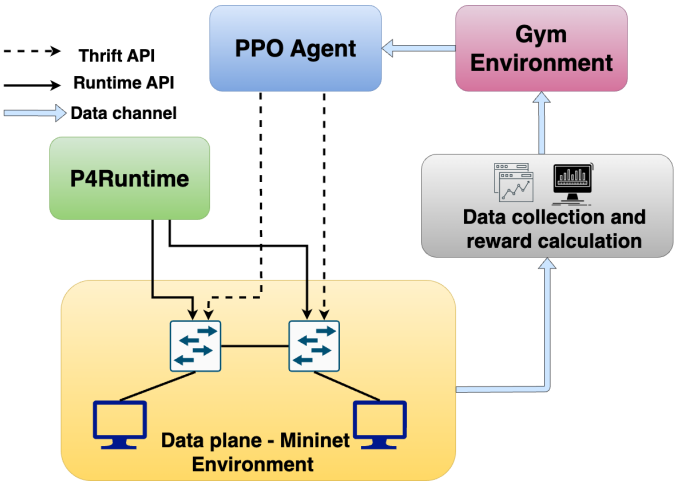

Fig. 2: Proposed framework

usage when compared to filtering packets based on source and destination IP addresses. The components of the proposed system are as follows.

PPO agent The PPO agent takes in the network state information and performs an action based on its current policy. The agent learns attack signatures in terms of dynamic feature values and these values are translated to the switch in the form of match-action rules. In this work, we drop the packets that match with the match-action rule installed by the agent. The packets that do not match the installed rules are considered as benign and they are processed normally by the switch. The PPO agent directly communicates with data plane switches with the help of ThriftAPI installing rules directly without involving the controller. We modify the P4Switch packet processing pipeline by incorporating a match-action table block to facilitate the installation of packet filtering rules from the PPO agent. The agent runs PPO algorithm from the *Stable-Baselines3* implementation with *MlpPolicy* [20].

P4Runtime control plane The P4Switches in the data plane are controlled and managed by the P4Runtime-enabled controller. The controller supporting P4Runtime API is responsible for installing rules to forward packets that enter the network. In this work, we use the *p4runtime_lib* python library to write a custom python program emulating a controller that supports P4.

Data collection unit This unit is used to collect and process the traffic information from the network. Several metrics like bandwidth utilization and latency can be used to describe the state of the network. In this work, we configure our agent to utilize traffic flow patterns and flow statistics as the environment state to perform necessary actions to filter out malicious packets and mitigate network attacks. In this work, we use CICFlowmeter [7] to extract the packet flow information and transmit relevant flow information as network state to the agent in batches. The agent executes actions based on its current policy for the given network state and installs a match-action rule in the P4 switch via the ThriftAPI. Based

on the impact of the action, the data unit would calculate the reward for the action and send it as feedback to the agent. Identifying a reliable reward calculation methodology solely utilizing the signals from the network is left as our immediate next work.

In the P4 packet processing pipeline, the match-action table that is dedicated to the attack signature is placed ahead of the forwarding table as shown in Fig. 1. This means that when a packet reaches the switch, it is initially matched with the *action signature* table. If it matches feature values indicative of a malicious packet, it is dropped. Only packets that are identified as benign are further processed based on the forwarding table of the switch. The controller is involved only in installing forwarding rules for previously unseen benign flows. This configuration protects the controller from being overwhelmed by the flooding of malicious flows. We present the empirical results showing the effectiveness of RL for packet classification in Section V-B.

A. Challenges

The goal of performing attack mitigation in-network using RL algorithms is to automate the process of identifying new attack signatures without having to involve the controller for every new attack flow. To achieve this goal, certain challenges need to be addressed:

Ensuring scalability of RL-based solution Several RL-based methods [21] have been developed where RL agents located on top of the controller have been configured to perform actions on a per-flow basis. This model is not scalable as it requires transmission of every new flow to the agent site for decision making which overwhelms both the control channel and the controller. In our work, the agent is configured to output packet or flow feature values that a malicious packet can take. Instead of transmitting every new packet to the agent, we collect the packets and extract flow information at the data plane and then send them to the agent in batches as the network environment state. For every state, the agent action will be used to install a match-action rule in the switch to filter any packet flow matching the attack signature in the future. This approach then allows the data switches to make local decisions on incoming packets without having to flood the controller.

Feature selection A network environment has a large number of variables that are difficult to keep track of and use efficiently while training the agent. It is important to use the domain knowledge and shortlist the features that are expected to provide maximum information to implement the RL algorithm. We perform feature ranking and utilize the features that can be collected and processed at ease by the data plane devices.

Reward calculation In RL, having a reliable reward function is critical for the agent to learn optimal policy. In the case of mitigating network attacks, a reward function is expected to help the RL agent identify and drop attack traffic and allow benign ones. Given that finding such a reward function is not a straightforward task, in our work, we use an oracle reward function that gives us reliable feedback. This strategy allows us to focus on our main goal of understanding the feasibility of using an RL-based approach for close-to real-time attack mitigation as the nature of the attack traffic changes in the network. Finding a practical and effective reward function is left as our immediate future work.

Generalizing for other attacks In this work, we only consider DDoS flooding attacks from a labeled dataset [22] to emulate a real-time network scenario by replaying the dataset. We provide a proof-of-concept solution with a single type of attack to demonstrate the feasibility of using RL algorithms to perform attack mitigation. We believe that by developing a reliable feedback mechanism that characterizes the expected behavior of target (benign) traffic patterns, one can train the RL agent to install rules to filter out packets that do not conform to the target (benign) traffic profile. We plan to extend our methodology to other attack scenarios to perform an extensive study in our future works in this direction as well.

V. Implementation and Evaluation

In this section, we present our simulation environment to experimentally evaluate the performance of an RL agent in mitigating flooding-based attacks. The RL agent is trained with the help of a custom OpenAI Gym environment that is configured to replay packets in our datasets. The datasets used in this work are CICDDoS2019 and CICIDS2017 datasets where the former one introduces TCP SYN Flood attacks and the latter one introduces benign flows into our experimental environment.

In this work, we use the *p4runtime_lib* python library to write a python program to emulate an SDN controller that supports communication with P4-based data plane devices based on P4Runtime API specification [18] and emulate our network environment using *mininet* [6]. This environment allows us to show that the attack signature generated by the agent as feature values can be installed in the data plane switches to filter malicious packets locally. In the following, we first present the implementation details of our solution and next report on our evaluation of its performance.

A. Implementation Details

The agent runs the PPO algorithm and learns by interacting with a custom OpenAI Gym environment. This environment is modeled to replay packets in batches for the agent as the network state. In RL, during each episode, the agent receives the most recent network flows as the environment state. The features used to define each flow are listed in Table I. In practice, this Gym environment would act as an intermediary between the network environment and the RL agent. In this section, we discuss the implementation detail of each of the components that are necessary for the agent to learn the optimal policy to filter attack packets efficiently.

a) State: In the case of SDNs, several metrics can be used to define network state. Metrics like bandwidth utilization, throughput, latency, and packet loss rate provide insights into overall network performance and status. In this case, we aim to develop a solution to filter malicious packets based on

dynamic packet and flow-based features rather than blacklisting flows based on source and destination IP addresses. We aim to use other dynamic features that distinguish malicious and benign packets. Therefore we define the state of the network environment in terms of packet and flow-based metrics based on previously collected well-known datasets([23], [22]). The list of 10 features used to define the traffic flow state of the network is tabulated in Table I.

For the simulation environment, we replay packets from previously collected and labeled datasets as the environment state. For each state, a batch of 10 flows from the dataset is sampled and given to the agent. Out of approximately 80 features in the datasets, a subset is used to express the network traffic state. During training, the next state is given to the agent based on its current action: If the current action of the agent installs a rule that drops malicious packets, the next state will have flow information comprising predominantly benign packets. When the agent's action drops benign packets and attack packets are untouched. The next state will comprise malicious packets, as the agent's action is not contributing positively to mitigate the ongoing attack. If the agent's action filters neither benign nor malicious packets, that action has no impact on the current state of the network. Hence, the next state will contain flow information of malicious flows as the attack is still ongoing. As a result, the next state seen by the agent reflects the effect of performing the previous action. The next state of the simulated environment is designed to resemble real-time network behavior that is expected for each action during flooding-based attacks.

b) Action: Attack detection is inherently a classification problem. While deploying an RL agent to perform attack detection and mitigation, it is important to ensure that the agent's actions are not merely classifying packets as benign or malicious. As mentioned earlier, using the RL agent as a classification engine is not scalable when the attacker launches flooding of packets from spoofed IP addresses. Therefore, in this work, we train the agent to identify attack signatures of packets in terms of dynamic features and install the rules in data plane devices to perform in-network packet filtering. The rules are installed with priority values such that newer rules are selected over the older ones until outdated rules are deleted. To manage switch memory more efficiently, we periodically purge older rules based on their priority value. In each experiment, the agent is configured to generate feature values for a pair of features from Table II.

This table is placed ahead of the forwarding table in the packet pipeline as shown in Fig. 1. The RL agent communicates with the data plane switches with the help of Thrift API [19]. For flow-based features, we use counters to evaluate the feature values and match the values using the *attack_signature* table before the packet is forwarded to the next hop based on the forwarding table. An example of the P4 table for attack signature with destination port as *hdr.tcp.dstPort* and packet length as *hdr.ipv4.totalLen* is shown in code listing 1.

```
table attack_signature{
    key = {
        hdr.tcp.dstPort: exact;
        hdr.ipv4.totalLen: exact;
    }
    actions = {
        drop;
        NoAction;
    }
    size = 4096;
    default_action = NoAction;
}
```

Listing 1: Ingress Processing with Custom match-action table

c) Reward: In RL, the agent updates its policy based on the numerical reward returned by the environment. In the case of data networks, one way of calculating rewards could be based on the impact of the action on network links. If the action leads to the dropping of attack packets, this will be reflected in the network links, and the reward can be calculated accordingly and fed back to the RL agent. We see that relying solely on the bandwidth to calculate rewards for the RL agent is unreliable. This is because any dropped packet (benign or malicious) contributes towards easing the bandwidth utilization. If the agent's actions install match-action rules that drop benign packets, the goal of attackers is achieved with the agent's help. Therefore, we need a dependable reward calculation mechanism for the agent to differentiate between attack and benign packets. In our approach, we use previously collected datasets and create a database of benign and malicious packet signatures as an oracle feedback mechanism that gives us reliable rewards.

While training the simulated system, we give reward values depending on the packets that are dropped. For every action of the agent, we cross-reference the feature values with the benign or malicious database. If the feature values match malicious packets, the system gets the highest positive reward value. If the agent installs rules that drop benign packets according to the oracle, we give the highest negative rewards. When neither malicious nor benign packets are dropped, we give a lower negative reward value to signal the agent that its action does not impact the network and we need the agent to update its policy to perform actions that drop attack packets.

B. Evaluation Results

In RL, the agent learns an optimal policy to perform actions in the environment such that the actions yield maximum rewards. In this work, the goal of the RL agent is to learn an optimal policy that can install match-action rules in the form of feature pairs to filter attack packets in the network.

We configure the agent to identify values for the considered feature pairs that a malicious packet could potentially present with. This ensures that any malicious packets matching these installed values are dropped by the data plane switches. Each

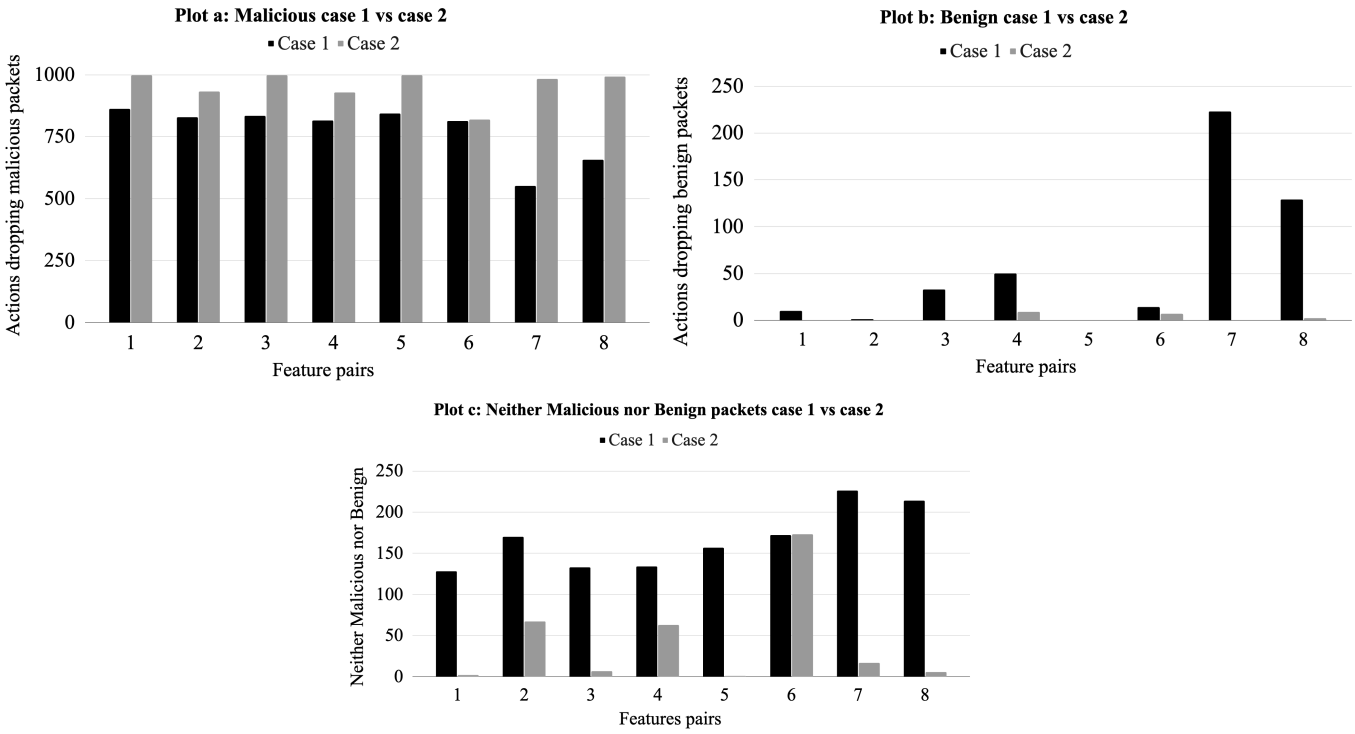

Fig. 3: Comparison between continuous policy updates and separate testing case

Index	Features	Malicious packets		Benign packets		Neither Malicious nor Benign	
		Case 1	Case 2	Case 1	Case 2	Case 1	Case 2
1	SYN flag Count, Destination Port	862	998	10	0	128	2
2	ACK flag Count, Source Port	829	933	1	0	170	67
3	ACK flag Count, Destination Port	834	999	33	0	133	1
4	SYN flag Count, Source Port	816	928	50	9	134	63
5	Destination Port, Packet Length Mean	843	999	0	0	157	1
6	Source Port, Packet Length Mean	814	820	14	7	172	173
7	SYN flag Count, Packet Length Mean	551	983	223	0	226	17
8	ACK flag Count, Packet Length Mean	657	992	129	2	214	6

TABLE III: Agent actions for 1000 time steps

rule installed by the agent could belong to three possible categories based on the dataset used for training. The feature pairs installed could match (1) malicious packet signature, (2) benign packet signature, or (3) there are no packets in the considered dataset with the installed feature pair at all. Therefore, we evaluate the agent's performance by reporting the number of positive (filter malicious packet), high negative (filter benign packet), and low negative (filters neither malicious nor benign packets) rewards the agent receives by following its policy. In Table III, we report the performance of the agent based on the rewards it receives during 1000 episodes for 8 different pairs of features as action parameters from two sets of experiments.

In case (1), we perform an analysis similar to a supervised ML case, where the model is trained using a training dataset, and its performance is evaluated using a test dataset. We split the dataset into train and test datasets. The agent is trained for 150000 time steps by replaying flows from the training set. Once the training is complete, we evaluate the quality of the final policy by replaying 1000 episodes from the previously unseen test dataset. The policy is not updated during testing and the agent performs its actions based on the final policy it has learned at the end of training. The testing is performed for 1000 episodes and we report the number of positive and negative rewards the agent receives based on the learned policy.

In case (2), we continue to train the agent with unseen data and allow updates on the policy. We perform 151000 episodes of training with the entire dataset and report the rewards the agent receives during the final 1000 episodes of training. The results are presented as a comparison between case 1 and case 2 for three different scenarios (1) the number of actions installed by the agent dropping malicious packets correctly (2) the number of actions installed by the agent dropping benign packets (3) the number of actions of the agent dropping neither malicious nor benign packets. In the case of actions dropping

malicious packets, we see that when the agent continues to learn, it can perform actions that yield high positive rewards around 90-98% of the time compared to using a static policy without allowing updates (Col. 3 vs Col. 4, Table. III). For the case of actions that filter benign packets, we see that the static model drops a significant number of benign packets. This behavior is undesirable as the goal of making the network unavailable for benign users is achieved with the agent's help.

We see that this number is significantly reduced in continuous update cases (Col. 5 vs Col. 6, Table. III). Similarly, the actions that drop neither benign nor malicious packets are high for static policy cases for all pairs of features. With continuous policy updates, this number is drastically low for all pairs of features (Col. 7 vs Col. 8, Table. III). These results can be observed from the plots in Fig. 3. We see that training the RL agent continuously helps it learn the changing nature of attack traffic and perform well in all three scenarios while the static policy case filters fewer malicious packets and more benign packets, while consistently installing large numbers of insignificant match-action rules that do not contribute towards filtering attack packets.

Since we translate these actions as match-action rules into the switch directly, it is important to ensure that only useful actions are installed as match-action rules for optimal usage of switch memory. Consequently, we protect the controller from becoming a bottleneck during flooding attacks. Since the principle of RL is to learn by experience, we show that with continuous policy updates, the agent can adapt and learn new attack signatures to help mitigate flooding attacks, while ensuring that the number of benign packets dropped is negligible.

These results show that given a reliable reward function, the proposed closed-loop RL framework can learn attack signatures to filter new attacks and protect the network. We see that performing in-network RL-based packet classification can help protect the network from the ever-changing nature of attack traffic. The attack signatures in the form of the agent's action can be used in several creative ways in P4 switches to make sure the number of benign packets that are dropped is near zero. Each of the feature combinations in Table III can implemented as a standalone unit or as an ensemble of match-action tables that drop packets only when the packet is deemed malicious by a majority of tables ensuring that fewer benign packets are dropped.

VI. Conclusion

In this work, we implement an RL agent on top of the control plane of the network with the sole purpose of mitigating flooding-based attacks. We see that utilizing RL algorithms with programmable switches helps with migrating towards automating attack detection operations for production-grade networks in a scalable manner. We observe that configuring the agent to identify dynamic feature values for installing match-action rules is more scalable than requiring the agent to perform packet classification for every new attack packet. We see that with continuous training RL can discern new attacks and flag them with the help of appropriate and reliable feedback from the network. In future work, we plan to work on exploring practical and reliable reward calculation mechanisms for RL to tackle network attacks and other operational issues in SDNs.

Acknowlewdgements. This work is partially supported by NSF award DGE-1922398.

References

[1] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021.

[2] A. Ganesan and K. Sarac, "Attack detection and mitigation using intelligent data planes in sdns," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022.

[3] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[6] "Mininet official website." [Online]. Available: http://mininet.org/

[7] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd ICISSP*, 2016.

[8] R. Parizotto, B. L. Coelho, D. C. Nunes, I. Haque, and A. Schaeffer-Filho, "Offloading machine learning to programmable data planes: A systematic survey," *ACM Computing Surveys*, 2023.

[9] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[10] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *INFOCOM*, 2021.

[11] K. A. Simpson and D. P. Pezaros, "Revisiting the classics: Online rl in the programmable dataplane," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–10.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[13] M. Zolotukhin, S. Kumar, and T. Hämäläinen, "Reinforcement learning for attack mitigation in sdn-enabled networks," in *2020 6th IEEE conference on network softwarization (NetSoft)*. IEEE, 2020.

[14] A. Mudgerikar, E. Bertino, J. Lobo, and D. Verma, "A security-constrained reinforcement learning framework for software defined networks," in *ICC*. IEEE, 2021.

[15] P. Bosshart, D. Daly, G. Gibb, M. Izzard *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, 2014.

[16] "simple_switch_grpc." [Online]. Available: https://github.com/p4lang/behavioral-model/tree/main/targets/simple_switch_grpc

[17] "Behavioral model (bmv2)." [Online]. Available: https://github.com/p4lang/behavioral-model

[18] "P4runtime specification." [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[19] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook white paper*, 2007.

[20] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[21] K. A. Simpson, S. Rogers, and D. P. Pezaros, "Per-host ddos mitigation by direct-control reinforcement learning," *IEEE Transactions on Network and Service Management*, 2020.

[22] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *ICCST*, Chennai, India, 2019.

[23] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *4th ICISSP*, Portugal, 2018.