EIReLaND: Evaluating and Interpreting Reinforcement-Learning-based Network Defenses

Steven Cheung*, Jared Claypoole*, Prakhar Sharma*, Vinod Yegneswaran*
Ashish Gehani*, Susmit Jha*, John Emanuello+ , Ahmad Ridley+
***SRI International, +Laboratory for Advanced Cyber Research**

Abstract

This paper explores the application of goal-oriented machine-learning techniques, and in particular leveraging recent advances in deep reinforcement-learning (RL) networks to automatically discover deviations from normal network activity and choose actions that mitigate unwanted network flows. We develop a software-defined network (SDN)-based evaluation environment called EIReLaND, using OpenAI gym and ContainerNet. We demonstrate that EIReLaND successfully mitigates three popular attack techniques (TCP state exhaustion, CPU/memory exhaustion, brute-force). Finally, we identify the strengths and blindspots of these trained models using four different interpretability techniques.

1 Introduction

Despite decades of development, network monitoring and defense systems [12; 28; 26; 22] still face challenges due to increasing volumes of traffic, diversity in application mix, and attack sophistication [13; 36; 33]. Our specific objective in this work is to develop a framework for explainable reinforcement-learning-based network defense. To that end, we begin with a formulation of highly dynamic and autonomous network defenses as a deep reinforcement learning (RL) problem. Here, we consider the application of RL for common types of attacks that modern networks are subject to – specifically, distributed denial of service (DDoS) and brute-force attacks – and conduct our experiments in the context of software-defined networks (SDNs) [9]. However, the broad formulation can be extended to traditional networks and other attacks.

We describe our work in the context of a new framework to design and evaluate SDN-based RL defenses that we call EIReLaND. The broad objectives of this system are three fold. First, we need the capability to experiment with RL-based defenses in both simulation and emulation, for a variety of network attack scenarios (*usability*). Second, we need the ability to evaluate against a broad class of implementation choices for our modular system (*modular extensibility*): client traffic modeling (different attack generation tools, attack patterns, and benign/malicious IP subnet distributions); RL agent design choices (various RL algorithms and policy model architecture choices); and RL environment design (choice of system observables to track and means of combining them into a reward function). Finally, we need to be able to reason about the correctness and performance of generated RL-models (*interpretability*). Our work is informed by a few notable recent efforts [8; 18; 2; 1] that have attempted to develop RL-based environments for evaluating network-based cyber-defenses; however, none of them consider model interpretability.

The reward function is a crucial component in any RL system, serving as the agent's only means of feedback on the policy it learns. In our experiments, the reward function corresponds to some aspect corresponding to preservation of network health (i.e., allowing continued operation of normal network activities while preserving security), which is sensed through the continuous analysis of the live sensor data as the new defenses are activated. We formulate our approach using an input space of observations from network monitors, application logs, and host system logs. We employ RL to learn an optimal policy, mapping network state to defensive actions such that the reward – corresponding to network health – is maximized. Defensive actions include fine-grained policy decisions to neutralize attack vectors, such as flow interventions and network filter deployment. These policy adjustments can also be passive, such as the dynamic reconfiguration of mechanisms aimed to improve intrusion detection and characterization accuracy, or to tune activity logging to continually adjust the fidelity of the security audit trail.

To validate the efficacy of the RL approaches, we conducted comprehensive simulation and emulation studies for different attack scenarios. Specifically, we considered a suite of model-free[1] RL algorithms [24]. These include policy-based algorithms (Proximal Policy Optimization (PPO) [31], Advantage Actor Critic (A2C)

[1]Model-based approaches require specification of an explicit model and are thus less generalizable. Exploring these is future work.

and Asynchronous Advantage Actor Critic (A3C)) [17], and Q-learning-based algorithms (Deep Q Network (DQN) [39]). Our reward functions involve features from different sources: network-layer features like IP addresses, network flow features, host features like CPU and memory utilization, and application-level features such as outcomes of SSH authentication events. We systematically explore various dimensions of the design and attack space, including use of state vectors, and characteristics of attack and benign traffic (e.g., distribution of attack and benign endpoints, and timing and intensity of traffic).

A fundamental challenge with deep-learning systems is lack of explainability [5]. To address this challenge, we develop a set of interpretability analysis techniques that enable a network analyst to explore and reason about RL policies generated by the system. Specifically, we develop four complementary analysis techniques that provide a deeper perspective into the models. The first is decision tree analysis that distills the models into human-readable decision trees. The second is an input-space decision analysis technique that illustrates how the learned policy model makes its decisions by analyzing the decisions it makes as a function of its inputs. Third, we describe an input-space confidence analysis technique that provides a more holistic view on the decision boundaries where the model performance degrades. Finally, we describe attribution analysis techniques that identify the key features that contribute to the decision process and how they vary in different regions of the input space.

Contributions: In summary, we make the following contributions:

- Develop a new SDN-RL framework for evaluating autonomous network defenses
- Present three different network attacks as a reinforcement learning problem
- Experimentally validate the proposed SDN-RL defenses in both simulation and emulation environments
- Identify the strengths and weaknesses of trained models using four different interpretability analysis techniques

2 Background

2.1 RL Algorithms

RL algorithms may be broadly classified into two categories: model-based and model-free algorithms. Our work on ElReLaND so far has been exclusively focused on model-free algorithms which in turn may be divided into policy-optimization-based and Q-learning-based algorithms.

Policy-optimization algorithms: These algorithms optimize parameters either directly by gradient ascent (or descent) on the performance objective $J(\pi_\theta)$, or indirectly, by maximizing local approximations of $J(\pi_\theta)$. Examples of policy-optimization algorithms that we consider include PPO [31], A2C, and A3C [17]. While A2C and A3C directly maximize performance by performing gradient descent on the performance objective function, PPO does so indirectly by using a surrogate performance-objective function. These algorithms are referred to as *on-policy* algorithms as each update only uses data collected from acting on the most recent policy.

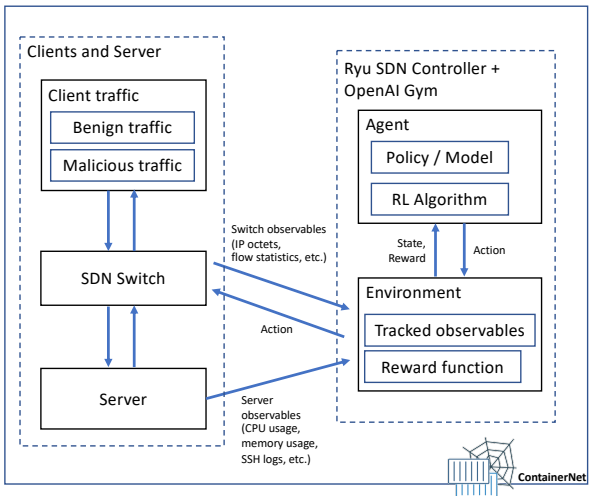

Figure 1: High-level overview of the ElReLaND architecture and its components.

Q-learning algorithms: These algorithms work by learning an approximator $Q_\theta(s, a)$ for the optimal action-value function $Q^*(s, a)$. Unlike policy-optimization algorithms, these are *off-policy*, i.e., each update step may use data collected at any previous point during training. DQN [39] is a classic example of an off-policy algorithm that we study in this paper.

2.2 Motivating Attack Scenario

We have studied the efficacy of ElReLaND using three broad attack classes, namely, volumetric denial-of-service attacks (e.g., TCP SYN flood [34]), algorithmic-complexity-based resource exhaustion (e.g., Apache range-header attacks [16]), and application-level bruteforce attacks (e.g., SSH bruteforce password guessing). Our goal here is not to develop better defense than existing solutions against those attacks, but to investigate the versatility of our RL-based approach against various network attacks. For the sake of brevity, we use SYN flood as a running scenario in this paper.

SYN flood is a canonical example of TCP state exhaustion attack that exploits the state allocated for three-way handshakes in establishing TCP connections. In a SYN flood attack, an adversary crafts and sends a large number of spoofed SYN packets to a target. When the target receives these SYN packets, it will send SYN-ACK packets to the clients, and keep track of the "half-open" TCP connections using the connection table data structure. However, the adversary is unable to complete the spoofed handshake as it does not receive the SYN-

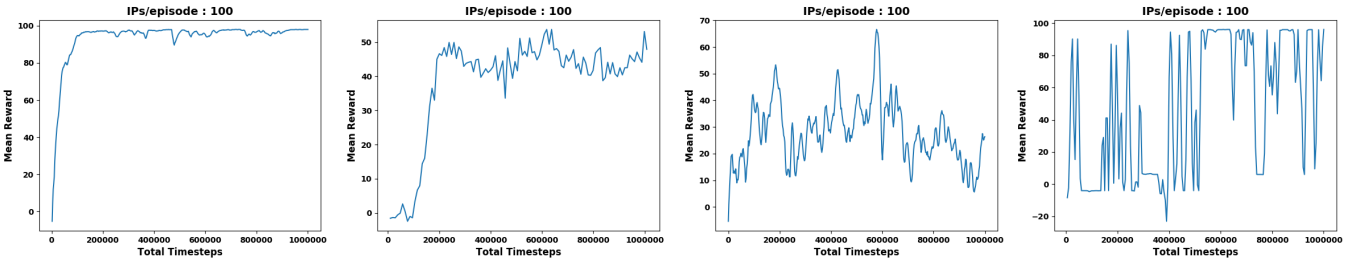

Figure 2: SYN Flood Attack Mitigation [Case 1; Simulation]: PPO (left), DQN (middle-left), A3C (middle-right), A2C (right)

ACK reply from the server. As a result, the connection table of the target will be filled up with half-open TCP connections that prevent other clients from establishing TCP connections with the target.

3 Experimental Setup

We employ a two-tier, simulation-emulation experimental approach for conducting empircial evaluations. In a nutshell, we performed initial exploration of the effectiveness of RL algorithms, observables, and reward functions using simulations, and conducted emulation experiments for the candidates that presented promising results. Compared with emulation, simulation enables us to explore a much larger space of RL algorithms and key parameters quickly, while emulation involves fine-grained, long-running experimental environments to study their effectiveness in a more realistic setting.

For simulations, we use the OpenAI Gym environment [25] (reinforcement learning toolkit) and Stable Baselines3 [23] (learning library based on the Gym API), which supports a myriad of RL algorithms, including PPO, DQN, A2C, and A3C. Our emulation platform is developed over ContainerNet [27] (software-defined network emulator based on Mininet and Docker containers) using the Ryu [32] SDN controller that we integrated with OpenAI Gym.

Figure 1 depicts the high-level topology for the software-defined network used for our experiments. The OpenFlow switch is connected to an OpenFlow controller, which handles network flows that don't match any flow rule in the switch, and makes changes to the flow table of the switch. In this work, the OpenFlow controller employs reinforcement learning algorithms to make decisions based on information about network flows (e.g., numbers and sizes of flows) and data collected from the servers (e.g., CPU and memory utilization). Harpoon was used for benign traffic generation [35]. Scapy [30] and Medusa [20] were used for attack traffic generation.

4 Experimental Evaluation

In this section, we describe evaluation results that attempt to answer the following research questions:

- **RQ 1:** Can RL-based defenses be used to model and effectively mitigate various forms of network attacks?
- **RQ 2:** How robust are these defenses to various choices of RL algorithms, hyper-parameters, reward functions, and attacker/benign subnet distributions?
- **RQ 3:** Can we use interpretability methods to understand how the learned policies are performing?

4.1 Evaluating SDN-RL Defenses in Simulation and Emulation

We conducted simulations to investigate the effectiveness of various RL algorithms under different attack settings. Specifically, we tested the following RL algorithms: PPO, DQN, A2C, A3C, Soft Actor Critic (SAC), and APEX-DQN. For attack settings, we studied various distributions of benign and attack subsets, from using a single benign IP address subnet whose first two octets were disjoint from those of the malicious subnets, to more challenging cases that used multiple benign and malicious subnets with more overlap in the first two octets.

For the reward function, we computed the number of correct actions (i.e., blocking or allowing TCP connection attempts from IP addresses) minus the number of incorrect actions. In other words, for an IP address that is a source of SYN flood attacks, the reward for the RL agent is incremented by 1 if the action is block; otherwise, it is decremented by 1. Similarly, for a benign IP address, the reward is incremented (resp. decremented) by 1 if the action is allow (resp. block).

There are two subtle points that are noteworthy here. The first is that rewards provided are cumulative delayed rewards, and not instantaneous rewards. We use delayed rewards because we don't know if the *allow* action was for a benign or malicious connection attempt until after the TCP handshake has had a chance to be completed. The second point is that while the ElReLaND system never has explicit knowledge of the ground truth of any flow, in the case of the SYN flood attack, the system may infer ground truth of any flow it allows by checking whether the TCP handshake succeeds. This means a blocking action introduces a tradeoff: while blocking a flow may

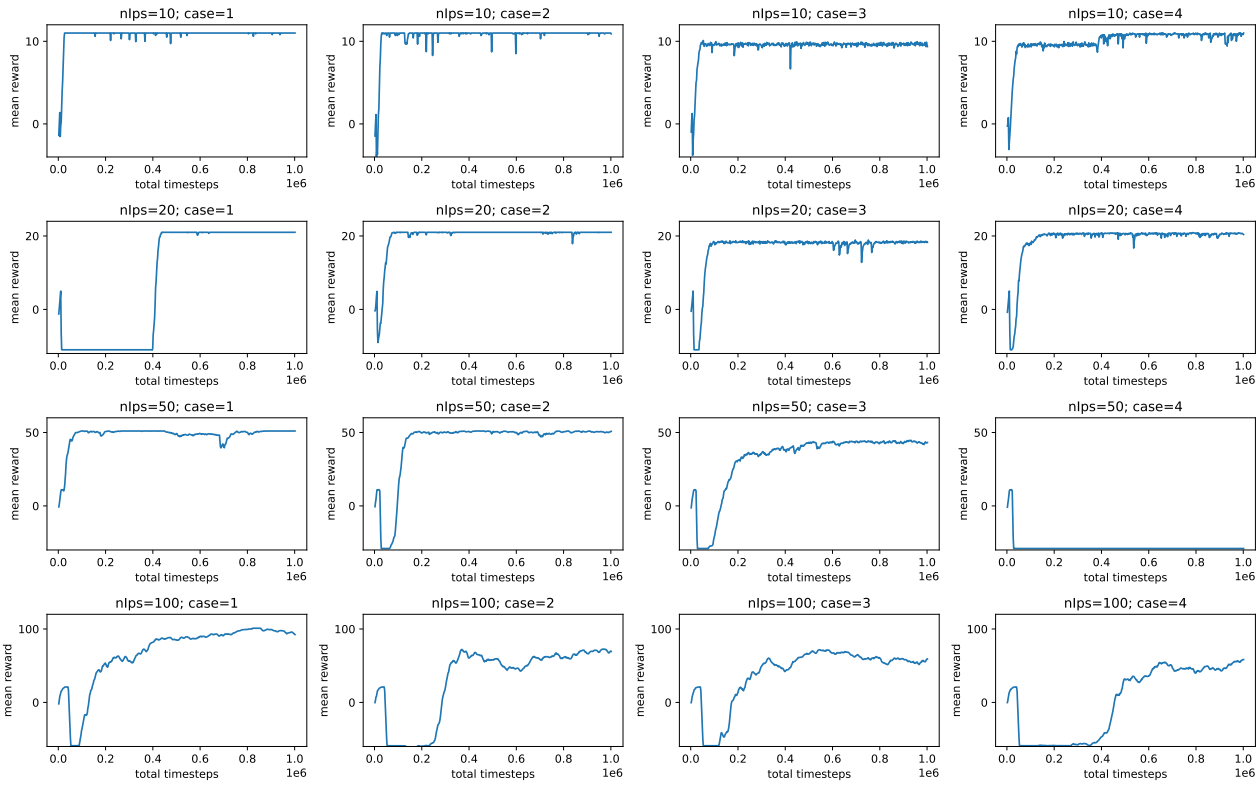

Figure 3: SYN Flood Mitigation [Cases 1-4; Delayed Reward (nIPs/episode): 10, 20, 50, 100; Simulation: PPO]

help mitigate the attack, it also prevents reward signal feedback about whether the block action was correct. In this work, we chose to allow all flows during the learning phase, regardless of the RL agent's recommended action, to determine perfect ground truth.

RL Algorithms: We performed simulations using different RL algorithms to compare their efficacy for mitigating SYN flood attacks. The results for PPO, A3C, DQN, and A2C are depicted in Figure 2. The x-axis corresponds to training timesteps, and the y-axis corresponds to the observed reward. Each timestep corresponds to arrival of a new IP address. Delayed rewards are provided at the end of each episode. Maximum reward per episode is 100. Using the 4-byte vector state representation for detecting and mitigating sources of SYN flood attacks, we found that PPO (Figure 2 (left)) converged quickly to the optimal reward value of 100. A3C and DQN were unable to achieve high rewards. For A2C (Figure 2 (right)), although it reached high reward values, they fluctuated significantly. In other experiments, not shown here, we employed SAC and APEX-DQN; we found that SAC performed poorly, and APEX-DQN failed to converge unless we reduced the delayed reward interval to once every 50 IP addresses instead of once every 100 IP addresses.

Malicious IP Address Distribution: We studied different degrees of "separation" between the benign subnet addresses and the malicious subnet addresses. Intuitively, the more the separation, the easier it is to learn a model to distinguish between benign and malicious subnets. In particular, we studied the following four cases with decreasing separation between benign and malicious subnets:

(1) Only one benign subnet with the first two octets disjoint from those of the malicious subnets. The benign subnet address is 10.1/16, and malicious subnet addresses are A.B/16, where $A, B \in [100 - 255]$

(2) For cases 2-4, we employed multiple benign subnets with subnet addresses 10.1/16, 20.5/16, and 130.105/16. The first octet of malicious subnet addresses may be smaller or larger than that of one of the benign subnets. Specificially, the malicious subnet addresses are A.B/16, where $A \in [50 - 100, 150 - 200]$, and $B \in [150 - 255]$

(3) The first two octets of malicious subnet addresses may be smaller or larger than that of one of the benign subnets. The malicious subnet addresses are A.B/16, where $A, B \in [50 - 255]$, $A \neq 130$, and $B \neq 105$.

(4) The malicious subnet addresses are A.B/16, where $A \notin \{10, 20, 130\}$ and $B \notin \{1, 5, 105\}$

Simulation Results: Using PPO as the RL algorithm, we evaluate robustness against SYN flood attacks for a variety of malicious IP distributions (cases 1-4) and

delayed reward values (nIps per episode ranging between 10 and 100), as shown in Figure 3. We note that the maximal reward is roughly equal to the number of IPs per episode (nIps).

For smaller delayed reward intervals, the learned policies quickly converge to a steady state with few fluctuations. This is expected, as the learning problem is less complex. For larger delayed reward intervals, the learned policies tend to converge more slowly, to lower reward values, and with larger fluctuations in the quasi-steady state. We note that even for the largest delayed reward values we explored (nIps=100), all four cases display reasonable training progress.

We also note that case 1 for nIps=20 and case 4 for nIps=50 experience delayed or failed convergence that appear to break the trend in the vertically adjacent training plots, which can be explained by non-determinism in the training process.

For validation, we have conducted experiments to compare the results for using simulation and emulation to evaluate using the PPO algorithm for protecting SDN against SYN flood. We found that both approaches yield comparable results.

Discussion: For RQ 1, our experimental results showed that the RL-based approach is effective for defending SDN from various forms of network attacks, including SYN flood, range-header, and SSH brute-force.

For RQ 2, we evaluated four different RL policies (PPO, A2C, A3C, and DQN) and found significant differences in their performance against the the attacks with PPO performing the best. We also studied various experimental settings, including several benign and malicious IP address distributions. We found that the RL-based approach is generally robust across different settings, although we observed degraded performance for the more challenging scenarios (e.g., Case 4 in Section 4.1).

4.2 Interpretability Analysis

Previously in this analysis, we evaluated the performance of learned policies using plots of aggregate metrics, such as percentage of flows of each type (benign, malicious) that are allowed and dropped, the reward value, and observables such as CPU and memory usage (for range-header attacks) and percentage of allowed flows which resulted in successful connections (for SSH brute-force attacks).

To perform finer grained analyses of the learned policies' performance, we now present a variety of interpretability techniques, each of which is useful in different circumstances. First we present decision tree analysis, followed by input space decision analysis, input space confidence analysis, and finally attribution analysis.

Decision tree analysis: We train a decision tree as a proxy for the learned policy, showing where the policy overwhelmingly makes mistakes. We provide training examples for the decision tree by freezing the policy and caching its predictions in a clone of the environment in which it was originally trained. Inputs to the decision tree are the same as the inputs to the original policy model (the current state of the environment), and the expected prediction used as "ground truth" to train the decision tree is the original policy's chosen action.

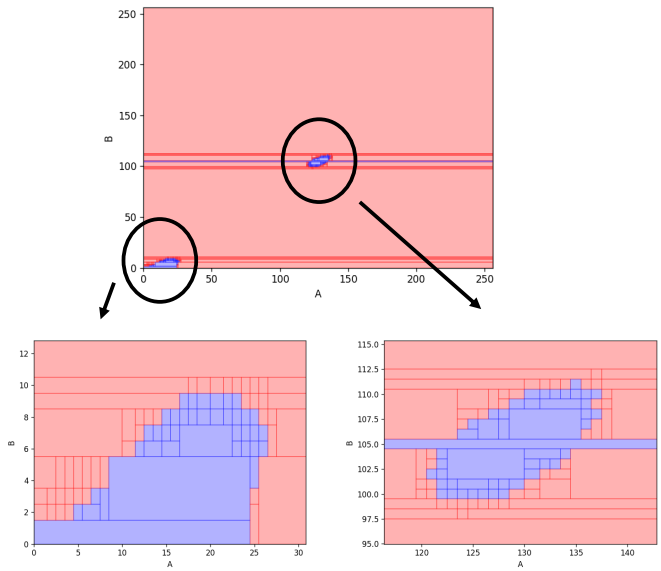

Figure 4: Visualizing the decision boundaries of a decision tree proxy for a learned policy for SYN flood, case 4. This is a more complicated case, where the decision tree is less concise. Blue regions indicate regions of IP octet A,B space the decision tree classified as benign, and red regions were classified as malicious. The only truly benign subnets were 10.1, 20.5, and 130.105.

By construction these decision trees provide binary output: all examples which fall on a particular leaf are labeled by the decision tree as either allowed or blocked. As each leaf corresponds to a region in input space, this method will only detect mistakes in the learned policy if the mistakes are being made for a significant fraction of examples.

Thus the decision tree divides the input space (primarily the possible values of the four IP octets) into disjoint regions, and it assigns a single decision value (allow or block) to each region. This is desirable if indeed the original learned policy made uniform decisions in a given region; however, if the original policy assigned different actions to examples within a single such region, the decision tree proxy will reflect only the majority action. This smoothing has a natural de-noising effect, but it can ignore a strong minority of coherent mistakes.

The decision trees we discuss here were trained only with octets A and B as input. In simple cases, this was enough to capture the full activity of the policies, which didn't make any mistakes. (See SYN flood cases 1 and 2.) In the more complex case (SYN flood case 4), we trained the decision tree proxy only on octets A and B for ease of visualization. We note that training on only a subset of the inputs exacerbates the "smoothing" effects

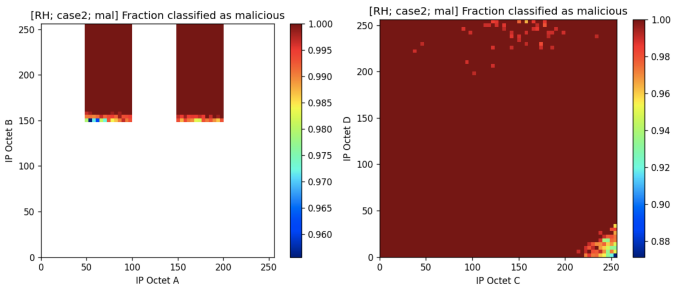

Figure 5: Input space decision analysis for range-header case 2, indicating the fraction of truly malicious IPs which were correctly identified, as a function of different IP octets. This model performs well, so the fractions are high.

discussed above.

In simple cases, we were able to completely characterize the learned policy using a decision tree with only a single decision rule:

- SYN Flood Case 1: Learned Decision Tree: `if A <= 55, allow; else, block`
- SYN Flood Case 2: Learned Decision Tree: `if B <= 127.5, allow; else, block`

In more complicated cases, the tree can have many (more than 10) decision rules, which makes it difficult for a human to interpret directly. We can visualize the decision boundaries to make it more human understandable. See Figure 4 for an example, where the decision tree visualization highlights that while the policy correctly classifies the truly benign subnets, it also misclassifies the immediately surrounding regions as benign.

Input space decision analysis: We attempt to more directly understand how the learned policy model makes its decisions by analyzing the decisions it makes as a function of its inputs.

Because the model receives four IP octets as its inputs, it is challenging to directly visualize the decisions as a function of the inputs. Instead, we plot an indicator of the decisions versus only two IP octets. A natural choice of pairings is octets A and B (which meaningfully discriminate between benign and malicious IP addresses) and octets C and D (which never fully discriminate between benign and malicious IPs). We split each axis into bins and plot a heatmap showing the fraction of correct decisions made in each bin. For ease of visualization, we make separate plots for (ground truth) benign and malicious IPs.

We note that our particular choice for visualization (fraction correct) requires knowledge of the ground truth (benign or malicious) for each IP. One could imagine making a slightly different visualization (e.g., the fraction of IPs classified as malicious) which does not require ground truth, which might be useful to examine the performance of a policy learned in the wild, where ground truth is unknown. This comes at the cost of mixing the benign and malicious IPs, which can have very different distributions, making the plots more difficult to interpret.

We evaluate the learned policy in the environment in which it was trained, noting which input features result in mistakes. Figure 5 shows an example for range-header simulations. These plots show that the few mistakes the model makes are concentrated in regions where IP octet B is small, as well as where C is large and D is small, or more sporadically where D is large. There are also more mistakes when octet A is small.

Input space confidence analysis: In certain cases, we can obtain an even more granular picture of how the learned policy model makes its decisions by looking not at the decision the policy makes, but rather at the quantitative output from the underlying multi-layer perceptron (MLP) model.

Under the hood, the model's dense layers output a "logit" score for each possible decision ('allow' and 'block', corresponding to the classes 'benign' and 'malicious'). The scores are then normalized by a softmax function, allowing them to be roughly interpreted as probabilities (assigned by the model) of belonging to each class. Because there are only two classes and the probabilities sum to one, we can characterize the model's output by a single probability. We choose to look at the probability corresponding to the 'malicious' class, which we will refer to as p_{mal}. The policy makes a decision based on the model's output using a majority threshold: if $p_{\text{mal}} \geq 0.5$, the policy blocks the IP; otherwise, the policy allows the IP.

First we look at the distribution of p_{mal} over all evaluation examples, keeping in mind the imbalance of benign and malicious examples. (Each episode of our simulations consists of a fixed number of IPs (between 10 and 100), 20% of which are benign.) We can also look at the distributions separately for ground truth benign and malicious IPs. In specific cases, we can further subdivide the benign and malicious sets of IPs into different groupings. For the range-header case we'll continue discussing from the previous section. A a natural choice is to split the benign IPs into the three A.B subnets (10.1, 20.5, 130.105), and the malicious IPs into $50 < A < 100$ and $150 < A < 200$, as shown in Figure 6.

Next we look at a scatter plot distribution of p_{mal} versus each of the four IP octet values. We can do this for any of the "bins" of IPs defined above: all, benign, malicious, or further subdivisions described above.

Figure 7 illustrates an example applied again to range-header case 2, focusing on malicious IPs. We immediately notice that above a certain threshold in octet B (approximately $B > 160$), the model makes entirely correct predictions, with uniformly high confidence. Below this threshold, however, the model makes more and more predictions of lower confidence, including some mistakes. This agrees with the story told by the input space decision analysis (Figure 5), where mistakes were only made for malcious IPs with low values of octet B. However,

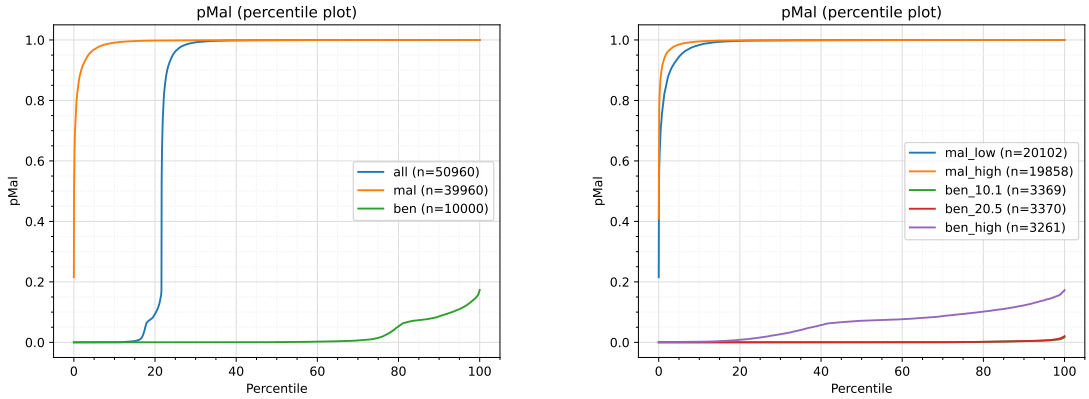

Figure 6: Percentile plots, indicating the distribution of p_{mal} for different groupings of IPs. (Left) Groupings: all IPs; malicious IPs (mal); benign IPs (ben). Note that approximately 20% of all IPs are benign. (Right) Groupings: malicious IPs with $50 < A < 100$ (mal_low); malicious IPs with $150 < A < 200$ (mal_high); benign IPs with A.B=10.1 (ben_10.1); benign IPs with A.B=20.5 (ben_20.5); benign IPs with A.B=130.105 (ben_high). We see that the benign 103.105 subnet is classified as benign with lower confidence than the other two benign subnets, yet always with $p_{\text{mal}} < 0.2$. Note that it's difficult to distinguish the curves for the benign 10.1 and 20.5 subnets on this plot, as they mostly coincide.

this analysis paints a more complete picture: while only a small fraction of the IPs were classified incorrectly, many more were classified correctly but with low confidence (p_{mal} closer to 0.5 than to 1). Rather than identifying a discrete decision boundary, we can instead identify a larger region where the model's confidence in its predictions slowly decreases.

Attribution analysis: We compute attributions of the model's decision with respect to its input features (primarily the 4 octets of the IP address), and look at how they vary in different regions of input space.

We use the integrated gradients [37] method, implemented by the captum [14] library. Note that the reinforcement learning environment library we used, stable-baselines3, doesn't directly expose the output of the underlying pytorch MLP model when making a prediction, so we needed to use a lower level API to access the underlying model itself. Then we were able to pass the appropriate model to captum and compute attributions.

We align "sorted feature plots" (Figure 8) with attributions (Figure 9), where horizontal axes align between the two plots and each x-axis value corresponds to a single example. We make a few observations. First, for all malicious IPs, the attribution due to octet B is larger in magnitude than not only that due to any other single input, but also the sum of any subset of other inputs. This indicates the model always considers octet B most strongly, even when the model assigns a malicious IP low confidence or misclassifies it altogether. One puzzle is how this can be true while the mistakes are so highly concentrated near certain extreme values of octets C and D (see plots of fraction classified as malicious vs octets C and D in Figure 5), yet the model assigns relatively small attributions to octets C and D in those regions.

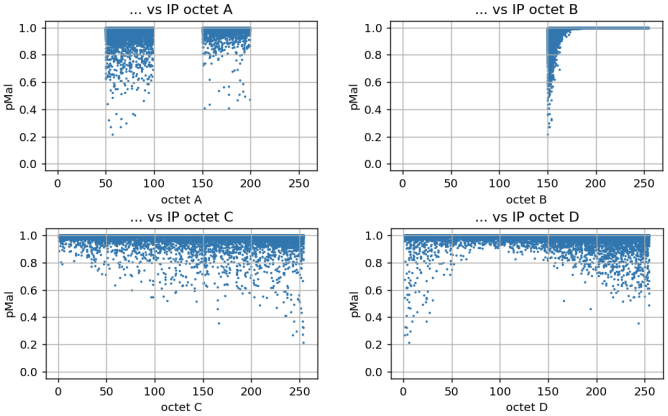

Figure 7: Plots of p_{mal} vs each of the four IP octet values (A.B.C.D) for malicious IPs. $p_{\text{mal}} < 0.5$ indicates an incorrect prediction that the given IP is benign.

Second, the model assigns uniformly low attribution to CPU and memory usage. This indicates that the model bases its decisions on the other inputs instead (i.e., the IP octets).

5 Related Work

5.1 RL-based Network Defense

Luong et al. [15] surveyed applications of RL for network management to optimize network performance under uncertainty and for complex and large networks. For SDN, Dake et al. [4] presented a more recent survey on using RL approaches for traffic engineering. Han et al. [10] studied the feasibility for applying RL for securing SDN. To facilitate studying the efficacy of RL-based SDN defense approaches in a robust, generalizable, and reproducible manner and to support modeling

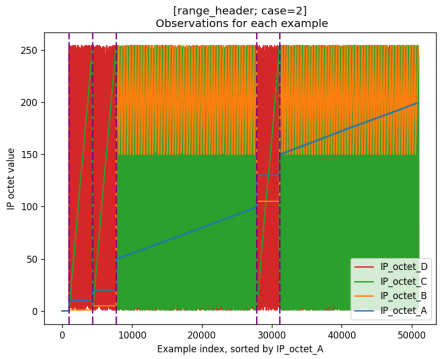

Figure 8: Input feature values for each evaluation example, plotted vs example index (sorted first by IP octet A, then octet B, etc). Each x-axis value corresponds to a single example, and the y-axis values indicate the input features for that example.

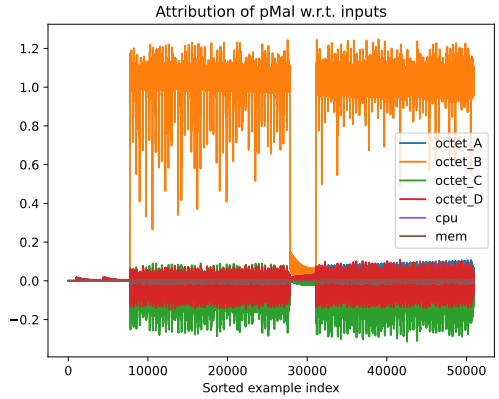

Figure 9: Attribution of p_{mal} with respect to each input feature, indicating how much each feature contributes to the model's decision. Each value on the x-axis corresponds to a single example (sorted first by IP octet A, then B, etc). For malicious examples, attribution due to octet B dominates, with some contributions coming from octets C and D. CPU and memory usage are not assigned significant attribution.

of sophisticated adversarial behavior, Molina-Markham et al. [18] presented the design of a network environment and software framework called FARLAND. CybORG is another RL-based toolkit for simulating and emulating autonomous cyber-defense operations [1]. While ElReLaND shares many high-level goals with FARLAND and CybORG, it differs in evaluating the effectiveness of RL-based SDN defense against three classes of network attacks, and in developing a framework for investigating the interpretability of the trained models. Mudgerikar et al. [21] proposed a security-constrained RL framework for protecting SDN that optimizes network performance and enforces security policies.

CyberBattleSim [2] provided an Open AI Gym simulation environment, with high-level abstraction of computer networks and vulnerabilities, for the training of automated RL-based cyber-defense agents. While the system is focused on simulating network topology and adversarial lateral movement, a limitation of the system is that no real network traffic is generated and no exploit code is executed.

Feng et al. [8] proposed an RL-based approach for mitigating application-level DDoS attacks. A main challenge pertaining to those attacks is that it may be difficult to accurately distinguish attack traffic from benign traffic. Their approach is based on Q-learning [38] and involves using a variety of environmental and contextual factors, including network and system load (e.g., CPU and memory, and I/O), and application-level behavior of clients (e.g., request frequency, size, and content). Our work is closely related to [8] with the following key differences: (1) [8] depends on the availability of ground truth to compute the reward functions in the training phase, whereas our reward function doesn't have such dependency; (2) [8] focuses on application-level DDoS attacks, and our work investigates a wider spectrum of network attacks, including range-header (an application-level DoS attack), SYN flood, and SSH brute-force; (3) [8] exclusively studies the use of Q-learning, while our work evaluates several RL algorithms.

5.2 Interpretability

Hinton and Frosst [11] distill the content of a neural network into an interpretable surrogate model, in the form of a soft decision tree. This is like the decision trees we use, except each branch is thresholded not on a single feature value, but on a linear combination of all features. Molnar [19] discusses the use of a decision tree as a more interpretable global surrogate model that can be used to understand the original black box model.

Miller et al. [3] use a soft decision tree as a global surrogate in an RL context. However, because we find the policies in this work depend only on the characteristics of an individual example (in this case, the IP address of a particular flow) rather than more aggregate characteristics (such as CPU and memory usage), there is no need to use the additional machinery that [3] develops to handle a truly state dependent system.

Feng et al. [7] directly trains an explainable RL model, using k-nearest neighbors (KNN) clustering in conjunction with a k-dimensional tree rather than a neural network model. Decisions made by their model can be explained by examining a particular input's distance from the nearest clusters. We make a few notes: (1) [7]'s approach loses the increased flexibility that comes with a neural network policy model, and possibly an RL-based method's ability to respond to novel attacks. (2) This method of comparing input features' distance from some reference features (in this case, the center of a cluster) is more likely to be useful for aggregate features such as the number of successful/unsuccessful SYN connections made by each IP address or average packet size sent and received by a given IP address, rather than for flow-specific features like the IP octets themselves.

While training a global surrogate model such as a decision tree can be useful, we have a few comments.

First, though the surrogate model is more interpretable by construction, there's a tradeoff between complexity and faithfulness to the original model. In particular, a surrogate model that is sufficiently complex to give a useful description of the model's behavior is often too complex to easily interpret. Second, in our case the model makes its decisions based solely on a handful of features – the four IP octets. We find that by directly analyzing the decisions the model makes as a function of this low-dimensional input space, typically through plots, we are often able to better interpret the model's behavior than by attempting to interpret a global surrogate.

Ribeiro et al. [29] propose the Local Interpretable Model-Agnostic Explanations (LIME) framework to explain a black box model by training not a single (i.e., global) surrogate model, but rather by training a surrogate model that explains the local behavior of the network around an example input. This allows each surrogate model to be extremely simple, and hence easily interpretable. The tradeoff is that a single local explanation is able to explain only the simplest models, so one needs to choose multiple neighborhoods, such that the original model's input space can be explained on more and more of its input space. The limiting factor then becomes how many such local surrogate models a user is willing to consider, how to choose exactly which neighborhoods to locally explain, how one is able to synthesize the local behaviors into a coherent picture, and how confident one can be that "enough" of the model's full behavior is considered. We note also that Dieber and Kirrane [6] provide an assessment of the LIME framework's effectiveness several years after LIME was first introduced. We leave it to future work to determine whether local surrogate models serve as a useful alternative to our global surrogates.

Sundararajan et al. [37] introduce an axiomatic construction for computing attribution; i.e., quantifying the contribution each of a model's input features had on that model's output prediction for a particular input example. While we found attribution analysis to be a good way of confirming that models largely ignored the CPU and memory usage in favor of the four IP octets, we generally found direct interpretability methods (such as input space decision and confidence analyses) and training an interpretable surrogate decision tree model to be more useful methods of explaining model behavior in this domain.

6 Conclusion

We have described the design and evaluation of a new framework for SDN-RL-based network defense evaluation called ElReLaND. We evaluated the system using three exemplar network attacks: SYN flood, range-header, and SSH brute-force, using both policy-based and off-policy RL algorithms. In our experiments, we found PPO was the best performing algorithm, and PPO was successfully able to generate defenses against all three attacks. An important component of ElReLaND is the ability to introspect the models generated by the system. To that end, we described four different analysis techniques that interpret the decision process captured by the generated models. In future work, we plan to expand the set of attacks and analysis techniques supported by ElReLaND, as well as conduct a deeper exploration of additional model-based and model-free RL algorithms.

References

[1] C. Baillie, M. Standen, J. Schwartz, M. Docking, D. Bowman, and J. Kim. Cyborg: An autonomous cyber operations research gym. *CoRR*, abs/2002.10667, 2020. https://arxiv.org/abs/2002.10667.

[2] W. Blum, J. Bono, and J. Parikh. Cyberbattlesim, 2020.

[3] Y. Coppens, K. Efthymiadis, T. Lenaerts, and A. Nowe. Distilling deep reinforcement learning policies in soft decision trees. In *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, pages 1–6, Aug. 2019.

[4] D. K. Dake, J. D. Gadze, G. S. Klogo, and H. Nunoo-Mensah. Traffic engineering in software-defined networks using reinforcement learning: A review. *International Journal of Advanced Computer Science and Applications*, 12(5), 2021. http://dx.doi.org/10.14569/IJACSA.2021.0120541.

[5] A. Das and P. Rad. Opportunities and challenges in explainable artificial intelligence (XAI): A survey. *CoRR*, abs/2006.11371, 2020. https://arxiv.org/abs/2006.11371.

[6] J. Dieber and S. Kirrane. Why model why? assessing the strengths and limitations of LIME. *CoRR*, abs/2012.00093, 2020.

[7] Y. Feng and J. Li. Toward explainable and adaptable detection and classification of distributed denial-of-service attacks. In *Deployable Machine Learning for Security Defense*, pages 105–121. Springer International Publishing, 2020.

[8] Y. Feng, J. Li, and T. Nguyen. Application-layer ddos defense with reinforcement learning. *Proc. 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020.

[9] P. Göransson and C. Black. *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann, 2014.

[10] Y. Han, B. I. P. Rubinstein, T. Abraham, T. Alpcan, O. Y. de Vel, S. M. Erfani, D. Hubczenko, C. Leckie, and P. Montague. Reinforcement learning for autonomous defence in software-defined networking. *CoRR*, abs/1808.05770, 2018. http://arxiv.org/abs/1808.05770.

[11] G. Hinton and N. Frosst. Distilling a neural network into a soft decision tree. 2017.

[12] Javitz and Valdes. The NIDES statistical component: Description and justification. mar 1993. http://www.csl.sri.com/papers/statreport/.

[13] A. Khraisat and A. Alazab. A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. *Cybersecurity*, 4, 2021.

[14] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, and O. Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[15] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim. Applications of deep reinforcement learning in communications and networking: A survey. *CoRR*, abs/1810.07862, 2018. http://arxiv.org/abs/1810.07862.

[16] MITRE. Cve-2011-3192, 2011. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3192.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[18] A. Molina-Markham, C. Miniter, B. Powell, and A. Ridley. Network environment design for autonomous cyberdefense. 2021. https://arxiv.org/abs/2103.07583.

[19] C. Molnar. Interpretable machine learning, 2022.

[20] J. Mondloch. Medusa parallel network login auditor, 2016. https://github.com/jmk-foofus/medusa.

[21] A. Mudgerikar, E. Bertino, J. Lobo, and D. Verma. A security-constrained reinforcement learning framework for software defined networks. *Proc. 2021 IEEE International Conference on Communications (ICC)*, 2021.

[22] A. Networks. Arbor ddos mitigation. https://https://www.netscout.com/arbor-ddos.

[23] G. A. C.-I. of Robotics and Mechatronics. Stable baselines3, 2022. https://github.com/DLR-RM/stable-baselines3.

[24] OpenAI. Kinds of RL algorithms, 2018. `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`.

[25] OpenAI. Gym, 2022. `https://github.com/openai/gym`.

[26] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, 1998.

[27] M. Peuster. Containernet, 2022. https://containernet.github.io.

[28] P. A. Porras and P. G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *1997 National Information Systems Security Conference*, oct 1997. http://www.csl.sri.com/papers/emerald-niss97/.

[29] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[30] Scapy. Packet crafting for python2 and python3, 2022. https://scapy.net/.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[32] F. SDN. Ryu, 2022. https://ryu-sdn.org.

[33] J. Smith. Extreme ddos defense. https://www.darpa.mil/program/extreme-ddos-defense.

[34] C. Software Engineering Institute. 1996 CERT Advisories, 1996. `https://resources.sei.cmu.edu/asset_files/WhitePaper/1996_019_001_496172.pdf`.

[35] J. Sommers. Harpoon: A flow-level traffic generator, 2022. https://jsommers.github.io/harpoon/.

[36] M. Stone. What is 5g security? explaining the security benefits and vulnerabilities of 5g architecture. https://cybersecurity.att.com/blogs/security-essentials/what-is-5g-security.

[37] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks, 2017.

[38] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[39] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. https://doi.org/10.1007/BF00992698.