AutoDefense: Reinforcement Learning Based Autoreactive Defense Against Network Attacks

Yu Mi
Case Western Reserve University
yu.mi@case.edu

David Mohaisen
University of Central Florida
david.mohaisen@ucf.edu

An Wang
Case Western Reserve University
an.wang@case.edu

***Abstract*—Attributed to the programmability and visibility provided by Software Defined Network (SDN) technologies, more flexible and complex functions can be performed on network attacks. However, identifying the attack traffic accurately for attack mitigation is a challenge. Most existing solutions leverage traffic characteristics to achieve this goal. Recent attacks characteristics have become more complex and indistinguishable from legitimate traffic. In this paper, we propose *AutoDefense*, a novel framework that leverages reinforcement learning techniques to deploy defense policies dynamically and adaptively based on the signals collected from the data plane. While we seek to achieve the same goal with the existing efforts where the network/server resources the attackers control should be limited, we allow more legitimate flows to enter the network, rather than relinquish bandwidth when attacks happen. Through evaluations, we demonstrate that *AutoDefense* could reduce 39% of the attack traffic and allow 48.6% more legitimate flows in the network. *AutoDefense* also improves the average flow completion time by 42.7% for the flows with a long tail latency.**

I Introduction

Distributed denial of service (DDoS) attacks have been the most persistent and lethal threats on the Internet, where they have recently been weaponized to cause significant damages to the Internet infrastructure [36], [44]–[46]. In particular, flooding attacks remain dominant, where they are simple to launch yet quite difficult to mitigate. Recent studies have identified trends that suggest the more sophisticated strategies and increasingly comprehensive targets that exacerbate this threat landscape [19], [48], [49].

To address this gap, the prior works attempt to distinguish legitimate and attack traffic in order to block the attack traffic. Such approaches could be broadly categorized as *detect-and-block* schemes. However, the existing detection mechanisms are mostly performed in a passive way by relying on the characteristics of the captured attack traffic. To improve the detection accuracy and limit the resources an attacker can occupy, we adopt a proactive tactic instead: We opportunistically increase the available bandwidth for certain flows in the hope that legitimate flows would grab more bandwidth than the attack flows. The key insight is the following: we hypothesize that the attackers try to overwhelm the target link bandwidth by issuing spoofed traffic, so they cannot respond to any bandwidth changes by generating reactive signals. On the other hand, legitimate (TCP)[1] flows could proactively take the spare bandwidth and send higher volumes of traffic, leading to potential behavior changes. We leverage the changes to distinct legitimate flows, and encourage more fractions of legitimate flows to arrive at the server by isolating them from the rest. The advantages of this approach are twofold. First, the domain knowledge of the attack traffic; e.g., attack model, is not required in this approach, since the efforts are focused on the legitimate TCP traffic. Second, this approach allows the system to react faster to attacks by replacing the anomaly detection with a temporary rate limiting.

Such a defense mechanism requires managing bandwidth flexibly and adaptively. Recent advances of SDN technologies and programmable network devices enable opportunities to achieve this goal. Nonetheless, designing an efficient framework is still a challenge for three reasons. ❶ The eventual goal of our framework is to opportunistically allocate more bandwidth to legitimate flows. To increase this opportunity, we need to narrow down the target flows from which we make selections. However, it is a challenging task since we do not have any *apriori* knowledge on the legitimate flows. ❷ With the recent advances in machine learning techniques, it has become tremendously popular among system operators to deploy effective and efficient defenses. While most existing efforts employ supervised or semi-supervised learning approaches to classify traffic, labeled data is hardly available in practice. Moreover, recent research works examined the robustness of such classification-based DDoS detection algorithms, and demonstrated that adversarial flows could be generated to decrease the detection rate to below 50% [6]. As a result, conventional end-to-end machine learning algorithms that generate packets labels do not fit our needs—for this reason, we propose to leverage a reinforcement learning (RL) based approach. ❸ It is challenging to design a model that allows RL to translate network events to effective classification rules that identify and isolate legitimate flows. RL has been utilized in the context of routing optimization, which is pioneered by Boyan *et al.* [13]. Recent attempts also apply RL techniques for traffic optimization [16]. Our problem is different in that we attempt to find a set of classification rules optimized for each individual device.

To fill this important gap, we propose an RL-based approach to generate clusters over packets sampled from the data plane, and derive optimal classification rules for legitimate traffic. The training process of RL, where an agent repeatedly interacts with the environment, addresses the limitations of the current operation modes of SDN by allowing the control plane to make intelligent decisions quickly with the properly defined set of actions. Such interactions could be triggered

[1]In this study, we focus on the legitimate TCP flows since their congestion control may further constrain the resources obtained by the legitimate flows under attack. We, however, do not limit the types of attack traffic.

automatically by events, such as attacks that are detected based on system metrics in the data plane. We call such a mode the *autoreactive* SDN mode. To narrow down the target flows, we also design and implement measurement schemes to identify flows that are most likely to be legitimate.

Several works proposed (supervised) decision-tree based solutions for packet classification [11], [20], [41]. However, our work is motivated by the need for an unsupervised learning algorithm, with a limited to no controller prior knowledge. As a result, we adopt a *decision-tree based clustering* technique, called *CLTree*, in our design. *CLTree* was initially proposed by Liu *et al.* to partition a data space into clusters by introducing virtual data points [28]. Given the clusters, we are able to formulate the optimization goal of our RL model as finding the set of packet clusters from which we generate classification rules that isolate the legitimate traffic to minimize the congestion signals in the data plane. We also employ various compact data structures, such as *bloom filters* and hash tables, to collect the signals required for the operation of our framework.

Main contributions. Our main contributions are as follows. ❶ We demonstrate the change of congestion signals upon attacks in the data plane through an empirical study (Section II). ❷ We present a learning based approach for identification of legitimate traffic based on data plane signals (Section III). ❸ We present a prototype that is implemented on Open vSwitch [39] and Pytorch [38] (Section III-D) to realize our approach. ❹ We presented a comprehensive evaluation of *AutoDefense*, and a demonstration that *AutoDefense* could effectively reduce the attack traffic by 39% and allow 48.6% more legitimate flows in the network, using application benchmark and synthetic traffic derived from real-world traces (Section IV).

II Motivation

II-A Temporal locality of traffic

Traffic locality is the phenomenon that is commonly seen in network systems and observed in various application scenarios, such as peer-to-peer (P2P) live streaming [29], content caching [43], and MapReduce task scheduling [47]. Benson *et al.* [12] conduct an empirical study of the network traffic in 10 data centers that belong to three different types of organizations, university, enterprise, and cloud data centers. One of the key findings in their study is that the number of active flows per second is under 10k per rack across all data centers. In a more recent study conducted by Hardegen *et al.* [22], they attempted to use DNN to predict the likely characteristics of real-world traffic flows. They collected network data from a real-world production network at Fulda University, and analyzed the characteristics of flows. In their analysis, they found that flow communications are active for a relatively short period of time, which further suggests that the average number of active flows per time unit should be small.

In a similar study by Avin *et al.* [10], they systematically analyze the structures featured by packet traces in networks. The packet traces used in this study come from three real-world traces and a few other synthetic traces. In this study, they introduce a metric, called *temporal complexity*, to represent the level of time dependency among packets. The lower the complexity, the stronger the time dependency there is. Their studies show that the data center workloads are skewed and bursty, featuring much temporal and spatial structure. The key takeaways of both studies suggest that the majority of traffic is carried by a relatively small subset of flows within a short period of time. Such findings reveal a promising opportunity to profile legitimate traffic within short time periods.

II-B Data plane signals upon attacks

Intuitively, TCP flows should react to the attack traffic due to the TCP congestion control mechanism. To further understand this reaction, we perform a measurement study in the network's data plane to observe the behaviors of legitimate flows when under attack. For this experiment, we interconnect three hosts with a software switch running Open vSwitch. One host runs Apache clients, another host runs the server, and the third host generates attack traffic to the server.

In this experiment, we assumed that both the legitimate users and the attackers send traffic to the target server. Moreover, we configured the legitimate users to run the Apache benchmark to request a file of 5 MB from the server, with 2000 concurrent requests. To simulate a realistic data center environment, we created 250 different IP addresses on the user machine using the aliasing feature and by configuring NIC aliases. The attack traffic is randomly generated using *hping3* [1] with spoofed IP addresses. We then collect the number of retransmitted packets that were generated by the legitimate users from the switch. We note that we enforce a rate limiting rule for all flows when the attack happens in this experiment. The results are shown in Figure 1.

In this figure, we use the x-axis to represent the elapsed time (in seconds) and the y-axis to show the number of the retransmitted packets per second. The grey area highlights the time span during which the attack is taking place. We also highlight the average number of packet retransmissions with two dash lines in the figure, which is 181 pps (packet-per-second) before the attack, and becomes 537.7 pps afterwards. As shown, this number is tripled upon the attack, which suggests that the attack traffic would cause legitimate flows to retransmit more packets. The enforced rate limiting rule further magnifies this effect. We also argue that a similar trend should also exist for the out-of-order packet deliveries. However, they are less likely to appear in our testbed, since the intermediate switch directly connects the source and the destination hosts.

We also analyzed two traces collected from real-world and synthetic attacks. The first trace contains a DNS amplification/reflection attack, staged by researchers between two sites (USC/ISI, Marina del Rey, California and CSU, Fort Collins, Colorado) [2] and additionally contains anonymized non-attack traffic. The second trace is the CSE-CIC-IDS2018 dataset [3], which contains synthetic attack traffic with seven different attack scenarios and legitimate traffic that simulates normal user behaviors. For both traces, we randomly select a segment that contains attack traffic and count the number of retransmissions for each flow. We compare the numbers for legitimate flows and malicious flows. The results are shown in Figure 2, where the x-axis represents the time in seconds, the y-axis shows the number of retransmissions per flow, and the grey areas highlight the time period when the attack took place. In Figure 2a, we can clearly observe that there are

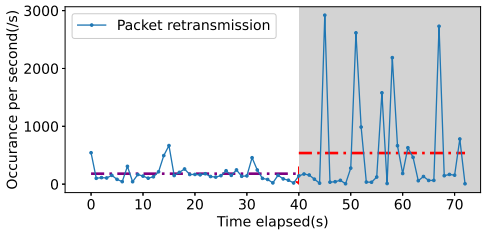
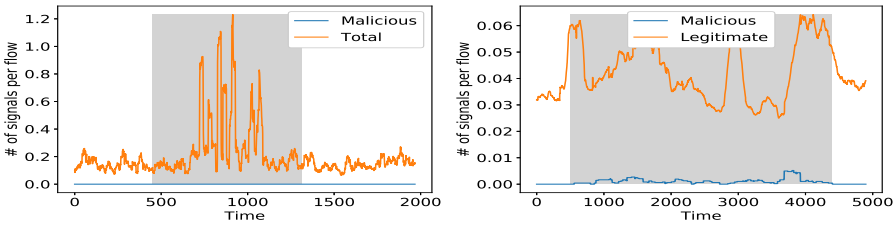


Fig. 1: The number of retransmitted packets before and after the attacks.



(a) DNS amplification attack. (b) LOIC HTTP attack.

Fig. 2: The number of retransmitted packets under various attacks.

significantly more packet retransmissions for legitimate flows during the attack and this number falls back to normal after the attack ends. Since the attack traffic does not contact packet retransmissions, we select the Low Orbit Ion Cannon (LOIC) HTTP flooding attack from the second trace and make the same comparisons. In Figure 2b, we can observe a similar trend with that in Figure 2a. In addition, we can also see that the number of packet retransmissions for the legitimate flows is much larger than that of the malicious flows. This observation provides important insights into the design of *AutoDefense* – that malicious flows are less likely to generate packet retransmissions during attacks.

II-C Learning-based clustering

Unsupervised clustering is an intuitive class of approaches and has been demonstrated as an effective direction for detecting intrusion attack traffic at a low cost [51], [52]. The key challenge is to label the generated clusters. Naturally, we can define certain metrics, such as cluster density, to distinguish different clusters based on predefined threshold values. However, it is difficult to determine good thresholds without any prior knowledge. Thus, we are motivated to apply a learning-based approach to tackle this hurdle.

III System Design and Implementation

III-A Overview of AutoDefense

AutoDefense aims to mitigate flooding attacks by observing the congestion signals when attacks happen in the network, which indicate that the legitimate flows may be adversely affected. Thus, *AutoDefense* collects packet samples from the data plane and generates mitigation policies based on these packets in the control plane. In order to generate the policies, we aggregate packets based on their source and destination IP prefixes by performing unsupervised clustering. To evaluate the effectiveness of the generated policies and adjust the clustering, we leverage reinforcement learning (RL) techniques to guide the partitioning and pruning of clusters. To provide feedback to the RL agent, *AutoDefense* also monitors flow statistics and activity signals in the data plane.

Figure 3 illustrates the architecture overview of *AutoDefense*. In the data plane, it consists of three main components: active flow monitoring, signalling packet recognition and metering/forwarding rules. The active flow monitoring is introduced to identify flows that have longer lifespans, which are more likely to be legitimate. The active flow monitoring helps guarantee that the collected samples contain some legitimate packets so that the RL model generates appropriate policies that allow these flows to get more bandwidth. The

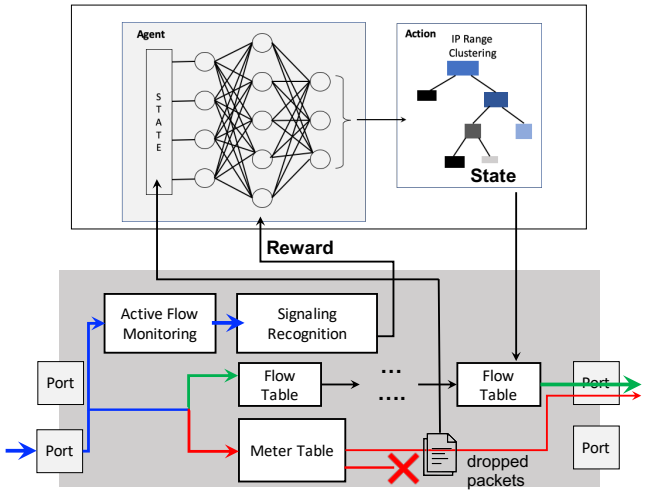


Fig. 3: Architecture of *AutoDefense*.

signalling recognition module detects congestion signals, such as out-of-delivery and packet retransmission, which often occur in legitimate flows when attacks happen. *AutoDefense* also employs the existing metering and forwarding tables in the switches to limit and forward traffic.

In the centralized controller, we train an RL model to build clustering trees based on the collected dropped packets. Although the growth of clustering trees is mainly driven by the sampled packets, the goal of the RL model is to compute a policy that maps the clusters to a pruning action that optimizes the rewards. Similar to many existing RL solutions [31]–[33], we also employ a neural network to implement the RL policies in our design. Finally, the model calculates rewards based on the results obtained from the signalling packet recognition module. The key idea is based on the insight that legitimate flows may generate more out-of-order deliveries and packet retransmission signals caused by congestion when attacks happen. Then the rewards are used to help the RL agent adjust the parameters, thus guiding the progress of the training. We discuss the details of each component in the rest of this section.

III-B Throttling attack traffic

When attacks happen, the switch may observe increasing packet drops at the egress queue due to congestion. As a reaction to this, the controller initializes rate limiting rules to control congestion in the networks. *AutoDefense* uses the meter table in the Open vSwitches to enforce the rate limiting policy. In our design, we redirect all the traffic to the same meter, thus flows competing for the limited number of tokens. Even though the rules are intended to prevent the attack traffic from reaching the target server, this does not help reduce collateral

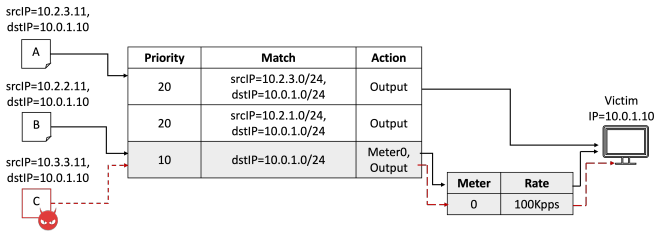


Fig. 4: Examples policies.

damage to the legitimate flows. That is, the packets would be dropped by the meter when there are not enough tokens. As a result, further actions need to be taken to protect the legitimate flows by electing them out.

An illustrative example of this idea is shown in Figure 4. In this example, the attack activity has been detected and two legitimate flows, A and B, and a malicious flow, C, are traversing the router. Initially, all three flows are processed by the default rate limiting rule of priority 10 in the forwarding table. Through active flow detection and collection (Section III-C) and clustering, a new forwarding rule for flow A is generated and inserted into the forwarding table. But flow B, which arrives later—thus not captured by the active flow detector—remains constrained by the rate limiting rule. Eventually, we attempt to generate forwarding rules for all the legitimate flows while confining the malicious flows to limited bandwidth with the rate limiting rule.

III-B1 Resuscitating legitimate flows.

The goal of this step is to identify and isolate the legitimate flows from the rate-limited path. To this end, *AutoDefense* collects the packets dropped by the metering rules and derives profiles of the legitimate traffic from them. Unlike most existing efforts that perform classifications based on the features extracted from the packets or flows, we use the real-time activity signals for identification instead. Such a design is based on our finding (as shown in Figure 1) that retransmitted packets would spike for legitimate traffic when congestion continues. The key challenge, however, is to monitor and report the real-time signals efficiently and effectively given the amount of traffic in the data center network systems. To address this challenge, we design and implement a filtering mechanism that helps focus the efforts on active flows, which are more likely to be legitimate. The filtering mechanism is discussed in detail in Section III-C.

On the controller side, *AutoDefense* leverages an RL model to group the dropped packets into clusters. Without any knowledge on the traffic patterns, we use a clustering via decision trees [28] approach, also known as *CLTree*, which aims to find the intrinsic structure of the packets. The growth of the tree is mainly dictated by the *information gain* criterion [7]. Each node in the tree represents an area in the IPv4 address space that is scoped by the ranges of source and destination IP addresses. To avoid the substantial overhead caused by frequent rule update in the data plane, we rely on a soft timer, *i.e., idle timeout*, that is associated with each rule, to eliminate the expired ones. Given the clusters, the RL agent then needs to distinct the 'good' clusters from the 'bad' ones. The RL agent performs tree pruning to implement this action.

To improve the efficiency, the built *CLTree* is updated once per episode, which contains five training steps. The packet samples that are collected in each step will be classified into one of the existing clusters. To extract useful information from the *CLTree*, we leverage tree pruning to identify the most probable legitimate clusters. Pruning is a data compression technique that reduces the size of the tree by eliminating the nodes that do not satisfy a particular criterion. Pruning generally requires human intervention based on their knowledge and experience to achieve the optimal size of the tree. Unfortunately, this requires additional human efforts and expertise. However, this challenge provides an opportunity for us to leverage the reinforcement learning techniques to learn the pruning parameters, which are discussed in detail next.

III-B2 What to learn for optimization

We note that recent studies have revealed a new trend of botnet-as-a-service (BaaS) that could be exploited by attackers to launch DDoS attacks [15], [23]–[25]. Such a trend will potentially lead the attackers to a wider footprint across the Internet. Moreover, the attackers are more likely to utilize the purchased bots to send attack traffic *alternatively* in order to avoid detection. As a result, the attack flows are more likely to come from dispersed sources with shorter duration. Based on that, we identify two features of the clusters that the RL model could utilize when making pruning decisions:

❶ **Number of instances in a cluster**: This feature reflects the duration of attack flows. The longer a flow is, the more packets it contains, thus more likely to be collected to the control plane, and vice versa. By the same token, legitimate flows will more likely fall in larger clusters. Our RL agent is designed to find the threshold of this parameter, which we call *min Y*, to eliminate the "bad" clusters and reduce the number of generated rules.

❷ **Instance density in a cluster**: This feature measures the level of flow dispersion. Using the number of instances alone could not optimize the pruning because a significant number of attack packets could be grouped into a single cluster. This is due to the fact that *CLTree* generates clusters based on *information gain*, which could be lower than a predefined threshold when a cluster contains a group of unrelated elements. For this reason, our RL agent also considers the instance density, and seeks to find its threshold that optimizes pruning. We call this threshold the *min ID*.

Given a node N_i from the clustering tree, assume that there are J samples; i.e., s_j for $0 < j \leq J$, in the node that fall in particular source and destination IP ranges. The IP range is defined as: $range_t(N_i) = [\arg\min(s_j.t), \arg\max(s_j.t)]$, $t \in \{srcIP, dstIP\}, j \in [1, 2, ..., J]$ Thus, the instance density, *i.e.,* ID, of a node N_i is calculated as: $ID(N_i) = J/\max(\arg\max(|range_t(N_i)|), 1)), t \in \{srcIP, dstIP\}$, which is the ratio between the number of nodes and the maximum range length. Both *min Y* and *min ID* are used in our pruning algorithm. In the algorithm, all the nodes whose instance number and density are lower than certain thresholds, *i.e., min Y* and *min ID*, and their subtrees are pruned. In addition to the two parameters learnt by the RL model, we also set an upper bound for the number of instances a cluster could have for it to be pruned, *i.e., max Y*. The purpose of this

value is to prevent larger clusters that contain both attack and legitimate packets from being pruned. Most of such clusters exist in the higher branches on the tree.

To determine *max Y*, we use the false positive rate of the active flow monitoring module in our system. A major component of this module is a bloom filter that helps filter out packets that have not been seen before. Given that the false positive rate of a bloom filter p could be calculated as a function of the number of elements n in the filter, the filter size m and the number of hash functions k, p is calculated as 0.03 in our system [34]. Thus, *max Y* is set to be 3%, which means that a cluster is likely to contain legitimate packets if it has more than 3% of the collected packets. More details of this module are discussed in Section III-C. In the following section, we discuss how our RL model learns the optimal parameters to generate the policies.

III-B3 How to learn for optimization

Finding the optimal pruning parameters to generate the policies could be challenging due to the dynamics of network traffic. Legitimate flows may be subject to change in terms of their volumes and sources constantly. In addition, the amount of attack packets that are collected by the switches within the data plane varies due to the false positives introduced by the active flow monitoring module. Both factors determine that the optimal pruning parameters are not constant values over time, motivating us to adjust them dynamically. An intuitive solution is to perform a grid search over possible parameters to identify the optimal ones. However, it may take long to perform an exhaustive search online while attacks are happening. Thus, we leverage the RL techniques to tune the parameters.

AutoDefense adopts Deep Q-Networks (DQN) [35] to approximate the mapping from states to actions. The states are observed from the environment and taken as input to the agent to generate an action. Since the goal of our RL model is to find the optimal parameters to prune the tree, the state of the clustering tree is taken as the environment that the agent interacts with. The states are represented as fixed-size vectors that contain five features of the clustering tree, including the current pruning parameters, the number of collected dropped packets, the number of rules that we currently have, and the average number of packets that the clusters contain. Each episode during the learning process represents a sequence of actions to adjust the pruning parameters. Specifically, the RL agent performs five pruning actions with the obtained parameters over the same clustering tree independently within each episode. The tree will be reset before starting each pruning action. Each pair of new parameters will be updated based on the values obtained from the previous time-step within each episode. In this manner, the agent has the opportunity to explore the parameters ranges exhaustively.

Actions. The agent is designed to increase or decrease the pruning parameters by a factor of a, where a represents an exponent. Given that we do not have any prior knowledge on the possible parameter range, we attempt to emulate exponential search in our system. The agent performs the actions to adjust both parameters, *i.e., min Y* and *min ID*, at the same time according to the following equations: $min\ Y_{t+1} = min\ Y_t \times 2^{a_{t1}}, min\ ID_{t+1} = min\ ID_t \times 2^{a_{t2}}$. Thus, the parameters will be either increased by twice or reduced by half depending on the actions. where $a_{t1}, a_{t2} \in \{-2, -1, 0, 1, 2\}$ and selected according to the actions taken by the agent.

Rewards.: In our design, we emphasize the rewards should reflect the effectiveness of the mitigation rules deployed in the data plane. As mentioned in § II, the number of out-of-order deliveries and retransmissions are key indicators of network congestion. Therefore, the reward is calculated as:

$$r_t = \min(T_e - (C_r + C_o)/\max(C_a, 1), 0) \quad (1)$$

where C_r and C_o represent the number of retransmitted packets and out-of-order packets collected in the data plane, respectively. C_a indicates the number of active flows in the data plane. The purpose of calculating the ratio between the number of signals and the active flows is to avoid rewarding the situation when the active flows are strangled by the attack traffic. When that happens, the absolute number of the signals could be low, while the number of active flows remain high. Thus, we calculate the average number of signals per active flow instead. In Equation (1), T_e represents a tolerance factor that represents the acceptable number of signals that could occur in a network without interrupting applications. Such a factor could be derived from empirical measurements of a network system. We set this value to be 0.03 as discussed in Section III-B2. The training process of the model is mainly driven by the flow statistics and signals we collect from the network data plane. We discuss the monitoring and collecting modules of *AutoDefense* in the following sections.

III-C Data plane monitoring

An essential step of *AutoDefense* is to monitor and collect packets from the data plane for training the RL model. However, it is a challenging task to monitor every single flow in the data plane, especially with the limited computation and memory resources of switches. On the other hand, it is unnecessary to collect packets from all flows to enable effective and efficient learning in our system. Since *AutoDefense* is tasked with generating profiles of the legitimate flows, we only collect packets that belong to active flows in the data plane. Next, we discuss how *AutoDefense* detects the active flows.

III-C1 Active flow collection

Detection. An important factor to consider when determining whether a flow is active or not is its occurrence frequency. Given a fixed time period, frequency could be captured by flow volume or packet counts. Therefore, we design and implement schemes to detect active flows in the switches. Many schemes have been proposed for this purpose on different platforms (see Section VI). To strike a balance between efficiency and accuracy, we adopt part of the detection scheme of Elastic Sketch [50], which is inspired by Ostracism—which is a procedure under Athenian democracy in which any citizen could be expelled through voting. This scheme consists of two parts: a "heavy" part for elephant flows and a "light" part for mouse flows. The "heavy" part of the scheme employs a hash table and the number of positive and negative collisions (votes) to identify heavy hitters. We introduce this table to our active flow detection module and extend it by including an additional filtering mechanism to narrow down our target flows.

Given the non-negligible chance that mice flows may prevent more active flows from accumulating vote^+ due to the collective volume and possible collisions, we add another layer of filter to prevent mice flows from passing through and reaching the hash table. We use a bloom filter for this purpose. Only packets that belong to an existing flow are allowed to increment the votes in the hash table. In practice, this could also be replaced by a counting bloom filter to enable a higher threshold for the mice flows. To maintain the entries and counts in the hash table, we periodically decrease all vote^+ and vote^{all} by half so that the new flows have a chance to survive in the table. Similarly, we clear the entire bloom filter bitmap every 40k packets. In addition to the packet-level statistics, we also maintain flow-level statistics; e.g., the timestamp of the last acknowledged packet in a flow, in the hash table for detecting flow signals; e.g., packet retransmissions and out-of-order deliveries.

Collection. Once active flows are identified, we further sample packets from the active flows that suffer from the attacks. Recall that the controller initializes rate limiting rules to throttle attack traffic upon the detection of the attacks, as mentioned in Section III-B. As a result, both legitimate and attack packets could be dropped by the same meter. Among the dropped packets, we associate them with the active flows previously detected and sample them with a fixed-size ring buffer data structure in the switch. When the ring buffer is full, the switch notifies the controller and forwards the collected samples, which will be used to generate clustering trees.

III-D Implementation

We implement *AutoDefense* over the Open vSwitch [39] and Ryu [17] controller platform. Note that the proposed scheme can be easily implemented over programmable switches as well, with very minimal modifications. The RL model is implemented based on the OpenAI Gym [14] with Pytorch [38]. For the data plane implementations, we instrument the Open vSwitch and extend the standard OpenFlow protocols so that we could perform additional operations. For the Open vSwitch, we implement some additional data structures to fulfill the monitoring purposes, such as the bloom filters and hash tables. To pair the entries in the hash table, we adopt the linear combination of two hash functions to generate more hash results as introduced by Kirsch *et al.* [27]. We also extend the Netlink messages to support the new control messages. Shared memory is used for transmitting packet samples from kernel to userspace.

IV Evaluation

IV-A Evaluation setup

AutoDefense is evaluated on the Cloudlab platform [40] with application benchmark, synthetic data generated by hping3, UFONet [4] and real-world traffic patterns. The evaluations are performed in a 10 Gbps network with bare-metal machines that run Ubuntu 16.04. Each host used in our experiment contains two Intel E5-2630v3 8-core CPUs, 64GB RAM, and Intel X710 NICs.

The setup is the same with that of the one in our motivation experiment (as described in Section II). Three machines are interconnected with each other by a host that runs Open vSwitch. They are running as clients, attackers, and the target server, respectively. In our evaluation, both the clients and the attackers generate traffic, and attempt to reach the target server at the same time. For legitimate traffic, we run the Apache benchmark over multiple simulated IP addresses (250 by default) by using NIC aliases. The clients are configured to download files of diverse sizes (5 MB by default) with different concurrency levels (2000 by default).

For synthetic attacks, we use *hping3* to send the attack traffic at the rate of 500 kpps. We also use a specialized DDoS traffic generating tool, *UFONet*, to generate the attack traffic in our evaluations. For the mimicry attacks, we configure a traffic generation tool, *Harpoon* [42], following the traffic patterns extracted from the WC'98 dataset [9]. *Harpoon* is a flow-level traffic simulation tool and could generate valid signals for the flows it simulated. The WC'98 dataset is an access record from the world cup official website in 1998, where the visitors issue random access. For the following experiments, the initial rate limiting meter in the *Open vSwitch* is set to 100 kpps. All evaluations demonstrate the average results of 3 experiments.

We answer three questions by our evaluations. ❶ What is the performance of the learning module in the control plane? ❷ How well does *AutoDefense*, taken as a whole, perform in defending against attack traffic? ❸ How accurate and effective will *AutoDefense* be when operating under mimicry attacks? To answer these questions, we evaluate *AutoDefense* from four aspects. We discuss each of them in the following.

IV-B Performance of the learning module

A major metric to evaluate the performance of the RL model is the coverage of the classification rules derived from the generated clusters. In other words, how many legitimate flows could be covered by the classification rules? The coverage is evaluated over two groups: the coverage of the newly generated rules at the end of each training epoch, and that of the rules remaining in the switch tables. For the newly generated rules, we use the packets sampled from the data plane for coverage evaluation. Recall that we use a sliding window scheme to pool the sampled packets across multiple epochs together for training. Thus, the packet pools are used in this evaluation. Whereas for the rules in the switch tables, we use all the legitimate flows in the dataset for the calculation. The results are shown in Figure 5

In this figure, $\text{Accuracy}_{\text{pool}}$ represents the coverage of the newly generated rules, and $\text{Accuracy}_{\text{rule}}$ is the coverage of the rules in the switch tables. The red arrow indicates the start of the attack traffic. By observing the trend of $\text{Accuracy}_{\text{pool}}$, we can easily see that the overall coverage of the newly generated rules is high, which is attributed to the high accuracy of the monitoring modules in the data plane. On the other hand, $\text{Accuracy}_{\text{pool}}$ also steadily increases as the learning progresses. The maximum coverage of $\text{Accuracy}_{\text{pool}}$ is about 90%.

By comparing these two curves, we can observe that $\text{Accuracy}_{\text{rule}}$ stabilizes at a higher coverage than $\text{Accuracy}_{\text{pool}}$, which could be up to 100%. The reason is that there exists a soft timer with each rule in the switch tables. So that legitimate flows could live longer in the data plane until they expire. To address the potential concerns with the limited table size,

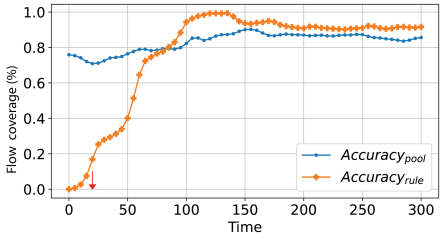


Fig. 5: Coverage of classification rules.

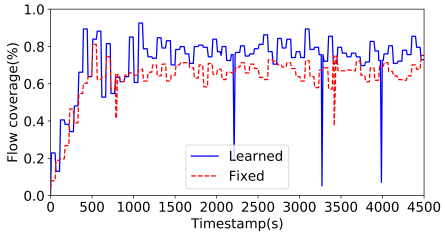


Fig. 6: Learning vs. fixed threshold pruning.

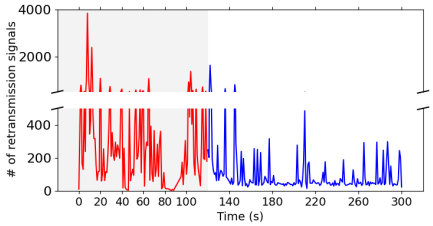


Fig. 7: Signal changes in the data plane.

we also dump the number of rules from the switch. While it fluctuates over time, the maximum number of rules we observe is 190 rules, which could hardly overflow the switch tables.

We also demonstrate the effectiveness of the proposed learning scheme by comparing its $\text{Accuracy}_{\text{rule}}$ with that of the scheme where fixed thresholds are used for pruning. The fixed thresholds are the median values of *min Y* and *min ID* obtained from the leaf nodes of the clustering trees. We can observe that $\text{Accuracy}_{\text{rule}}$ is higher when the learning-based scheme is utilized. Therefore, the learning-based pruning is more effective than using fixed thresholds.

IV-C Effectiveness of defense

Effictiveness of learning. We first discuss the improvement of learning over the collected samples. We compare the coverage policy based on learned pruning parameters against the fixed pruning parameters. The fixed pruning parameters are generated by calculating the median of minID and minY across all the clusters. As shown in Figure 6, the learned policies have the edge over the fixed parameters, proving that learning could improve the accuracy of policy generated.

Number of retransmitted packets. The goal of the proposed framework is to allocate more bandwidth to the legitimate flows, thus increasing the fraction of legitimate traffic that could reach the target server. As a result, we analyze the number of retransmitted packets that happen in the data plane and compare those with and without *AutoDefense*. The results are shown in Figure 7. In this figure, the x-axis represents time, while the y-axis shows the number of retransmitted packets per second. In this experiment, *AutoDefense* is disabled until the 120^{th} second (as highlighted in a grey shade). As such, we use distinct colors to highlight the different time spans. The average packet retransmission before the defense is enabled is 332.6 pps, while it becomes 98.6 pps afterwards. This indicates that the number of retransmitted packets drops significantly upon the deployment of *AutoDefense*.

Flow completion time (FCT). For the completeness of our evaluation, we also evaluate the effectiveness of *AutoDefense* from the application level. Typically, users want their flows to finish as quickly as possible. Thus, FCT is an important metric to evaluate user experience. In our experiment, each source IP address may issue multiple flows to download a file repeatedly from the Apache server since the IP addresses are scheduled to send requests in a round-robin fashion.

❶ **Synthetic Attacks.** In this experiment, we use *hping3* to generate random attack traffic and present the Cumulative Distribution Function (CDF) of the 50^{th} percentile and the 95^{th} percentile FCT across all users in Figure 8. In addition to the results in cases with and without the defense, we also have a baseline that represents the CDF of FCT without attack traffic. From these results, we conclude that *AutoDefense* improves the 50^{th} percentile FCT by 16.6% on average, and the 95^{th} percentile FCT by 42.7% on average. The disparity between *AutoDefense* and the baseline is due to the overhead and the false negatives in monitoring.

❷ **UFONet Attacks.** We leverage *UFONet* to launch a combination of various attacks, including HTTP-based flooding, TCP-based flooding, and UDP-based flooding attacks. We create 50 virtual IP addresses in total, any random 20 of which launch these attacks for 1,000 times per attack type in each round. The overall attack traffic rate is about 800 Mbps. In this experiment, the clients are configured to download files of diverse sizes, i.e., 500 KB, 5 MB and 50 MB, respectively. The results of the FCT distribution are shown in Figure 9. In this figure, we show the maximum FCT, the minimum FCT, and the median FCT values. The values for 500 KB file flows are shown on the left y-axis while those for the other two groups are shown on the right y-axis. We can observe that *AutoDefense* can effectively reduce the long tails of FCT values during attacks. The overall FCT values are slightly higher than those of the baseline scenario. The reason is that more flows can finish the benchmark tests in the case with *AutoDefense*. Moreover, we observe that *AutoDefense* works more efficiently for shorter flows, since they are more time sensitive.

IV-D Mimicry attack evaluation

Recent attacks show a trend of mimicry flash crowds that are hard to be detected by mitigation systems [26]. Thus, we also evaluate the performance of *AutoDefense* under mimicry attacks derived from the WC'98 dataset. The legitimate flows are generated based on the longer flows in this dataset. The pattern that Harpoon simulates includes the flow size distribution and the interval between each connection, where the simulation could mimic the normal user from both the temporal and spatial domain. Figure 10 shows the flow size distribution of mimicry and legitimate flows.

Active flow detection. In mimicry attacks, attackers aim to resemble the legitimate users. Therefore, the accuracy of active flow detection largely determines the performance of *AutoDefense*. We calculate statistics of the detection results, which are shown in Figure 11. We can see that the detection can achieve reasonable accuracy that enables effective learning and signals collection.

Legitimate flow rate. To evaluate the effectiveness of *AutoDefense* in defending against the mimicry attacks, we calculate the aggregated rate (in Mbps) of legitimate flows and compare

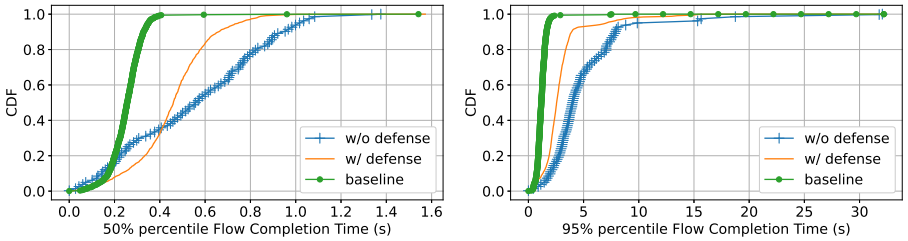


Fig. 8: Packet loss with different file sizes: 50^{th} (left) and 95^{th} percentile (right).

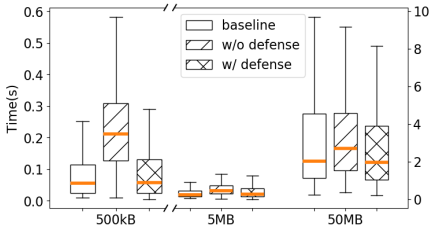


Fig. 9: FCT in different scenarios.

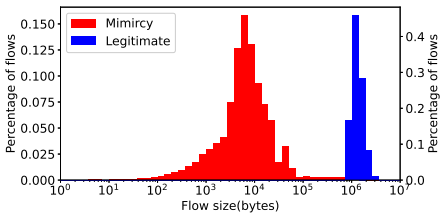


Fig. 10: Flow size distribution.

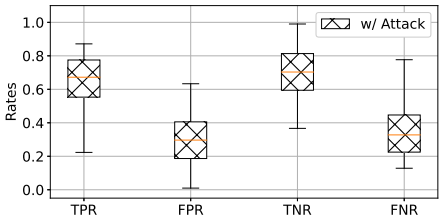


Fig. 11: Accuracy of active flow detection.

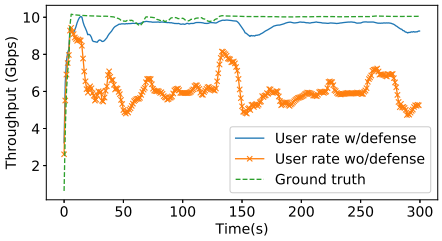


Fig. 12: Legitimate flow rate.

that with the case without the defense. The results are shown in Figure 12, where the attack is launched around the 15^{th} second, and the ground truth represents the legitimate flow rate without any attacks. We observe that *AutoDefense* manages to keep the legitimate flow rate close to the ground truth (line rate) at 10 Gbps, which otherwise would decrease by half without any defense. The average packet size of the legitimate flows is about 1300 bytes. Overall, *AutoDefense* mitigates the mimicry attacks effectively.

IV-E Comparison with MiddlePolice

In this study, we compare the performance of *AutoDefense* with that of the prior related work, *MiddlePolice* [30]. *MiddlePolice* protects the target from DDoS attacks by deploying middlebox functions in the network dataplane. A middlebox measures the total downstream loss rate between the middlebox and the target as well as the loss rate of each individual sender. Then, credits are assigned to each sender to minimize the downstream loss rate. When the credits are used up, and the downstream loss rate exceeds a predefined threshold, packets from this sender will be blocked.

We evaluate the kernel module of *MiddlePolice* by deploying it in our Top-of-Rack (ToR) switch, and the capability handling module (CHM) in the target hosts. However, to keep updating the downstream loss rate, *MiddlePolice* requires a few long-term connections; otherwise, the network will be fully throttled since the loss rate cannot be updated once it exceeds the threshold. In this experiment, we emulate 8 legitimate users and use *hping3* to generate *SYN* flooding traffic. We disable the source IP address spoofing in this experiment because random source traffic will not trigger the detection period of *MiddlePolice*.

Figure 13 compares the throughput of the legitimate flows with *AutoDefense* and *MiddlePolice*. In the figure, the $x-$axis shows the time and the $y-$axis shows the throughput in Mbps. We can observe that both *AutoDefense* and *MiddlePolice* can defend against such flooding attacks effectively. *AutoDefense* can achieve almost line rate, which is close to the ground truth flow rate. But *MiddlePolice* experiences nearly 30%

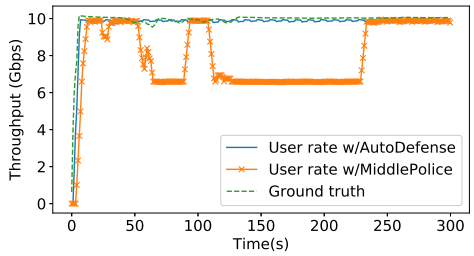


Fig. 13: *AutoDefense* vs. *MiddlePolice*.

bandwidth degradation when the downstream loss rate exceeds the predefined threshold because *MiddlePolice* does not admit more packets (by limiting credit allocation) in the following periods in this case. Such a credit allocation scheme makes *MiddlePolice* more effective when senders' traffic do not share bottleneck downstream links with the attack traffic.

V Discussion

AutoDefense is designed based on the detection of TCP packet retransmission signals. Thus, *AutoDefense* is effective in defending against flooding attacks, such as the TCP SYN flooding attacks and HTTP GET flooding attacks, although it may fail to detect attacks that do not cause network congestion, such as the infiltration attacks. Moreover, *AutoDefense* relies on the changes of the retransmission signals to guide the learning of RL agent so that it can better distinguish between legitimate flows and attack flows. As a result, attacks that are constructed to simulate legitimate user behaviors are more difficult to be distinguished by *AutoDefense* since they also generate the same packet retransmission signals.

VI Related Work

Most cyber threats that exist on the Internet today rarely stay the same for extended periods of time. As a result, there are many new threats that enterprise networks must deal with all the time. Machine learning-based techniques have been particularly efficient and effective in detecting such evolving threats [5], [8] by eliminating the complexity of conventional defense mechanisms that often require significant human intervention. Dishi *et al.* developed a machine learning-based framework to detect IoT DDoS attack traffic and find

that random forest, K-nearest neighbors, and neural network classifiers are especially effective [18]. A similar effort is conducted by Hamza *et al.*, in which they develop a framework to detect anomalous patterns of MUD-compliant network activities by performing three classification techniques and find that clustering-based approaches perform the best among others [21]. To tackle the high false alarm problem, Nguyen *et al.* design and implement a federated learning framework to train a distributed DNN that collects packets from multiple vantage points for anomaly detection [37]. In the defense space, Liu et al. [30] proposed *MiddlePolice* which deploys middlebox functions to protect targets against DDoS attacks through filtering.

VII Conclusion

DDoS has been a consistent and lethal threat to the Internet. Conventional defense solutions that leverage the end-to-end machine learning techniques to detect and mitigate against attack traffic may be unreliable. They also heavily rely on their prior knowledge of the attacks. To address these concerns, we propose a framework, called *AutoDefense*, that leverages an RL model to identify and isolate legitimate flows to allow more legitimate traffic into the system. To that end, we train a neural network to find the optimal policy that maps a set of sampled packets to a group of packet classification rules. Our evaluations show that *AutoDefense* could reduce 39% of the attack traffic and allow 49% more legitimate flows in the network. In the future, we will address issues that may arise due to model size, drift, and latency considerations.

VIII Acknowledgement

We would like to thank the reviewers of CNS'22 for their feedback. This work is supported in part by the NSF grants CNS-2008468 and a Google Faculty Research Award. Dr. Mohaisen is also supported in part by NRF-2016K1A1A2912757.

References

[1] Active network security tool. [Online]. Available: http://www.hping.org/
[2] "Usc/isi ant project," https://tinyurl.com/2p84v339, 2013.
[3] "IDS 2018 | Datasets," https://tinyurl.com/yfrb2vun, 2021.
[4] "UFONet – Denial of Service Toolkit," https://ufonet.03c8.net, 2021.
[5] A. Abusnaina *et al.*, "Adversarial learning attacks on graph-based iot malware detection systems," in *Proceedings of IEEE ICDCS*, 2019.
[6] ——, "Examining the robustness of learning-based ddos detection in software defined networks," in *Proceedings of IEEE DSC*, 2019.
[7] R. Agrawal *et al.*, "An interval classi er for database mining applications," in *Proceedings of the VLDB*, 1992.
[8] H. Alasmary *et al.*, "Soteria: Detecting adversarial examples in control flow graph-based malware classifiers," in *Proceedings of IEEE ICDCS*, 2020.
[9] M. Arlitt *et al.*, "1998 world cup web site access logs," 1998.
[10] C. Avin *et al.*, "On the complexity of traffic traces and implications," *Proceedings of ACM SIGMETRICS*, 2020.
[11] F. Baboescu *et al.*, "Scalable packet classification," *Proceedings of ACM CCR*, 2001.
[12] T. Benson *et al.*, "Network traffic characteristics of data centers in the wild," in *Proceedings of ACM IMC*, 2010.
[13] J. Boyan *et al.*, "Packet routing in dynamically changing networks: A reinforcement learning approach," *Adv. Neural Inf. Process. Syst.*, 1993.
[14] G. Brockman *et al.*, "Openai gym," *arXiv:1606.01540*, 2016.
[15] W. Chang *et al.*, "Characterizing botnets-as-a-service," in *Proceedings of ACM SIGCOMM*, 2014.
[16] L. Chen *et al.*, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of ACM SIGCOMM*, 2018.
[17] R. S. F. Community, "Ryu sdn framework," https://ryu-sdn.org/, 2017.
[18] R. Doshi *et al.*, "Machine learning ddos detection for consumer internet of things devices," in *Proceedings of IEEE SPW*, 2018.
[19] C. Fu *et al.*, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proceedings of ACM CCS*, 2021.
[20] P. Gupta *et al.*, "Packet classification using hierarchical intelligent cuttings," in *Hot Interconnects VII*, 1999.
[21] A. Hamza *et al.*, "Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity," in *Proceedings of ACM SOSR*, 2019.
[22] C. Hardegen *et al.*, "Predicting network flow characteristics using deep learning and real-world network traffic," *Proceedings of IEEE TNSM*, 2020.
[23] K. Huang *et al.*, "Systematically understanding the cyber attack business: A survey," *ACM CSUR*, 2018.
[24] ——, "Casting the dark web in a new light," *MIT SMR*, 2019.
[25] G. Kambourakis *et al.*, "The mirai botnet and the iot zombie armies," in *Proceedings of IEEE MILCOM*, 2017.
[26] B. A. Khalaf *et al.*, "An adaptive model for detection and prevention of ddos and flash crowd flooding attacks," in *ISAMSR*, 2018.
[27] A. Kirsch *et al.*, "Less hashing, same performance: building a better bloom filter," in *European Symposium on Algorithms*, 2006.
[28] B. Liu *et al.*, "Clustering via decision tree construction," in *Foundations and advances in data mining*. Springer, 2005.
[29] Y. Liu *et al.*, "A case study of traffic locality in internet p2p live streaming systems," in *Proceedings of IEEE ICDCS*, 2009.
[30] Z. Liu *et al.*, "Middlepolice: Toward enforcing destination-defined policies in the middle of the internet," in *Proceedings of ACM CCS*, 2016.
[31] H. Mao *et al.*, "Resource management with deep reinforcement learning," in *Proceedings of ACM HotNets*, 2016.
[32] ——, "Neural adaptive video streaming with pensieve," in *Proceedings of ACM SIGCOMM*, 2017.
[33] ——, "Learning scheduling algorithms for data processing clusters," in *Proceedings of ACM SIGCOMM*, 2019.
[34] M. Mitzenmacher *et al.*, *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*, 2017.
[35] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013.
[36] S. Moss, "Major ddos attack on dyn disrupts aws, twitter, spotify and more," https://tinyurl.com/2p8nctwf, 2016.
[37] T. D. Nguyen *et al.*, "Dïot: A federated self-learning anomaly detection system for iot," in *Proceedings of IEEE ICDCS*, 2019.
[38] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Adv. Neural Inf. Process. Syst.*, 2019.
[39] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *Proceedings of USENIX NSDI*, 2015.
[40] R. Ricci *et al.*, "Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications," *Magazine of USENIX login;*, 2014.
[41] S. Singh *et al.*, "Packet classification using multidimensional cutting," in *Proceedings of ACM SIGCOMM*, 2003.
[42] J. Sommers *et al.*, "Harpoon: a flow-level traffic generator for router and network tests," *Proceedings of ACM SIGMETRICS*, 2004.
[43] S. Traverso *et al.*, "Temporal locality in today's content caching: why it matters and how to model it," *Proceedings of ACM CCR*, 2013.
[44] A. Wang *et al.*, "Capturing ddos attack dynamics behind the scenes," in *Proceedings of DIMVA*, 2015.
[45] ——, "A data-driven study of ddos attacks and their dynamics," *Proceedings of IEEE TDSC*, 2018.
[46] ——, "Delving into internet ddos attacks by botnets: characterization and analysis," *Proceedings of IEEE/ACM ToN*, 2018.
[47] Q. Xie *et al.*, "Scheduling with multi-level data locality: Throughput and heavy-traffic optimality," in *Proceedings of IEEE INFOCOM*, 2016.
[48] J. Xing *et al.*, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *Proceedings of USENIX Security*, 2021.
[49] ——, "A vision for runtime programmable networks," in *Proceedings of ACM HotNets*, 2021.
[50] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of ACM SIGCOMM*, 2018.
[51] S. Zanero *et al.*, "Unsupervised learning techniques for an intrusion detection system," in *Proceedings of ACM SAC*, 2004.
[52] ——, "Unsupervised learning algorithms for intrusion detection," in *Proceedings of IEEE NOMS*, 2008.