Application-Layer DDoS Defense with Reinforcement Learning

Yebo Feng, Jun Li, Thanh Nguyen
Department of Computer and Information Science
University of Oregon
{yebof, lijun, thanhhng}@cs.uoregon.edu

***Abstract*—Application-layer distributed denial-of-service (L7 DDoS) attacks, by exploiting application-layer requests to overwhelm functions or components of victim servers, have become a rising major threat to today's Internet. However, because the traffic from an L7 DDoS attack appears legitimate in transport and network layers, it is difficult for traditional DDoS solutions to detect and defend against an L7 DDoS attack.**

In this paper, we propose a new, reinforcement-learning-based approach to L7 DDoS attack defense. We introduce a multi-objective reward function to guide a reinforcement learning agent to learn the most suitable action in mitigating L7 DDoS attacks. Consequently, while actively monitoring and analyzing the victim server, the agent can apply different strategies under different conditions to protect the victim: When an L7 DDoS attack is overwhelming, the agent will aggressively mitigate as many malicious requests as possible, thereby keeping the victim server functioning (even at the cost of sacrificing a small number of legitimate requests); otherwise, the agent will conservatively mitigate malicious requests instead, with a focus on minimizing collateral damage to legitimate requests. The evaluation shows that our approach can achieve minimal collateral damage when the L7 DDoS attack is tolerable and mitigate 98.73% of the malicious application messages when the victim is brought to its knees.

***Index Terms*—application-layer DDoS, distributed denial of service (DDoS), reinforcement learning, anomaly detection**

I. Introduction

Application-layer distributed denial of service attacks [1], or layer 7 (L7) DDoS attacks, represent a type of malicious behavior that attack the application layer in the network model. These L7 DDoS attacks exploit application-layer messages (e.g., web requests) to swamp specific application functions or components of a victim server (e.g., a web server) to disable or degrade their services, impacting legitimate users' experience.

L7 DDoS attacks are on the rise and becoming conspicuous threats on today's Internet. One of the best-known L7 DDoS attacks happened in March 2015 when a massive number of HTTP requests poured towards GitHub [2], causing much reduced availability and higher latency to GitHub's service. This attack worked by injecting nefarious JavaScript code pieces into numerous web pages to redirect a high volume of users' HTTP traffic to GitHub. More recently, Imperva reported a notable L7 DDoS attack [3] in July 2019. This attack was the longest and largest that Imperva has ever seen, lasting 13 days and reaching a peak volume of 292,000 requests per second.

978-1-7281-6887-6/20/$31.00 ©2020 IEEE

Unfortunately, the detection and defense of L7 DDoS attacks are still not well-studied [1], [4]. Worse, attackers continuously evolve their toolkits and develop more sophisticated L7 DDoS attack techniques. It is therefore compelling to accurately identify L7 DDoS attacks and generate effective mitigation tactics against them.

The key to addressing L7 DDoS attacks is to distinguish L7 DDoS traffic from the legitimate application-layer traffic. This task is difficult, however, given that an L7 DDoS attacker can purposely fabricate application-layer messages that look legitimate, as discussed above. An L7 DDoS message can even be identical to a legitimate application-layer message.

Interestingly, the legitimacy of an application-layer message is heavily dependent on its environment or context. The same application-layer message may be legitimate in one environment, but totally malicious in another. Or similarly, depending on how a client has been interacting with a server in the past, a newly received request from the client may be legitimate in one case, but illegitimate in another. For example, an HTTP GET message is totally legitimate during the routine operation of an HTTP server but could be malicious during an HTTP flooding attack. In another words, it is a Markov decision process to determine whether an application-layer message is legitimate or not.

We thus seek to discover what methodologies would be the most effective in distinguishing L7 DDoS traffic from the legitimate application-layer traffic by considering environmental and contextual factors, instead of only inspecting the messages themselves. This paper proposes the first *reinforcement-learning-based* method that incorporates environmental and contextual factors to distinguish L7 DDoS traffic from the legitimate application-layer traffic. It monitors and analyzes a variety of environmental and contextual factors including those related to the system and network load of the victim server (e.g., disk I/O , CPU operation, memory usage, or link utilization) and the dynamic application-layer behaviors of clients (e.g., request type, size, frequency, and content).

Furthermore, this method streamlines the L7 DDoS defense by integrating the operations of attack detection, message classification, and attack mitigation. Rather than producing labels of each application-layer message for a separate L7 DDoS mitigation module to handle the message, in order to mitigate L7 DDoS attacks, this method directly outputs the action to take for each application message under different

circumstances. Actions can include blocking the message upstream, blocking it locally, or postponing its processing.

In addition, this method receives feedback from the actions taken, allowing it to fine-tune what actions are the best for a given situation. With the design of a new multi-objective reward function, this method can determine the most suitable actions to take in a way that (1) minimizes the amount of discarded legitimate messages to provide the service as much as possible to clients when the victim load is low and (2) maximizes the amount of filtered L7 DDoS messages to prevent the server from collapse when the victim load is high.

The evaluation shows that this approach can identify the majority of DDoS traffic and significantly increase the capacity of the victim server. At the peak of L7 DDoS attacks its accuracy is 0.9553 and true positive rate is 0.9873, while when the attacks are not overwhelming the collateral damage is as low as 0. The implementation of this method, while not intricate, provides satisfying performance when running on the server node. With less than 30,000 training episodes, this method can easily adapt to an unacquainted victim server environment.

II. Related Work

The current defense models against L7 DDoS primarily follow a two-phase procedure, which performs *detection & classification* to identify the malicious sources or application messages and then *mitigates* the attack by conducting access control. These models treat the two phases as two separate modules, making it difficult to modulate mitigation strategies according to the conditions of specific attacks. On the contrary, our approach considers attack classification and mitigation as an integral whole to pursue the best L7 DDoS defense efficacy. Below we detail the related work in each phase.

A. Detection & Classification Approaches

We categorize the previous approaches to L7 DDoS detection & classification into three types: *statistical methods*, *learning-based methods*, and *Markov-based methods*.

1) Statistical methods: Researchers build statistical models on both benign and malicious L7 DDoS traffic and then apply them to detect and classify L7 DDoS attacks. For example, DDoS Shield [5] characterizes L7 DDoS attacks on the basis of the application workload parameters that they exploit. It presets the threshold values on the workload parameters according to the measurements and labels the behaviors that exceed the thresholds as malicious; Yatagai et al. [6] proposed a method that detects HTTP GET flood by modeling the browsing order of webpages and the correlation between browsing time and page information size. In general, statistical methods are efficient and steerable. They have decent accuracies in discovering simple L7 DDoS attacks such as HTTP flood and low-and-slow DDoS attack, however, they may have non-ideal effects on handling unseen and complicated attacks.

2) Learning-based methods: As machine learning algorithms are becoming more and more sophisticated, many researchers harness such techniques on big data for detecting L7 DDoS attacks. Seufert et al. [7] proposed a three-layer feed-forward neural network to detect L7 DDoS attacks, using features extracted from the header fields of packets; Yadav et al. [8] applies Stacked AutoEncoder, a deep learning architecture that aims to receive high level features, to generate features from web server logs and build a logistic regression classifier to identify L7 DDoS attacks. Besides, researchers also proposed unsupervised-learning-based detection methods that can extract knowledge or patterns from unlabeled data. ARTP [9] detects L7 DDoS by leveraging the K-means algorithm and performing analysis on features such as request interval, request chain context, and request length. In summary, while a trend, leveraging machine learning in identifying L7 DDoS has mixed results based on the feature extraction method, the system design, and the learning algorithm.

3) Markov-based methods: A Markov model is a stochastic model used to model randomly changing events in probability theory [10]. It assumes that the future state depends only on the current state, and we can infer the next state by performing probability analysis on its past. Works such as [11]–[14] track the related behavior of the users and utilize hidden semi-Markov model along with random walk graph to trace the attacks. We consider Markov-based methods as the state of the art because L7 DDoS attacks are *stateful*. As solutions to tackle stateful problems, Markov-based methods can provide fine-grained detection and classification results with decent accuracies. Inspired by this idea, we adopt reinforcement learning to the L7 DDoS problem, which is a Markov decision process that inherits advantages from both learning-based approaches and Markov models.

B. Mitigation Approaches

As for the mitigation approaches of L7 DDoS, there are mainly two research directions [1]. One is to mitigate attacks on the victim side, such as blocking automated application requests by utilizing user puzzles (e.g., [15], [16]) and setting up specific IPTables or IDS rules [17]. They can reach message-level mitigation granularities but may sacrifice the efficiency of mitigation, since the victim is still required to receive the malicious packets before mitigation, making the system still vulnerable to volumetric L7 DDoS attacks. Another direction is to mitigate L7 DDoS attacks in the network. Once the victim determines the attack sources, it can leverage some traffic filtering or rerouting systems (e.g., [1], [18], [19]) to mitigate attacks from within the network, without consuming any resources on the victim's side. However, they may cause a considerable amount of collateral damage since traffic from benign IPs may also be filtered. Our approach, different from the above directions, incorporates both victim-side mitigations and in-network mitigations for efficient and effective defense against L7 DDoS attacks.

III. Threat Model

An L7-DDoS victim server can be a single-node application server, or contain many components as illustrated in Figure 1. We assume that L7-DDoS attackers can form a massive botnet

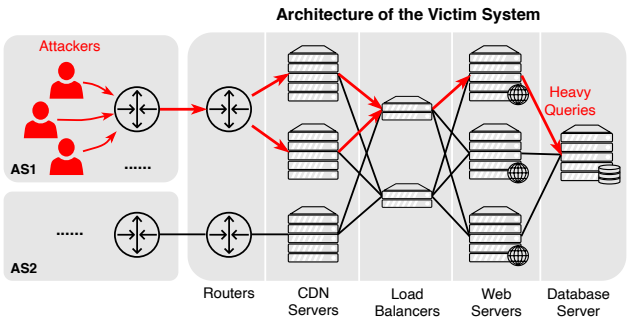


Fig. 1: An example of the victim model. The attacker is performing lethal attacks towards the database server of the victim system.

to exploit the vulnerability of the victim system, with the source IP addresses of the bots distributed over different autonomous systems (ASes). Also, we assume that the attackers can systematically measure the victim server's operation conditions in order to figure out the vulnerable spot, thus adjusting their attack tactics accordingly.

After investigating the operational models of current L7-DDoS attacks, we categorize L7 DDoS attacks into three types: request flooding attack, leveraged attack, and lethal attack.

1) Request Flooding Attack: In this attack, the attacker overwhelms the system by sending application-layer requests at a high rate from different IP addresses. The attacker's bots may locate in certain IP blocks or distribute all over the Internet among different ASes to make it challenging to identify the attack sources. Then, the botmaster can control the bots to generate requests of any arbitrary frequencies and content to overwhelm the victim.

2) Leveraged Attack: This attack leverages the flaws of the victim system to amplify the threat. Thus it can take down the application server with minimal bandwidth and very few requests. For example, low and slow attacks. The attacker controls bots to utilize tools like R.U.D.Y. [20] or Slowloris [21] to slowly send out the requests to the victim. This procedure keeps many connections to the target server open and holds them open as long as possible, tying up the thread. Other types of leveraged attacks may leverage heavy SQL queries, unbalanced API calls, or flawed message queues to overwhelm the victim with a small amount of application-layer requests.

3) Lethal Attack: In this threat, the attacker first scans the victim system to pinpoint the current performance bottlenecks or vulnerabilities (e.g., I/O, memory space, or database server), which are also called lethiferous spots. Then, the attacker formulates the optimal attack tactics to overwhelm the lethiferous spots. Furthermore, the attacker may adjust attack tactics dynamically based on the condition variations of the victim server to make the attack even more effectual. In general, this intelligent attack is highly threatening to all types of victim systems and difficult to detect due to its dynamics.

IV. System Design

A. Overview

In this paper, we assume that L7 DDoS attacks cannot be easily identified through flow-level data since malicious messages will disguise their traffic flows as legitimate. Hence, our solution is on victim-side and considers many factors, such as clients' behavioral information, the network load of the victim server, and the system load of the victim server. We also assume that launching an L7 DDoS attack is a stateful process, just as the process of establishing a TCP connection and collectively sending out the HTTP requests. Thus, we use reinforcement learning (RL) [22], a stateful machine learning technique based on Markov decision process, to construct the attack classification model and formulate appropriate tactics to protect the victim.

RL is a burgeoning area of machine learning concerned with how software agents ought to take actions in an environment to maximize some notions of cumulative rewards. Once the RL agent has made a decision, it gets a reward value to sense whether the current move is suitable or not. Then, it revises the policy to adopt the feedback dynamically. Compared with other L7 DDoS defense approaches, the following advantages make RL more competent to deal with L7 DDoS attacks:

- As a Markov decision process [23], RL aims to maximize the cumulative rewards throughout the monitoring process, which takes advantage of contextual information to infer potential threats.
- In L7 DDoS attacks, the boundary between benign and malicious messages is blurry. Instead of primitively classifying the message as either benign or malicious, the RL agent focuses on formulating appropriate defense tactics that suit current environmental conditions.
- RL allows us to use a multi-objective reward function to mentor the agent on constructing an adaptive and dynamic defense strategy against L7 DDoS attacks.

A typical RL system has five elements: *agent*, *environment*, *reward*, *state*, and *action*. The environment is typically stated in the form of a Markov decision process (MDP) and the MDP transition function gives a new state for each incoming application message, processed in sequence. The agent gets the state from the environment (the environment includes the victim server and some related network infrastructures in our case), then sends the next action to the environment. The environment will conduct the action and give feedback to the agent about the suitability of the action by sending a reward value.

Figure 2 shows the detailed system architecture of our approach. The goal is to train a defense policy π in the training phase and apply it in the monitoring phase to defend against L7 DDoS attacks. The victim can be a single node webserver or a complicated server cluster discussed in Section III. If the victim is a large server cluster, the components (e.g., load balancers, databases, and webservers) need to gather their system information to form an aggregated state s and forward it to the defense agent. In the training phase, there is a reward

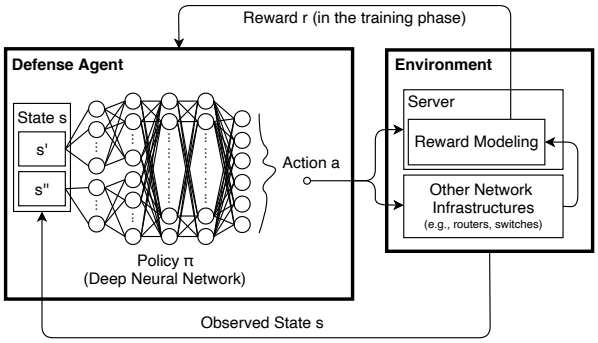


Fig. 2: System Architecture

modeling component on the victim-side, which gets access to the ground truth of the simulated traffic. Therefore, the victim server can evaluate the efficacy of the mitigation action and generate the reward value r according to the reward function in real-time. However, once we complete the training phase and put the defense agent into the monitoring phase, the reward modeling component, as well as the reward values, are no longer needed. In the monitoring phase, the defense agent only needs to generate action a according to the observed state s and trained policy π.

The actions generated by the agent have two categories: the victim-side mitigation actions, which only need to be conducted on the server-side (e.g., scheduling actions), and the in-network mitigation actions, which need to be conducted on some external network infrastructures for filtering out specific traffic.

The rest of this section elaborates on the design details of our approach.

B. States

The state is represented by a state vector s. In our implementation, each state s has twelve dimensions. Each dimension of s is a value that represents a feature. In this L7 DDoS detection system, we expect s to comprehensively represent both the environmental situations and the current application message's features. Thus, we further divide s into two parts, message state s' and environmental state s'' ($s = s' \cup s''$). State s' summarizes the content of the incoming message and the sender's historical behaviors. It helps the defense agent to infer how abnormal the message or the client is. Meanwhile, state s'' extracts the information from the victim server's system situation. It gives the defense agent a perspective of the whole system's healthy degree.

To calculate the state vector in real-time, we define time t as the primary resolution value. For example, if we want to know the average behavior interval of a client, we only need to sample all its past behaviors during the last time t to calculate the value.

1) Message state s': s' is an eight-dimensional vector that extracts eight features from the current application message. It is designed to reflect the historical activities, resource consumption, and behavioral characteristics of the message. The eight features are shown below:

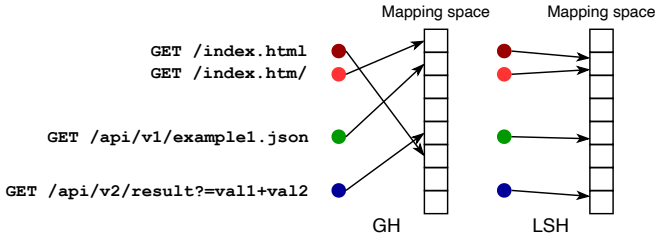


Fig. 3: Examples for the general hashing (GH) and Locality Sensitive Hashing (LSH).

- $bytes_m$*: the number of bytes in a message.*
- $bytes_b$*: traffic size from the message's IP block.* The victim will predefine some IP blocks to classify clients' source IP addresses. $bytes_b$ is a numeric value that indicates the total number of bytes from the incoming message's IP block within time t. This feature is useful to identify request flooding attack.
- ave*: the average behavior interval of the client.* Assume that the client has sent n messages during the last time t, and each interval is denoted by x_i (where $i = 1, 2, ..., n-1$). ave is defined as: $ave = \frac{1}{n-1} \sum_{i=1}^{n-1} x_i$.
- dev*: the average absolute deviation of the client's behavior intervals.* This feature is defined as: $dev = \frac{1}{n-1} \sum_{i=1}^{n-1} |x_i - ave|$.
- num_m*: the number of messages from the client.* This feature is the number of the message sent by this client during the last time t.
- num_{sm}*: the number of all the similar received messages.* For each received message, the server will calculate the number of similar messages within time t promptly. This value plays a crucial role in identifying request flooding attacks, leveraged attacks, and lethal attacks. However, calculating this value is expensive, as we need to buffer a considerable amount of messages in the memory and perform complicated string matchings. Thus, we leverage Locality Sensitive Hashing (LSH) [24] to optimize the calculating process.

Different from traditional hashing functions, LSH can output close or identical values from similar input strings, making it efficient in the duplicate checking. Figure 3 shows an LSH example, where the horizontal positions of the four dots represent the difference in their contents. This method requires training before conducting queries, so we collect request message strings that can represent all the application messages that the server can handle, then preprocess the message strings to make them simplified but still informative enough to outline the messages' intentions, behavioral patterns, and the clients' platforms. For example, the message below is a typical HTTP GET message:

```
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/4.0 (Windows; U; Windows NT 6.0;
     en-US; rv:1.9.1.4)
Accept: text/html,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
```

```
Connection: keep-alive
Cookie: PHPSESSID=n465xmdh435may4ib0skrjq360
```

The preprocessing procedure eliminates redundancies in the strings (strings with red color). We then concat the rest of the information in a fixed order, and joint them by deleting all the spaces and line breaks.

```
/index.htmlMozilla/4.0(Windows;U;WindowsNT6.0;en-US;rv
:1.9.1.4)text/html,application/xml;q=0.9,*/*;q=0.8en-us,
en;q=0.5ISO-8859-1,utf-8;q=0.7,*;q=0.7300keep-
alivePHPSESSID=n465xmdh435may4ib0skrjq360
```

The original message string turns out to be the string above after the preprocessing procedure, and we use such data to train the LSH function for queries. Whenever there is an input message string m, LSH will input the preprocessed string and generate an output hashing value h. The system will store this hashing value h in a set H with an expiration time of t. Every time the system checks the set H, it will remove all the expired values. We also defined a difference threshold Δ to find similar strings. Therefore, the number of similar messages num_{sm} is defined as:

$$num_{sm} = |\{k | k \in H \wedge |k - m.h| \leq \Delta\}|.$$

• *$cons$: request consumption.* The sever estimates the consumptions of all the requests that it can handle in advance and builds a precalculated consumption score table. Given a message m, the server will extract the request from m and generate the $cons$ value according to the consumption score table. This feature is important to identify leveraged attacks.

• *φ: the ratio of incoming traffic size to outgoing traffic size.* The server estimates the outgoing traffic size $bytes_o$ if it responses this message, then calculate the ratio by: $\varphi = bytes_m/bytes_o$.

2) Environmental state s'': s'' is a four-dimensional vector that extracts four features from the server and network's current conditions. This vector is supposed to be a good representative of the environmental metrics so that the agent can correctly infer how dangerous the server's condition is and what is the system bottleneck currently. The four features are shown below:

• *$util_{cpu}$: CPU utilization.* This value is the occupancy rate of the CPU. If the victim system has multiple servers, $util_{cpu}$ is equal to the maximum CPU occupancy rate in the cluster.

• *$util_{mem}$: memory utilization.* This value is the occupancy rate of the memory. If the victim system has multiple servers, $util_{mem}$ is equal to the maximum memory occupancy rate in the cluster.

• *$util_{link}$: link utilization.* This value is the occupancy rate of the link bandwidth. If the victim system has multiple link, $util_{link}$ is equal to the maximum link occupancy rate in the system.

• *$eutil_{link}$: expected link utilization.* If the victim has statistical data about the expected link utilization rates in different periods of the week, this value is the expected link utilization rate in an ordinary situation. Otherwise, this value is the $util_{link}$ during the previous time t.

C. Actions

As discussed in Section II, individuals can utilize in-network and victim-side mitigation approaches to defend against L7 DDoS attacks. We further derived six types of particular actions (shown as below) that an agent can take in the defense process. Each of the action a targets some specific circumstances.

- Action a_i: enabling the server to receive and respond the current application message ordinarily.
- Action a_{ii}: enabling the server to receive the current message ordinarily but postpone the processing procedure.
- Action a_{iii}: drop the current application message on the victim-side.
- Action a_{iv}: drop all the application messages that have the content similar to the current message on the victim-side.
- Action a_v: blocking all the traffic from the IP address of the current application message in the upstream router.
- Action a_{vi}: blocking all the traffic from the IP block of the current application message in the upstream router.

a_i and a_{ii} are *scheduling actions*, a_{iii}, a_{iv}, a_v, and a_{vi} are *defensive actions*. In another taxonomy, a_i, a_{ii}, and a_{iii} are *single-targeted actions*, which only affect the current application message. a_{iv}, a_v, and a_{vi} are *multiple-targeted actions*, which affect a group of application messages.

Conducting defensive actions on particular messages does not necessarily mean the messages are malicious because the agent can choose to sacrifice some false positive rates to ensure the functioning of the server during a severe attack. Similarly, conducting scheduling actions on a particular message does not guarantee the legitimacy of the message. If the system is on idle time, and the malicious message cannot cause some real harm against the server, the agent will take conservative strategies to minimize collateral damages.

D. Reward Function

The overall objective of the reward function is mentoring the defense agent to form a defense policy to fulfill the following requirements in the training phase:

- When system occupation rate is low, minimize the false positive rate of mitigation to ensure all the legitimate messages can be properly processed.
- When system occupation rate is high, maximize the true positive rate of mitigation to block all possible attacks in order to prevent the system from crushing.
- The agent is encouraged to conduct multiple-targeted actions rather than single-targeted actions so that the agent can discover rules in the attacks instead of inefficiently labeling every single message.

In order to address the goals above, we construct a piecewise function $R(a(m))$ as the reward function to mentor the agent conduct suitable actions on correct messages. Here, $a(m)$ denotes conducting action a on message m. In the training phase, whenever the action a is placed, the reward modeling component will use $R(a(m))$ to calculate the reward value

r, telling the defense agent how suitable the current action a is. In this paper, if the agent conducts defensive actions on the legitimate messages, we consider these messages as false positive samples, and vice versa. We also define γ the system occupation rate, which is calculated as:

$$\gamma = max(util_{cpu}, util_{mem}, util_{link}).$$

Additionally, $|fp|$ denotes the number of false positive samples, $|tp|$ denotes the number of true positive samples, $|fn|$ denotes the number of false negative samples, and $|tn|$ denotes the number of true negative samples. We define a policy transition threshold value α to decide when the agent should adjust the defense policy to minimize the false positive rate or to maximize the true positive rate. In this paper, we set α as 0.75.

When $\gamma < \alpha$, we set the reward function $R_1(a(m))$ for single-targeted actions as Equation 1.

$$R_1(a(m)) = \begin{cases} -2 & for\ false\ positive\ sample \\ 1 & for\ true\ positive\ sample \\ -1 & for\ false\ negative\ sample \\ 0 & for\ true\ negative\ sample \end{cases} \quad (1)$$

This reward function gives the agent more penalties when false positive generated, which aims to constraint the agent to ensure all the possible legitimate messages can be properly processed when the system load is within a safe zone.

In the scenario that the agent is making multiple-targeted actions, and $\gamma < \alpha$. Assume that the action a will affect a set of messages $M = \{m_1, m_2, ..., m_n\}$, we set the reward function $R_2(a)$ as Equation 2.

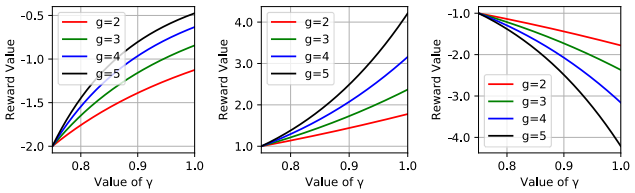
$$\begin{aligned} R_2(a(M)) &= \eta \sum_{i=1}^{n} R_1(a(m_i)) \\ &= \eta(-2|fp| + |tp| - |fn|) \end{aligned} \quad (2)$$

Where η is the reward multiples. We can set η as a value more one so that the agent would get extra rewards or penalties when making multiple-targeted actions. The larger η is, the more the agent is encouraged by the reward functions to take multiple-targeted actions for conducting the defense policy effectively. This mechanism is necessary for the defense agent because monitoring a large amount of incoming messages is an expensive operation and could become the a system vulnerability itself. The agent can fix this problem by frequently generating multiple-targeted actions.

When $\gamma \geq \alpha$, the victim system is heavily loaded, which means the highest priority of agent is to mitigate as many L7 DDoS attacks as possible to guarantee the proper functioning of the server. In this scenario, we set the reward function $R_3(a(m))$ for single-targeted actions as Equation 3.

$$R_3(a(m)) = \begin{cases} -\frac{2}{(\frac{\gamma}{\alpha})^g} & for\ false\ positive\ sample \\ (\frac{\gamma}{\alpha})^g & for\ true\ positive\ sample \\ -(\frac{\gamma}{\alpha})^g & for\ false\ negative\ sample \\ 0 & for\ true\ negative\ sample. \end{cases} \quad (3)$$

Where g is the hazard index, an input parameter that determines how eager the victim wants the attack to be mitigated. The larger g is, the more tactics shifts the agent will have according to the environment, but g should always be larger or equal to 1. Figure 4 shows the curves of the reward function in this scenario with different g values (we set $\alpha = 0.75$ in the curves), we can intuitively see the variation of the reward functions based on the change of γ. The agent will get less and less penalties from false positive samples with the increasing of γ. Conversely, both the rewards from true positive samples and the penalties from false negative samples will rise significantly. This reward function design will constraint the agent to identify and block as many malicious application messages as possible, with the cost of sacrificing a little bit false positive rate.



(a) For false positive samples (b) For true positive samples (c) For false negative samples

Fig. 4: Single-targeted Reward Functions for $\gamma \geq \alpha$ and $\alpha = 0.75$

In the scenario that the agent is making multiple-targeted action a on a set of message M ($M = \{m_1, m_2, ..., m_n\}$), and $\gamma \geq \alpha$, we set the reward function $R_4(a(M))$ as Equation 4:

$$\begin{aligned} R_4(a(M)) &= \eta \sum_{i=1}^{n} R_3(a(m_i)) \\ &= \eta(-\frac{2}{(\frac{\gamma}{\alpha})^g}|fp| + (\frac{\gamma}{\alpha})^g|tp| - (\frac{\gamma}{\alpha})^g|fn|) \end{aligned} \quad (4)$$

$R_4(a(M))$ is in direct proportion to the summation of reward values that returned by all the affected messages. Still, we use the reward multiples parameter η to encourage the agent to take multiple-targeted actions rather than single-targeted actions.

E. Training

The training of the deep reinforcement learning agent follows Q-value iteration [25]. For every state s, the agent will generate an action a, which creates a state-action pair. The reward function will also return a reward value r based on the state-action pair, therefore, we define a function Q that calculates the quality of a state-action combination: $Q : s \times a \rightarrow r$.

At time i, assuming the agent is located in s_i'' and receives a message state s_i', it will select an action a_i to take. After the agent observed the reward r_i, it will enter a new environmental state s_{i+1}'' and update the value of Q. The core of the algorithm is the value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_i, a_i) \leftarrow (1-\beta) \cdot Q(s_i, a_i) + \beta \cdot (r_i + \zeta \cdot \max_a Q(s_{i+1}, a)),$$

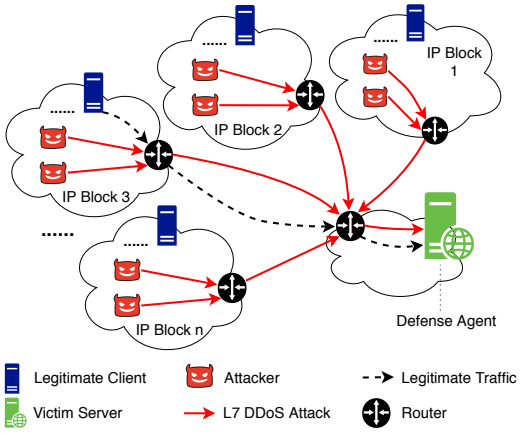


Fig. 5: Topology of Simulated Network Environment

where β is the learning rate, and ζ is the discount factor.

However, the state s we use in this schema is a twelve-dimensional vector, which could generate too large value space for the system to cover in both training and monitoring phases. To tackle this problem, we use a deep neural network to serve as a likelihood function for estimating the $Q(s, a)$.

Just as the topology diagram in figure 2 shows, we leverage a five-layer neural network to approximate the policy function. There are three hidden layers, one input layer, and one output layer in the neural network. The input layer has 12 nodes to import the state vector s, and the output layer has six nodes to generate the recommendation rates for six possible actions respectively. The second and fourth layers have 14 nodes, while the third layer has 15 nodes. A unique aspect about neural network is that the first and second layer are not fully connected. Instead, we separate the nodes for s' from s'' to ensure that the neural network can treat the two sub-state vectors differently.

The training of the agent is similar to the training of ordinary neural networks, in which we define a loss function to measure how good the agent's tactic is. The loss is a value that indicates how far our action a is from the actual target:

$$loss = (r + \epsilon \max_{\acute{a}} \hat{Q}(s, \acute{a}) - Q(s, a)),$$

where ϵ is the decay rate, and $r + \epsilon \max_{\acute{a}} \hat{Q}(s, \acute{a})$ is the actual target. The training of the neural network is also the process of minimizing the $loss$ value with back propagation.

Each state s consists of a message state and an environmental state ($s = s' \cup s''$). The agent will continuously get environmental state s'' but only get message state s' when there is an incoming application message. Thus, the agent will only be activated when it receives s' in both training and monitoring phases.

V. Evaluation

A. Implementation and Simulations

We utilized Open vSwitch [26] and Mininet [27] to construct the simulation environment. Figure 5 shows the basic topology of the simulated network environment. There are n IP blocks in this network; each of them has 5 legitimate clients and 5 malicious clients. Besides, we constructed the RL-based L7 DDoS attack defense system with OpenAI Gym [28] and Keras [29].

We simulated a victim system by constructing a Node.js web server that handles HTTP requests and SMTP requests. The server runs on a virtual machine with 6GB RAM and a 4-core 2.0 GHz CPU. It also maintains an HTTP-based API that can read its hard disk and return selected images. The API is a designed performance bottleneck (lethiferous spot) for attackers to exploit.

For L7 DDoS attacks, we used simulated traffic rather than captured traffic because L7 DDoS attacks are diverse — malicious messages in one environment can be legitimate in another. Moreover, there are few packet-level L7 DDoS traffic available in public repositories. The majority of the existing public L7 DDoS datasets are log files or preprocessed features. Thus, we used the Application Layer DDoS Simulator [30] to simulate request-flooding attacks. For leveraged attacks, we used Slowloris [21] to simulate the most typical leveraged attack — low and slow attack. In the end, we used modified HULK program [31] to generate lethal attacks towards the known performance bottleneck.

Based on our empirical studies (of which we skip the details for space considerations), we set some of the parameters in this approach as follows: for the number of IP blocks n in the evaluation, we set it to be 10; for the policy transition threshold value α, we set it to be 0.75; the learning rate β for agent training is 0.25 in this implementation; for the hazard index g, we set it to be 3.

B. Ability of Mitigating Attacks

Although the proposed method does not need to generate the precise labels of incoming messages, we can still evaluate its ability to mitigating L7 DDoS attacks by inferring the correctness of output action a. As indicated in Section IV-C, we count messages that trigger scheduling actions as legitimate requests. Conversely, we count messages that trigger defensive actions as malicious messages. All the evaluation metrics in this section are based on this regulation.

We simulated benign messages and launched the L7 DDoS attacks to the victim simultaneously for evaluating the agent's accuracy of mitigating L7 DDoS attacks. During the test, we firstly ensured the volume of legitimate messages was always under the victim server's capacity so that the server would not crash due to legitimate activities. Afterward, we adjusted the amount of malicious messages to test the performance of this approach with different system loads. Here, we consider the system load as the system occupation rate γ defined in Section IV-D.

Figure 6 shows the trends of mitigation accuracies, false positive rates, and true positive rates during different system loads (we consider malicious messages as positive samples in this paper), where the y-axis represents the system workload, and the x-axis represents the rate value.

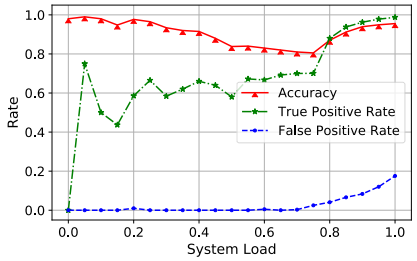


Fig. 6: Performance Metrics of Different System Loads

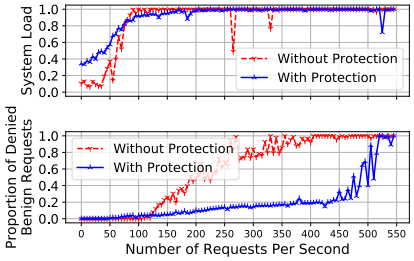


Fig. 7: Efficacy of Attack Mitigation

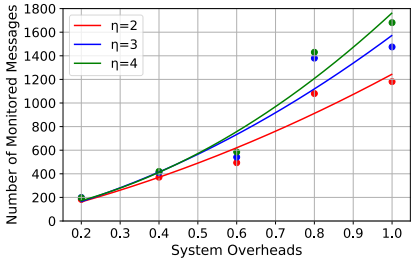


Fig. 8: System Overheads for the Defense Agent

When the system load is at a low rate, we can get a nearly 100% mitigation accuracy, since the majority of the messages are benign, the agent will minimize the false positive rate at this point. However, when both the system workload and the volume of attacks are increasing, the accuracy has some apparent drops. Although the attack volume increased, the defense agent still uses the defense tactic that aims at minimizing the false positive, guiding the agent to sacrifice the true positive rate for letting the server adequately process most of the legitimate requests. Thus, the false positive rate remains approximately zero within this zone. On the contrary, the trend for true positive rates fluctuates in low system-load scenarios, because the volume of malicious requests is still low, making it hard to reach statistical significance.

The transition comes in when the system workload is at 0.75. From this point, the defense agent assumes that the server system is in hazardous conditions, so it has to mitigate as many attacks as possible to protect the victim server. As the system load goes higher, the value of $(\frac{\gamma}{\alpha})^g$ in the reward functions becomes larger, and the false positive rate becomes less and less critical. Hence, we can distinctly see that the defense agent starts maximizing the true positive rate. This sacrifices some false positive rates but still increases the overall accuracies. In the end, when the system load stabilizes at 100%, the accuracy, true positive rate, and false positive rate are 0.9553, 0.9873, and 0.1756, respectively.

In brief, this evaluation result proofs that the reinforcement learning agent can intelligently formulate applicable tactics to defend against L7 DDoS attacks, and the mitigation accuracies of the tactics are satisfactory.

C. Mitigation efficacy

We evaluated the efficacy of our approach and presented the results in Figure 7. The x-axis in the figure represents the number of application messages made to the victim server per second, including both the legitimate messages and malicious messages. The y-axis of the upper subplot represents the system load, while the y-axis of the lower subplot represents the proportion of denied benign messages.

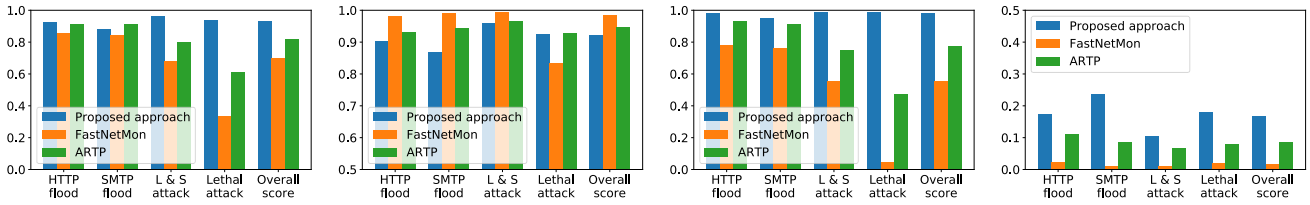
Initially, the resource consumption of the server without protection is lower than the server with the agent running because the deployment of the defense agent costs a certain amount of computing resources, especially for maintaining the LSH function, message monitoring, and the operation of the deep neural network. However, this consumption will pay back shortly with the increasing number of receiving messages. We can see that the proportion of denied benign messages increases observably for the server without protection. If we assume that a server is considered to be proper functioning when the deny rate of legitimate messages is lower than 20%, the capability of the server without protection is approximately 140 messages per second. After reaching 250 messages per second, the server without protection is almost useless, with the majority of message requests getting denied. While for the server with the defense agent's protection, the deny rate of legitimate messages goes higher than 20% only after the number of messages per second hitting 440, which is **3.15** times the capability of the unprotected server.

Therefore, this approach can significantly enhance the service capability of the server and make the victim resilient during some severe L7 DDoS attacks.

Additionally, running the defense agent requires system overheads. Figure 8 shows the system overheads for the defense agent when monitoring different numbers of messages per second. We experimented by removing the server function of the victim system. Hence, all of the computing resources were devoted to the defense agent, and we can directly measure the system overheads. From previous experiments, we already know that the capability of the server without protection is approximately 140 messages per second. In this figure, we can see the agent can monitor 1.5 times the maximum messages that the server can process with less than 20% of computing resources. Besides, the larger the reward multiples η is, the more the agent is encouraged to take multiple-targeted actions, thus increasing the monitoring efficiency. We set η as 2 in other experiments. However, if we increase the value of η to 4 and devote all the system overheads to the agent, the agent can monitor nearly 1800 messages per second, which is more than 12 times the server's maximum processing capability.

D. Comparison evaluation

We also compared our approach with two other DDoS attack detection approaches. One is FastNetMon [32], a commercial DDoS detection software that applies statistical methods. Although this software is not designed for L7 DDoS attacks, it



(a) Accuracy (b) Precision (c) Recall (d) False Positive Rate

Fig. 9: Results of Comparison Evaluation

offers good performance on general DDoS detection. Another is ARTP [9], a learning-based detection approach particularly designed for L7 DDoS attacks.

To evaluate their performance, we simulated the traffic of HTTP flood, SMTP flood, low and slow attacks, and lethal attacks. Figure 9a shows the accuracies of these approaches. Our approach achieves the best accuracy scores for detecting HTTP flood, low and slow attack, and lethal attack. ARTP only slightly exceeds our approach in detecting SMTP flood. Although our approach does not have perfect scores in precision and false positive rate (as shown in Figure 9b and Figure 9d), it still accomplishes the initial design objective, which is to sacrifice a little bit false positive rate to block as many malicious requests as possible during the peaks of attacks. As we can see from Figure 9c, our method achieves the best recall scores in identifying all types of attacks because it can adjust the mitigation strategies dynamically based on the condition of the victim server.

E. Robustness in different environments

We have noticed the importance of the robustness of RL-based approaches. Therefore, we deployed the trained agent in different environments to evaluate the adaptability of our approach. Figure 10 shows the evaluation results of robustness. Environment 2 is slightly different from the trained environment, which is assigned with a 4GB RAM and a 3-core 2.0 GHz CPU. Environment 3 has the same hardware as environment 2 but runs different application services, which offers video streaming and download services with HTML5. Environment 4 has a 12GB RAM and a 8-core 2.7 GHz CPU, which is quite a different hardware environment compared with others.

The evaluation results show that the accuracies of our approach only drop a little bit if the environment changed slightly. Even if the application service changed in environment 3, the agent could still achieve around a 90% accuracy at the attack peak. In fact, the design philosophy of our approach promotes adaptability. For instance, many of the features we designed are proportions rather than an absolute value; we avoided using the application-specific features in the state.

F. Agent training

In the training phase, we trained the defense agent in the platform for 80 hours, with nearly 35,000 episodes. We recorded the L7 DDoS attack detection accuracies during different stages to evaluate the efficiency of agent training. As we can see from the results (Figure 11), the training process goes relatively slow and precarious during the first 20,000 episodes. Then it evolves quickly from around 65% accuracy to more than 90% accuracy in the next 7,500 episodes, enabling the defense agent to offer decent protection to the victim server. Eventually, the accuracy of the agent stabilizes near 96% after 30,000 episodes of training. This evaluation also proves that it is feasible to retrain the defense agent within half a week to fit a whole new environment in a real deployment.

G. Service delay

Since the defense agent will continuously inspect all the incoming application messages during the server operation, the service delay could be an underlying concern that impacts the user experience. Therefore, we measured the lengths of delays under different system workloads and presented the results in a boxplot (as shown in Figure 12).

Here, we define the length of delay as the time duration from sending out a message to receiving the whole reply. A delay less than 0.5 seconds is imperceptible to the users. As we can see in the Figure 12, the average delay time for the server remains under 0.5 seconds when the system workload is less or equal to 90%. Although the delay without any defense approaches implemented is around 0.25 seconds, the presence of the defense agent is still unremarkable to the clients most of the time. Even when the system workload reaches 100% and the attackers are trying to overwhelm the victim server, the service delay can still lay within an acceptable range (0.4 seconds to 1.25 seconds). Meanwhile, the system without any protections is already in an unusable condition under this circumstance.

VI. Conclusions

L7 DDoS attacks are becoming far more sophisticated and threatening than before. Compared with traditional DDoS attacks, L7 DDoS are difficult for conventional DDoS approaches to detect and mitigate. This paper proposes a reinforcement-learning-based approach that can self-evolve according to the interactions with the environment. It continuously monitors and analyzes a variety of metrics related to the server's load, the dynamic behaviors of clients, and the network load of the victim, to detect and mitigate L7 DDoS attacks, including choosing the most appropriate mitigation tactic. Different from typical DDoS detection approaches that label the traffic as either legitimate or malicious, this approach

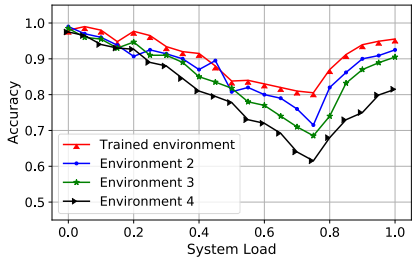


Fig. 10: Accuracies in Different Environments

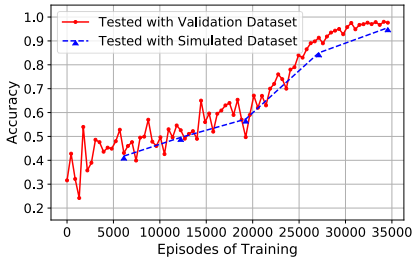


Fig. 11: Convergence Trend while Training

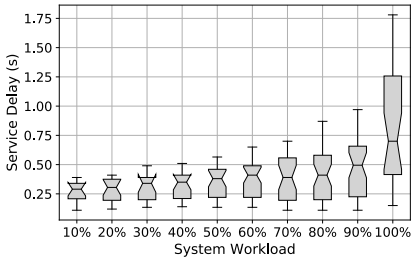


Fig. 12: Box Plot for Service Delay

employs a new multi-objective reward function that minimizes false positive rate to avoid collateral damage when the victim system load is low and maximizes the true positive rate to prevent the server from collapse when the victim system load is high. Evaluation shows that this approach protects a victim server from L7 DDoS attacks effectively; it can mitigate 98.73% of the malicious application messages when the victim is brought to its knees and achieve minimal collateral damage when the L7 DDoS attack is tolerable.

VII. Acknowledgment

The authors would like to thank Derek Strobel from the University of Oregon for his proofreading and constructive suggestions on this work.

References

[1] A. Praseed and P. S. Thilagam, "DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 661–685, 2018.

[2] S. Anthony, "GitHub battles "largest DDoS" in site's history, targeted at anti-censorship tools," https://arstechnica.com/, March 30, 2015.

[3] V. Simonovich, "Imperva blocks our largest DDoS L7/brute force attack ever (peaking at 292,000 RPS)," https://www.imperva.com/blog/, July 24, 2019.

[4] Cloudflare, "Application layer DDoS attack," https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/.

[5] S. Ranjan, R. Swaminathan, M. Uysal, and E. W. Knightly, "DDoS-Resilient scheduling to counter application layer attacks under imperfect detection." in *INFOCOM*, 2006.

[6] T. Yatagai, T. Isohara, and I. Sasase, "Detection of HTTP-GET flood attack based on analysis of page access behavior," in *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Aug, pp. 232–235.

[7] S. Seufert and D. O'Brien, "Machine learning for automatic defence against distributed denial of service attacks," in *2007 IEEE International Conference on Communications*, 2007, pp. 1217–1222.

[8] S. Yadav and S. Subramanian, "Detection of application layer DDoS attack by feature learning using stacked autoencoder," in *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, 2016, pp. 361–366.

[9] K. M. Prasad, A. R. M. Reddy, and K. V. Rao, "Anomaly based real time prevention of under rated App-DDoS attacks on web: An experiential metrics based machine learning approach," *Indian Journal of Science and Technology*, 2016.

[10] P. A. Gagniuc, *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[11] Y. Xie and S.-Z. Yu, "A novel model for detecting application layer DDoS attacks," in *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)*, vol. 2, 2006, pp. 56–63.

[12] S.-Z. Yu and Y. Xie, "A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors," *IEEE/ACM transactions on networking*, vol. 17, no. 1, pp. 54–65, 2009.

[13] Y. Xie and S. Yu, "Monitoring the application-layer DDoS attacks for popular websites," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 15–25, Feb 2009.

[14] C. Xu, G. Zhao, G. Xie, and S. Yu, "Detection on application layer DDoS using random walk model," in *2014 IEEE International Conference on Communications (ICC)*, pp. 707–712.

[15] H. Zhang, A. Taha, R. Trapero, J. Luna, and N. Suri, "Sentry: A novel approach for mitigating application layer DDoS threats," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 465–472.

[16] S. Sivabalan and P. Radcliffe, "A novel framework to detect and block DDoS attack at the application layer," in *IEEE 2013 Tencon-Spring*, pp. 578–582.

[17] D. Moustis and P. Kotzanikolaou, "Evaluating security controls against HTTP-based DDoS attacks," in *IISA 2013*, pp. 1–6.

[18] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS attack detection and mitigation using SDN: methods, practices, and solutions," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, 2017.

[19] J. M. Smith and M. Schuchard, "Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive bgp routing," in *2018 IEEE Symposium on Security and Privacy (S&P)*, 2018, pp. 599–617.

[20] M. M. Najafabadi, T. M. Khoshgoftaar, A. Napolitano, and C. Wheelus, "Rudy attack: Detection at the network level and its important features," in *The twenty-ninth international flairs conference*, 2016.

[21] RSnake, J. Kinsella, H. Gonzalez, and R. E. Lee, "Slowloris HTTP DoS," https://github.com/XCHADXFAQ77X/SLOWLORIS, 2009.

[22] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[23] R. A. Howard, "Dynamic programming and markov processes." 1960.

[24] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[25] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[26] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *The 12th NSDI*, 2015, pp. 117–130.

[27] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, 2010, pp. 19:1–19:6.

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

[29] F. Chollet *et al.*, "Keras," 2015.

[30] Storm Security, "Application layer DDoS simulator," https://stormsecurity.wordpress.com/2009/03/03/application-layer-ddos-simulator.

[31] B. Shteiman, "Hulk DoS tool," https://github.com/grafov/hulk, 2017.

[32] P. Odintsov, "FastNetMon–very fast DDoS analyzer with sflow/netflow/mirror support," https://github.com/pavel-odintsov/fastnetmon/.