# Towards Intelligent Defense against Application-Layer DDoS with Reinforcement Learning

Yebo Feng
University of Oregon
Eugene, USA
yebof@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, has become a major rising threat to today's Internet. However, because the traffic from an L7 DDoS attack appears totally legitimate in transport and network layers, it is difficult to detect and defend against an L7 DDoS attack with traditional DDoS solutions.

In this paper, we propose a new, reinforcement-learning-based approach to detecting and mitigating L7 DDoS attacks. By continuously monitoring and analyzing the system load of the victim server, the dynamic behaviors of clients, and the network load of the victim server, our approach can choose one of the most suitable mitigation actions, such as blocking DDoS upstream, blocking DDoS locally, or postponing L7 requests, thus achieving the best mitigation efficacy of the L7 DDoS attack. Moreover, with the help of a new multi-objective reward function, when a L7 DDoS attack is overwhelming the reinforcement learning agent can selectively sacrifice legitimate requests to keep the victim server functioning, and when otherwise the agent affects little legitimate requests.

Our evaluation results show that our approach can protect a victim server from L7 DDoS attacks effectively by detecting 98.73% of the L7 DDoS traffic flows at the peak system load, with most of them mitigated.

## KEYWORDS

application-layer DDoS, distributed denial of service (DDoS), reinforcement learning, anomaly detection

## 1 INTRODUCTION

Application-layer distributed denial of service attacks [21], or layer 7 (L7) DDoS attacks, represent a type of malicious behaviors 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. Kaspersky Lab's DDoS report for Q1 2019 [12] indicates that L7 DDoS attacks are growing both qualitatively and quantitatively. One of the best-known L7 DDoS attacks happened in March 2015 when massive 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, on July 24, 2019, Imperva reported the most recent notable L7 DDoS attack [27]. The 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.

Compared with traditional DDoS attacks, L7 DDoS attacks are more complicated to launch; however, they are more efficient in inundating a victim server and more challenging to detect. The three characteristics below are the main reasons for making such attacks particularly dangerous:

(1) Traditional DDoS attacks target and cause anomalies at the network or transport layer (e.g., flooding a network link), while L7 DDoS attacks target primarily specific application functions or components of a victim server. L7 DDoS attacks primarily cause anomalies at the application layer and does not always lead to anomalies at the lower layers. Popular traffic monitoring tools [9] and DDoS traffic classification approaches [8], which typically only work at the layers below the application layer, can be clueless in defense against L7 DDoS attacks.
(2) Tradition DDoS attacks, while attacking lower layers, do not care to ensure the application-layer legitimacy of the DDoS traffic. Conversely, L7 DDoS attacks flood a victim application with purposely fabricated application-layer messages that look legitimate, which they intend the victim application will process and respond in the same way as legitimate application-layer messages.
(3) In contrast to traditional DDoS attacks that often inundate a network link or exhaust resources for network or transport layer functions, L7 DDoS attacks can first pinpoint specific vulnerabilities of a victim application, such as a bottleneck involved in disk I/O , CPU operation, or memory usage, and then hit the application with application-layer messages that target the vulnerabilities.

Unfortunately, the detection and defense of L7 DDoS attacks are still not well-studied [5, 21]. Worse, attackers continuously evolve their toolkits and develope more sophisticated L7 DDoS attack techniques. Thus, it is 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 a L7 DDoS attacker can purposely fabricate application-layer messages that look legitimate, as discussed above. A L7 DDoS message can even be identical to a legitimate application-layer message.

Interestingly, the legitimacy of application-layer messages is heavily dependent on its environment or context. The same application-layer message may be legitimate in one environemnt, 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. 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 can incoporate 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, for each application message, this method directly outputs the action to take under different circumstances. Actions can include blocking the message upstream, blocking it locally, or postponing its processing, in order to mitigate L7 DDoS attacks.

More importantly, 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 evaluations show that this approach has outstanding efficacy when mitigating L7 DDoS attacks and satisfying performance when running on the server node. With a 0.9553 accuracy and a 0.9873 true positive rate at the peak of L7 DDoS attacks, this approach can identify the majority of DDoS traffic and significantly increase the capacity of the victim server. Also, the implementation of this method is not intricate. With less than 30,000 training episodes, this method can adapt to an utterly unacquainted victim server environment.

The rest of this paper is organized as follows. After describing the related work in Section 2, we introduce the threat and defense models in Section 3. We then detail this approach in Section 4. We evaluate this approach in Section 5 and discuss open issues and future work in Section 6. The conclusions of the paper is in Section 7.

## 2 RELATED WORK

Previous methodologies differs from our proposed approach in that (1) our approach is an integration of attack detection, message classification, and attack mitigation. While previous approaches treat attack detection & classification and attack mitigation as isolated components; (2) previous L7 DDoS detection & classification approaches explicitly label the identities of application messages, while the proposed approach implicitly detect L7 DDoS attacks and classify the messages; (3) previous L7 DDoS mitigation methods are either in-network or victim-side approaches, while the proposed approach can choose adaptive mitigation methods in different situations; (4) most of the previous approaches set a defense strategy and stick on it throughout their deployments, while the proposed method can adopt different defense strategies according to the victim server's conditions.

We detail related work below, including the detection & classification of L7 DDoS and attack mitigation.

### 2.1 Detection & Classification Approaches

As for detection & classification of L7 DDoS attacks, due to the unique attack model, it is infeasible to identify L7 DDoS attacks only through traffic flows. Thus, most existing L7 DDoS attack detection & classification approaches rely on [payload information and historical behaviors of clients] [21] to find the trace of L7 DDoS. However, due to the high complexity of L7 DDoS attack and the [low detection efficiency], many of the current methodologies are [difficult to deploy] and [limited to only certain types of attack patterns]. This paper lists some representative detection methods as follow, which are categorized into three types: statistical method, learning-based method, and probability method.

- **Statistical Method:** overview of such caregory. there are some statistical detection methods based on the activity rates, request costs, and system metrics. For example, DDoS Shield [23] uses some threshold values to model the legitimate requests and limit the malicious application behaviors; [38] detects HTTP GET flood by analyzing the page access behaviors.
- **Learning-based Approach:** overview of such caregory. [25] detects L7 DDoS attacks by including partial information of HTTP request headers in a three-layer feed-forward neural network; [17] utilizes random tree algorithm to construct the classification model based on preprocessed datasets; [37] applies Stacked AutoEncoder to generate features from web logs, then builds a logistic regression classifier to detect L7 DDoS attacks; [20] proposed a clustering-based strategy called Anomaly-based Real Time Prevention (ARTP) of under-rated L7 DDoS attacks, which is able to conduct early detection to even unknown attacks.
- **Markov ... Method:** overview of such caregory. The L7 DDoS attack is the consequence of a series of behaviors and each prior activity of the clients can contribute to the estimation of attack probability. [34–36, 39] track the related behavior of the users and use hidden semi-Markov model and random walk graph to classify the attack.

### 2.2 Mitigation Approaches

In general, there are two directions for us to mitigate L7 DDoS attacks [21]. One is to mitigate on the victim-side. For example, blocking automated application requests by utilizing user puzzles

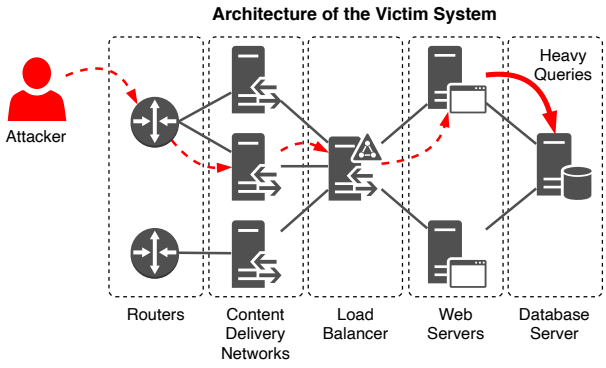



**Figure 1: An example of the victim model. The attacker is performing lethal attacks towards the database server of the victim system.**

(e.g., [28, 31, 40]), which has a desirable effect on some of the human-user-targeted applications but is helpless for systems that allow automatic operations. Another direction is to mitigate in the network. Once individuals determine the attack sources, it is straightforward to leverage some traffic filtering or throttling systems like the firewall to wipe out the identified attacks [21]. However, this approach is highly dependent on the accuracy of the detection approaches.

## 3 THREAT AND DEFENSE MODELS

An L7-DDoS victim server can be a single-node application server, or contain many components as illustrated in Figure 1. It may include a content distribution network (CDN) to provide end-users with high availability and fast content delivery, where the CDN uses reverse proxy servers to request missing content from the original application server. It may include a load balancer when the application server is composed of more than one machine. It may also include database servers at the backend to maintain data and answer queries. Any such component may be vulnerable to an L7-DDoS attack.

### 3.1 Threat Model

With usually a complicated infrastructure, a victim may have various vulnerabilities that a L7-DDoS attack can exploit. For example, the bandwidth of its CDN could become the bottleneck upon a peak of requests, the application server itself may be exhausted if the processing load is very high, or the database servers may be under pressure when an extremely large number of disk I/O operations are triggered.

Without losing generality, we assume the L7-DDoS follows a threat model as illustrated in Figure 1. L7-DDoS attackers can form a massive botnet to exploit the vulnerability of the victim, with the source IP addresses of the bots distributed over different autonomous systems (ASes). Upon the command of the botmaster, every bot can send specific application-layer requests to the victim server. 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.

*3.1.1 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 operational mechanism of this attack can be either naïve or sophisticated. First, the attacker nodes can be located in certain IP blocks or distribute all over the Internet with different ASes to make it challenging for identifying the attack sources. Secondly, the botmaster can control the bots to generate requests in monotonous or diverse ways. Attacking with diverse requests brings a significantly higher threat against the victim because detecting such activities is intractable, but launching such an attack requires more coordination and resources.

*3.1.2 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, target thread-based web servers with the aim of tying up every thread with slow requests, thereby preventing benign users from accessing the service [6]. It leverages the vulnerability of HTTP protocol that keeps the connection open during the transmissions of HTTP requests. The attacker controls bots to utilize tools like R.U.D.Y. [16] or Slowloris [24] 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.

*3.1.3 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 spot. Then, the botmaster formulates the optimal attack tactics based on the measurement result. During the attack session, all the bots will follow the formulated tactics to send application requests related to the lethiferous spot for overwhelming the victim. Furthermore, the botmaster 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.

### 3.2 Defense Model

The defense approaches that a victim can adopt to defend against traditional DDoS attacks are mainly network-based. One can defend against traditional DDoS attacks by dropping the traffic of malicious sources in upstream routers [22], filtering the packets in a scrubbing center [1], or rerouting the traffic out of the obstructed links [29]. However, for L7 DDoS attacks, the defense strategies can be more flexible:

- For malicious requests that will add considerable pressures to the link bandwidth, we can utilize network-based defense methods to throttle their traffic before reaching the victim server.

- For malicious requests that only abuse computing resources on the server-side but will not lead to link congestion, we can conduct application-based defense approaches on the server-side. The defense approaches include but not limited to: ignoring the application requests, postponing the requests until the system workload is in an acceptable condition, and CAPTCHA [32].

Although the victims have bountiful selections of defense methods, it is intricate to optimize and coordinate different defense strategies according to the conditions of victim server, identities of the application requests, and network environment. This automated decision-making methodology is one of the key problems this paper attempts to address.

# 4 SYSTEM DESIGN

## 4.1 Overview

L7 DDoS attacks cannot be easily identified through flow-level data since the attack requests will disguise their traffic flows as legitimate. Moreover, performing an L7 DDoS attack is a stateful process, such as the process of establishing a TCP connection and sending out the HTTP requests. This feature makes the stateless traffic data limited in the inference of attacks. Thus, we need to inspect clients' behavioral information and take more factors into consideration for attack detection, such as conditions of the victim server, network traffic situation, and request histories of the clients to set up dynamic adaptive detection strategies. In order to achieve that, we use reinforcement learning (RL) [11] to construct the attack classification model and formulate appropriate tactics to protect the victim. Reinforcement learning 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 will get a reward value to sense whether the current move is suitable or not. Then, it will revise the policy to adopt the feedback dynamically.

Compared with other detection and defense approaches, reinforcement learning has the following advantages that make it more competent to deal with L7 DDoS attacks:

- In L7 DDoS attacks, the traffic flows generated by legitimate and illegitimate clients can be identical, so we cannot merely use supervised methods to pinpoint the malicious clients. Conversely, reinforcement learning can train the agent based on the rewards instead of labels, which is more suitable to analyze equivocal behavioral data.
- The client's historical activities are also significant factors for L7 DDoS detection. Reinforcement learning is a Markov decision process [10], which will take past states into inferences for the current decision making.
- Instead of primitively classifying received requests as either benign or malicious, we want to use this approach to formulate appropriate defense tactics based on real-time environmental conditions. Reinforcement learning is particularly efficient in obtaining information from the agent-environment interactions and generate suitable strategies accordingly.

**Table 1: Key Notations**

| Symbol | Description |
|---|---|
| $s$ | A state vector that represents an overall state ($s = s' \cup s''$). |
| $s'$ | A state vector that represents incoming application requests. |
| $s''$ | A state vector that represents the environmental features. |
| $a$ | An action that the agent chooses to take, and $a_i$ represents the $i$th action. |
| $r$ | A reward value that the environment returns to the agent, and $r_i$ represents the $i$th reward value. |
| $\gamma$ | The current system occupation rate. |
| $\alpha$ | The policy transition threshold value. |
| $\eta$ | The reward multiples for adjustments of the reward function. |
| $g$ | The hazard index, determines how eager the agent want the attack to be mitigated. |
| $\beta$ | The learning rate for agent training. |
| $\zeta$ | The discount factor for agent training. |

The rest of this section elaborates on the design details of our approach, some of the vital notations we use in this paper are listed in table 1.

## 4.2 Operational Model

A typical reinforcement learning system has five elements: *agent*, *environment*, *reward*, *state*, and *action*. The environment is typically stated in the form of a Markov decision process (MDP) [15] and the MDP transition function gives a new state for each incoming application request, processed in sequence. The agent gets the state from the environment (in our case, server and network situations are the environments), 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. However, the correctness of the classification results is unknown in some cases, and due to the delay of the external defense system, the agent cannot obtain feedback on the current decision in real-time. So we designed another reward modeling component to infer the possible reward.

Figure 2 is the detailed system architecture of our approach. In the training phase, the reward modeling component also inputs the ground truth information to generate precise reward values. The actions generated by the agent have two categories: the internal defense rules, which only need to be deployed on the server-side (e.g., ignore specific requests), and the external defense rules, which needs an external defense system to deploy them on some network infrastructures for filtering specific traffic.

## 4.3 States

Each state is represented by a state vector $s$, which has twelve dimensions. Each dimension of the state $s$ 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

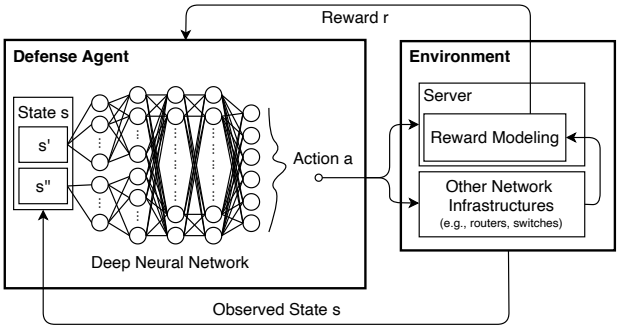



**Figure 2: System Architecture**

current application request's features. Thus, we further divide $s$ into two parts, making $s = s' \cup s''$. Here, request state $s'$ is representative of the incoming request and environmental state $s''$ is representative of the environmental situations, including the victim server's workload information and the network link's occupation.

*4.3.1 Request state.* $s'$ is an eight-dimensional vector that extracts eight features from the current incoming application request. It is designed to reflect the historical activities, resource consumption, and behavioral characteristics of the incoming request. The eight features are shown below:

- **Traffic size from the IP block:** the server will predefine some IP blocks for the clients' source IP addresses. This feature is a numeric value that indicates the total traffic size from the incoming request's IP block within time $t$.
- **Average behavior interval:** this feature is a numeric value which indicates the average request interval from the same source address within time $t$.
- **Interval deviation:** this value is the deviation of all the request intervals of this source address within time $t$.
- **Request size:** this feature is a numeric value which indicates the size of the current incoming request message. If a request is sent by multiple packets, the request size should be the sum of all the related packets' payloads.
- **Number of requests within time $t$:** this value is the number of the requests made by this source address within time $t$.
- **Number of similar requests within time $t$:** for each received request, the server will calculate the number of similar requests within time $t$ promptly. This value plays a crucial role in identifying naive attacks, request flooding attacks, and bottleneck attacks. However, one issue worthy of our concern is that calculating this value is expensive, as we need to buffer a considerable amount of requests in the memory and perform complicated string matchings. Thus, we leverage Locality Sensitive Hashing (LSH) [7] to optimize the calculating process. It firstly calculates a hashing value out of the request message and stores it in the memory for time $t$. We can then figure out the number of similar requests within time $t$ by counting the number of hashing values that fall within a difference threshold $\Delta$. Different from traditional hashing functions that will generate random values out of similar inputs, LSH can output close or identical values from similar input strings. Figure 3 shows an LSH example, where the horizontal positions of the four dots represent the difference in their contents. Thus, LSH is appropriate and efficient in data classification and duplicate checking. This method requires training before conducting queries, so we first collect standard request message strings that can represent all the application requests that the server can handle, then preprocess the message strings to make them simplified but still informative enough to outline the requests' intentions, behavioral patterns, and the clients' platforms. For example, the message below is a typical HTTP GET request:

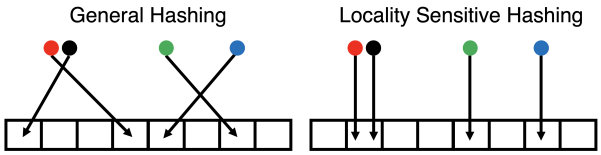



**Figure 3: Examples for the general hashing and Locality Sensitive Hashing.**

```
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-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=n465xmdh435may4ib0skrjq360
Cache-Control: no-cache
```

The preprocessing procedure removes all the attributes' names and common information (strings with red color), which eliminates redundancies in the strings. We then sort 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=0.5gzip,deflateISO-8859-1,utf-8;q=0.7,*;q=0.7300keep-alivePHPSESSID=n465xmdh435may4ib0skrjq360no-cache
```

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 query string, LSH will take the preprocessed string to go through the whole space to find the closest training data, and then generate an output hashing value accordingly.

- **Request consumption:** the sever estimates the consumptions of all the requests that it can handle in advance and give a score to the current request base on the precalculated consumption score table.
- **Ratio of incoming traffic size to outgoing traffic size:** the server estimates the outgoing traffic size if it responses this request, then calculate the ratio.

*4.3.2 Environmental state.* $s''$ 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:

- **CPU utilization:** this value is the occupancy rate of CPU.
- **Memory utilization:** this value is the occupancy rate of the memory.
- **Link utilization:** this value is the occupancy rate of the link bandwidth.
- **Expected link utilization:** we assume that the server 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.

## 4.4 Actions

As discussed in Section 3.2, individuals can utilize network-based and application-based approaches to defend against L7 DDoS attacks. We further derived six types of particular action $a$ (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 request ordinarily.
- Action $a_{ii}$: enabling the server to receive the current request ordinarily but postpone the processing procedure for several seconds.
- Action $a_{iii}$: ignoring the current application request.
- Action $a_{iv}$: ignoring all the application requests that have request contents similar to the curent request.
- Action $a_v$: blocking all the traffic from the IP address of the current application request in the upstream router.
- Action $a_{vi}$: blocking all the traffic from the IP block of the current application request in the upstream router.

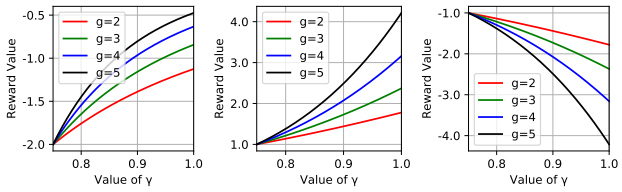
For legitimate requests, the agent can take action $a_i$ if the victim system still has redundancy. However, the agent will take action $a_{ii}$ to deal with legitimate requests if the victim system is heavily loaded.

For suspicious requests, the agent can either take action $a_i$ when the system load is low or take action $a_{ii}$ when the system load is high.

For malicious requests, the agent will determine the action to take based on the severity of the attack and the status of the victim system. If the system is on idle time and the malicious request cannot cause some real harm against the server, the agent will conduct action $a_{iii}$ to handle the illegitimate request. However, the agent will still remain open to future requests of the attacker node for reducing the false positive rate and saving the rule space in the upstream router. If some attacker nodes keep sending requests with malicious intentions and causing considerable damages to the victim server, the agent can notify the upstream routers to conduct action $a_v$. In some situations, the agent may find that the current request is part of an organized L7 DDoS attack that targets particular component of the system, it is sensible to take action $a_{iv}$. Action $a_{iv}$ has direct effects on multiple future requests, which can efficiently prevent the victim server from crushing. Finally, an agent can conduct action $a_{vi}$ to the malicious requests if the victim server is being consumingly attacked. Although $a_{vi}$ may significantly increase the false positive rate, it can offer most effective protection to the victim server in urgent circumstances.

By conducting advisable actions on incoming requests repeatedly, the victim system can obtain a trustworthy defense against L7 DDoS attacks and provide routine service to legitimate clients simultaneously.

## 4.5 Reward Function




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

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

Based on the current state $s$ and reward $r$, the agent will generate two kinds of actions. One type of action is single-targeted, which only decides the current request to be correctly processed or not. Another type of action is multiple-targeted, which decides a group of requests to be correctly processed or not. The overall objectives of the reward function design are mentoring the agent to set up the detection and defense policy to fulfill the following requirements:

- When system consumption is low, minimize the false positive rate to ensure all the legitimate requests can be properly processed.
- When system consumption is high, maximize the true positive rate 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 single request, and the defense of L7 DDoS attack can be more prompt.
- Save the defense budget after the requirements above are satisfied.

In order to address the goals above, we construct a piecewise function $R(s)$ as the reward function to mentor the agent conduct suitable actions on correct requests. In this paper, if the agent conducts action $a_{iii}$, $a_{iv}$, $a_v$, and $a_{vi}$ on the legitimate requests, we consider these requests as false positive samples, and vice versa. We also define $\gamma$ the system occupation rate (we treat the system occupation rate as CPU load in the evaluation of this paper), the number of false positive samples is $|fp|$, the number of true positive samples is $|tp|$, the number of false negative samples is $|fn|$, and the number of true negative samples is $|tn|$. We define a policy transition threshold value $\alpha$ 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 $\alpha$ as 0.75.

When $\gamma < \alpha$, we set the reward function $R(s)$ for single-targeted actions as follow:

$$R(s) = \begin{cases} -2 & False\ positive\ sample \\ 1 & True\ positive\ sample \\ -1 & False\ negative\ sample \\ 0 & True\ negative\ sample \end{cases}$$

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

In the scenario that the agent is making multiple-targeted actions, and $\gamma < \alpha$, we set the reward function as $R(s)$ follow:

$$R(s) = \eta(-2|fp| + |tp| - |fn|)$$

Where $\eta$ is the reward multiples. We can set $\eta$ as a value more one so that the agent would get extra rewards or penalties when making multiple-targeted actions. The larger $\eta$ 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 RL agent because monitoring a large amount of incoming requests is an expensive operation and could become the a system vulnerability itself. The agent can fix this issue 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 $R(s)$ for single-targeted actions as follow:

$$R(s) = \begin{cases} -\frac{2}{(\frac{\gamma}{\alpha})^g} & \textit{False positive sample} \\ (\frac{\gamma}{\alpha})^g & \textit{True positive sample} \\ -(\frac{\gamma}{\alpha})^g & \textit{False negative sample} \\ 0 & \textit{True negative sample} \end{cases}$$

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 **??** 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 $\gamma$. The agent will get less and less penalties from false positive samples with the increasing of $\gamma$. 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 requests as possible, with the cost of sacrificing a little bit false positive rate.

In the scenario that the agent is making multiple-targeted actions, and $\gamma \geq \alpha$, we set the reward function as $R(s)$ follow:

$$R(s) = \eta(-\frac{2}{(\frac{\gamma}{\alpha})^g}|fp| + (\frac{\gamma}{\alpha})^g|tp| - -(\frac{\gamma}{\alpha})^g|fn|)$$

$R(s)$ is in direct proportion to the summation of reward values that returned by all the affected requests. Still, we use the reward multiples parameter $\eta$ to encourage the agent to take multiple-targeted actions rather than single-targeted actions.

## 4.6 Training

The training of the deep reinforcement learning agent follows Q-value iteration [33]. 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 $i$, assume that the agent is located in $s_i''$ and receives a request state $s_i'$, then it selects an action $a_i$, observes a reward $r_i$, enters a new environmental state $s_{i+1}''$, and $Q$ is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information:

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

where $\beta$ is the learning rate, and $\zeta$ is the discount factor.

However, there is an intractable problem in this approach. 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 simulation and training phases. In order to handle this issue, we use a deep neural network [14] to serve as a likelihood function to estimate 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 one input layer, three hidden layers, and one output layer in the neural network, where the input layer has 12 nodes to import the twelve-dimensional vector $s$, and the output layer has six nodes to generate the recommendation rates for six possible actions respectively. The first and third hidden layers have 14 nodes, while the second hidden layer has 15 nodes. There is one crucial design detail in the neural network that the first layer and second layer are not fully connected. Instead, we separate the nodes for $s'$ from $s''$ to make sure the neural network can treat the two sub-state vectors differently.

In order to logically represent this neural network and conduct training, we express this as a formula that we can perform optimizations on. Just as the training of ordinary neural networks, we define a loss function in the training phase. The loss is just a value that indicates how far our action is from the actual target:

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

where $\epsilon$ is the decay rate, and $r + \epsilon \max_{\acute{a}} \hat{Q}(s, \acute{a})$ is the actual target. Then we just continuously train this to minimize the loss.

There is one thing to be noticed is that state $s = s' \cup s''$, and the agent will continuously get environmental state $s''$. But it will only get request state $s'$ when there is an incoming application request. So, the agent will only be activated when it receives $s''$ in both training and detection phases.

# 5 EVALUATIONS

## 5.1 Implementation and Simulations

We utilized Open vSwitch [19] and Mininet [13] 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.

We simulated a victim system by constructing a 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 a HTTP-based API that can read its hard disk and return selected images, which is the designed performance bottleneck for attackers to exploit.

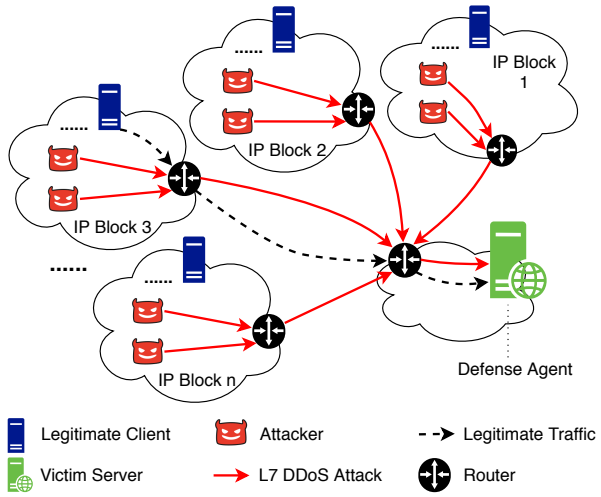



**Figure 5: Topology of Simulated Network Environment**

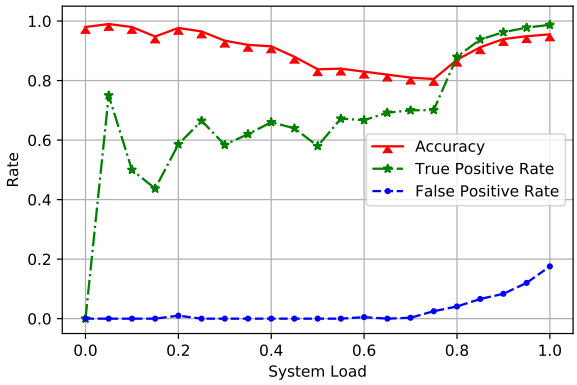



**Figure 6: Performance Metrics of different system loads**

For the simulation of L7 DDoS attacks, we utilized some well-recognized simulation software from public repositories to generate attack traffic that follow the threat models in Section 3.1. We used the Application Layer DDoS Simulator [30] to simulate request-flooding attacks by generating HTTP flood and SMTP flood. For leveraged attacks, we used Slowloris [24] to simulate the most typical leveraged attack, low and slow attack. In the end, we used modified HULK program [26] to generate lethal attacks, which creates malicious requests that target the above designed performance bottleneck in the victim server.

For the simulation of legitimate application requests, we developed several home-grown scripts to randomly send diverse requests to the victim server with relatively long interval time. We also tuned hyper-parameters of the scripts in a way that the legitimate requests would not cause significant performance degradation of the victim servers.

We constructed the reinforcement-learning-based L7 DDoS attack defense system with OpenAI Gym [3] and Keras [4]. Among these tools, Gym is used for environment formulation, agent construction, and test benchmarks. We used Keras particularly for building the deep neural network and model training.

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 $\alpha$, we set it to be 0.75.
- The learning rate $\beta$ for agent training is 0.25 in this implementation.
- For the hazard index $g$, we set it to be 3.

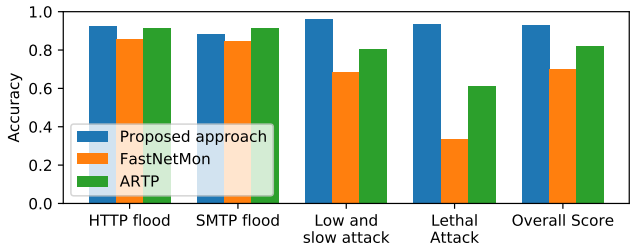
## 5.2 Ability of Detecting Attacks

Although the proposed method does not generate the precise labels of incoming requests, we can still evaluate its ability of identifying L7 DDoS attacks by inferring the output action $a$. In Section 4.4, we designed six types of action $a$ to handle the application requests. In order for straightforward evaluation results, we count requests that generate action $a_i$ and $a_{ii}$ as legitimate requests. Conversely, we count requests that generate action $a_{iii}$, $a_{iv}$, $a_v$, and $a_{vi}$ as malicious requests. All the evaluation metrics in this section are based on this regulation.

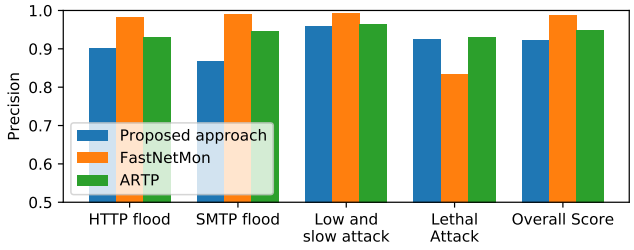
*5.2.1 Detection Accuracy.* After the defense agent was well-trained, we simulated benign requests and launched the L7 DDoS attacks to the victim simultaneously for evaluating the agent's accuracy of identifying L7 DDoS attacks. During the test, we firstly ensured the volume of legitimate requests was always under the victim server's capacity so that the server would not crash due to legitimate activities. Afterwards, we adjusted the volume of application requests to test the performance of this approach with different system loads. Here, we consider the system load as the occupancy rate of the system bottleneck. For example, if the memory space is the bottleneck of the victim system, then we treat the occupancy rate of memory as the system load. There was one problem, we found that it was challenging to simulate the scenarios that the system has extremely low loads because both the victim server and the reinforcement learning agent were operating in one single node, making the virtual machine have nearly 30% system load initially. In order to get fix this problem and fetch the evaluation results from low-system-load scenarios, we input the defense agent with fake system utilization information during the beginning part of the evaluation to cover the scenarios that the system load is lower than 35%.

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

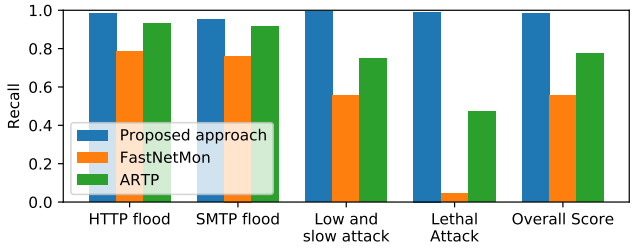
When the system load is at meager rates, we can get nearly 1.0 accuracy for the following two reasons: 1. Based on the heuristics of the reward functions, the agent will minimize the false positive rate at this point; 2. There are few requests with malicious intentions at this time point, and the majority of the requests are benign.



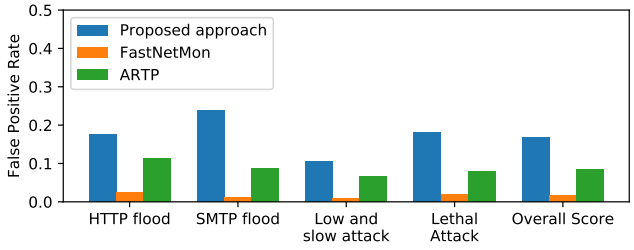

**(a) Accuracy**




**(b) Precision**




**(c) Recall**




**(d) False Positive Rate**

**Figure 7: Results of Comparison Evaluation**

However, when both the system workload and the volume of attacks are increasing, the accuracy has some apparent drops. Because although the attack volume increased, the defense agent still uses the defense strange that aiming at minimizing the false positive, guiding the agent to sacrifice the true positive rate for letting the victim server adequately processes most of the legitimate requests. Thus, the false positive rate remains approximately zero within this zone. On the contrary, the trend for true positive rate is somehow tanglesome in low-system-load scenarios, because the volume of malicious requests is still low, making it hard to reach statistical significance. Nonetheless, we can still obviously see the defense agent takes true positive rate with indifference at these moments.

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 recognize as many attacks as it can 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 immediately, sacrificing a little bit false positive rate, but still increases the overall accuracy. At 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 defense the L7 DDoS attacks, and the detection accuracy of the tactics is satisfactory.

*5.2.2 Comparison Evaluation.* We also compared our approach with two other L7 DDoS attack detection approaches. One of them is FastNetMon [18], a widely-used commercial DDoS detection software that applies statistical detection methods. Although this software is not targeted on L7 DDoS attacks, it offers good performance on general DDoS detection. Another detection model we used in this evaluation is ARTP [20], as stated in Section 2.1, it is a machine-learning-based detection approach particularly designed for L7 DDoS attacks.

We simulated the traffic of HTTP flood, SMTP flood, low and slow attacks, and lethal attacks with the methods described in Section 5.1. The intensities of these simulated attacks were around two times of the victim server's capacity. Then, we replayed the traffic and applied these methods to generate the performance metrics of the detection results.

Figure 7a shows the accuracies of these approaches. The proposed 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. As for FastNetMon, it achieves reasonable accuracy scores on detecting request flooding attacks but gets low performance on identifying leveraged and lethal attacks. Because FastNetMon is not an L7-specific DDoS detection software and it cannot dig more information other than traffic rates. Although our proposed approach does not have perfect scores in precision and false positive rate (as shown in Figure 7b and Figure 7d), 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 7c, the proposed method achieves decent recall scores, which means it successfully identifies the majority of the malicious requests. On the contrary, FastNetMon gets very low false positive rates, but it fails to identify a large proportion of low and slow attacks and lethal attacks, making it unusable during the peaks of attacks. The ARTP model has poor performance on detecting lethal attacks as well, because it cannot adjust the detection strategies dynamically based on the system occupancy.

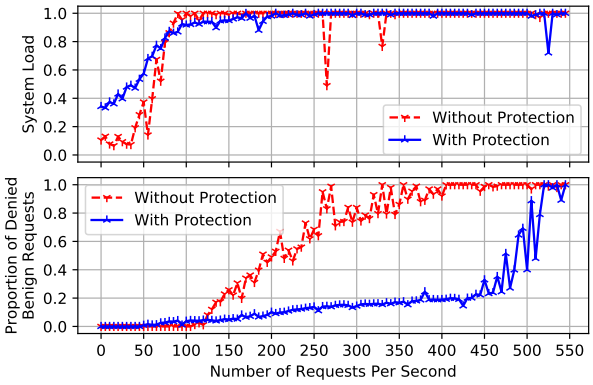



**Figure 8: Efficacy of Attack Mitigation**

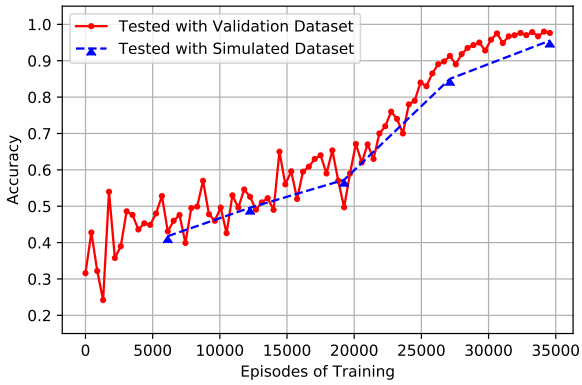



**Figure 9: Convergence Trend while Training**

## 5.3 Mitigation Efficacy

We evaluate this approach's efficacy of L7 DDoS attack mitigation in this subsection. Figure 8 shows the efficacy results, where the x-axis represents the number of applications requests made to the victim server per second, including both the legitimate requests and illegitimate requests. The y-axis of the upper sub-plot represents the system load, while the y-axis of the lower sub-plot represents the proportion of denied benign requests.

In order to generate this figure, we first simulated both benign and malicious requests towards the victim server. Just as in section 5.2, the volume of benign requests was always under the victim server's capacity, in this case, peaked at 100 requests per second. Then, we increased the volume of requests to record the system loads and the proportions of denied benign requests. Finally, we replayed the same requests towards the same victim server but with the protection of the defense agent, recording the same type of data along the process.

Initially, the resource consumption of the server without protection is lower than the server with the agent's protect, because the deployment of the defense agent costs a certain amount of computing resource, especially for maintaining the LSH function, request monitoring, and the operation of the deep neural network. However, this consumption will payback shortly with the increasing number of receiving requests. We can see that the proportion of denied benign requests 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 requests is lower than 20%, then the capability of the server without protection is approximately 140 requests per second. After reaching 250 requests per second, the server without protection is almost useless, with the majority of requests getting denied. While for the server with the defense agent's protection, the deny rate of legitimate requests goes higher than 20% only after the number of requests 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.

## 5.4 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 time consumption for agent training. Figure 9 shows the convergence trend while model training, where the x-axis represents the number of episodes, and the y-axis represent the accuracy value. We obtained the accuracy values by utilizing two types of testing datasets. One is a preset validation dataset, which provides straightforward test criteria, but the contents of the dataset always remain the same. Another dataset is the real-time simulation of both attack traffic and legitimate traffic, which offers practical evaluation results and enables us to inspect whether the training model is getting overfitting.

As we can see in the figure, 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, making the defense agent into decent protection for the victim server. Eventually, the accuracy of the agent stabilizes near 96% after 30,000 episodes of training. This evaluation also proofs that it is feasible to retrain the defense agent within half a week to fit a whole new environment in a real deployment.

## 5.5 Service Delay

Since the defense agent will continuously inspect all the incoming application requests 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 10).

Here, we define the length of delay as the time duration from sending out a request to receiving the whole reply. It is usual for an HTTP server to have around 0.5 seconds' delay when responding to clients' requests. As we can see in the Figure 10, 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 during

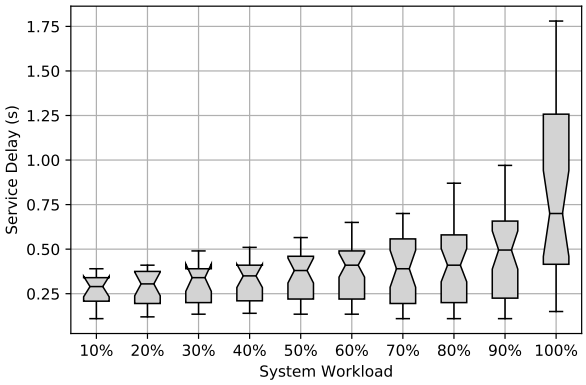



**Figure 10: Box Plot for Service Delay**

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). While the system without any protections is already in an unusable condition under this circumstance.

## 6 LIMITATIONS AND OPEN ISSUES

In most cases, this reinforcement-learning-based approach can accurately identify L7 DDoS attacks and generate defense tactics simultaneously. However, it has some limitations:

(1) Application-layer DDoS attack is highly dependent on the victim system and the network environment, so are the attack detection and defense solutions. Thus, this approach may require us to retrain the agent to accommodate the current victim system when the environment is changed dramatically.
(2) This method is a learning-based approach, although the training data of this approach can be enhanced to cover more attack models, thus improving the capability of this approach, nonetheless, if the training does not include information from zero-day attacks, this method probably will not be able to deal with them.

This approach also faces several open issues as possible future working items:

(1) Currently, we mainly use HTTP flood, SMTP flood, low and slow attack, and API-based lethal attack to train the agent. However, there are still many other types of L7 DDoS attacks existing such as XML-based attack. It would be a meaningful improvement to simulate more types of attacks in the training phase to make the reinforcement learning agent more robust.
(2) The performance bottleneck of this approach is in the victim server, because the victim server takes charge of both the message decoding and environmental information transmission. We can further increase the efficiency of the RL agent by using more advanced data structures or algorithms.
(3) As a learning-based approach, this method could be vulnerable to adversarial attacks. It is significative to develop some systematic methodologies to evaluate and improve the robustness of this method.

## 7 CONCLUSIONS

L7 DDoS attacks are becoming far more sophisticated and threatening than before. Compared with traditional DDoS attacks, L7 DDoS are effective in overwhelming victim servers but difficult for conventional DDoS approaches to detect and mitigate. This paper proposes a reinforcement-learning-based approach, thus able to 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. Moreover, different from typical DDoS detection approaches that label the traffic as either legitimate or malicious, this approach 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 enough. Evaluation shows that this approach protects a victim server from L7 DDoS attacks effectively; it can detect 98.73% of the L7 DDoS traffic flows at the peak system load, mitigating most of them.

## REFERENCES

[1] Sharad Agarwal, Travis Dawson, and Christos Tryfonas. 2003. *DDoS mitigation via regional cleaning centers.* Technical Report. Sprint ATL Research Report RR04-ATL-013177.
[2] Sebastian Anthony. March 30, 2015. GitHub battles "largest DDoS" in site's history, targeted at anti-censorship tools. https://arstechnica.com/.
[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
[4] François Chollet et al. 2015. Keras.
[5] Cloudflare. [n.d.]. Application Layer DDoS Attack (Resources on Cyber-Security and How the Internet Works From Cloudflare). https://https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/.
[6] Cloudflare. [n.d.]. What Is A Low And Slow Attack? (Resources on Cyber-Security and How the Internet Works From Cloudflare). https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/.
[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 253–262.
[8] Christos Douligeris and Aikaterini Mitrokotsa. 2004. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks* 44, 5 (2004), 643–666.
[9] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. 2014. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2037–2064.
[10] Ronald A Howard. 1960. Dynamic programming and markov processes. (1960).
[11] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.
[12] Vitaly Kupreev, Ekaterina Badovskaya, and Alexander Gutnikov. May 21, 2019. DDoS attacks in Q1 2019. https://securelist.com/.
[13] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*. ACM, New York, NY, USA, Article 19, 6 pages. https://doi.org/10.1145/1868447.1868466
[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.
[15] George E Monahan. 1982. State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science* 28, 1 (1982), 1–16.
[16] Maryam M Najafabadi, Taghi M Khoshgoftaar, Amri Napolitano, and Charles Wheelus. 2016. Rudy attack: Detection at the network level and its important features. In *The twenty-ninth international flairs conference.*

[17] Jema David Ndibwile, A Govardhan, Kazuya Okada, and Youki Kadobayashi. 2015. Web Server protection against application layer DDoS attacks using machine learning and traffic authentication. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, Vol. 3. IEEE, 261–267.

[18] P Odintsov. [n.d.]. FastNetMon - very fast DDoS analyzer with sflow/netflow/mirror support. *Dostupné z: https://github. com/pavel-odintsov/fastnetmon/* ([n. d.]).

[19] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. 2015. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 117–130.

[20] K Munivara Prasad, A Rama Mohan Reddy, and K Venugopal Rao. 2016. 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).

[21] Amit Praseed and P Santhi Thilagam. 2018. DDoS attacks at the application layer: Challenges and research perspectives for safeguarding Web applications. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 661–685.

[22] Sivaramakrishnan Ramanathan, Jelena Mirkovic, Minlan Yu, and Ying Zhang. 2018. SENSS Against Volumetric DDoS Attacks. In *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 266–277.

[23] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, and Edward W Knightly. 2006. DDoS-Resilient Scheduling to Counter Application Layer Attacks Under Imperfect Detection.. In *INFOCOM*. Citeseer.

[24] RSnake, John Kinsella, Hugo Gonzalez, and Robert E Lee. 2009. Slowloris HTTP DoS. https://web.archive.org/web/20150426090206/http://ha.ckers.org/slowloris.

[25] Stefan Seufert and Darragh O'Brien. 2007. Machine learning for automatic defence against distributed denial of service attacks. In *2007 IEEE International Conference on Communications*. IEEE, 1217–1222.

[26] Barry Shteiman. 2017. Hulk DoS tool. https://github.com/grafov/hulk.

[27] Vitaly Simonovich. July 24, 2019. Imperva Blocks Our Largest DDoS L7/Brute Force Attack Ever (Peaking at 292,000 RPS). https://www.imperva.com/blog/.

[28] Sujatha Sivabalan and PJ Radcliffe. 2013. A novel framework to detect and block DDoS attack at the application layer. In *IEEE 2013 Tencon-Spring*. IEEE, 578–582.

[29] Jared M Smith and Max Schuchard. 2018. Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive BGP routing. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 599–617.

[30] STORMSECURITY. [n.d.]. Application Layer DDoS Simulator. https://stormsecurity.wordpress.com/2009/03/03/application-layer-ddos-simulator.

[31] Suriadi Suriadi, Douglas Stebila, Andrew Clark, and Hua Liu. 2011. Defending web services against denial of service attacks using client puzzles. In *2011 IEEE International Conference on Web Services*. IEEE, 25–32.

[32] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. 2003. CAPTCHA: Using hard AI problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 294–311.

[33] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[34] Y. Xie and S. Yu. 2009. Monitoring the Application-Layer DDoS Attacks for Popular Websites. *IEEE/ACM Transactions on Networking* 17, 1 (Feb 2009), 15–25. https://doi.org/10.1109/TNET.2008.925628

[35] Yi Xie and Shun-Zheng Yu. 2006. A novel model for detecting application layer DDoS attacks. In *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)*, Vol. 2. IEEE, 56–63.

[36] Chuan Xu, Guofeng Zhao, Gaogang Xie, and Shui Yu. 2014. Detection on application layer DDoS using random walk model. In *2014 IEEE International Conference on Communications (ICC)*. IEEE, 707–712.

[37] Satyajit Yadav and Selvakumar Subramanian. 2016. 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)*. IEEE, 361–366.

[38] T. Yatagai, T. Isohara, and I. Sasase. 2007. 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*. 232–235. https://doi.org/10.1109/PACRIM.2007.4313218

[39] Shun-Zheng Yu and Yi Xie. 2009. A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors. *IEEE/ACM transactions on networking* 17, 1 (2009), 54–65.

[40] Heng Zhang, Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. 2016. Sentry: A novel approach for mitigating application layer DDoS threats. In *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 465–472.