Exploring New Opportunities to Defeat Low-Rate DDoS Attack in Container-Based Cloud Environment

Zhi Li, Hai Jin, *Fellow, IEEE*, Deqing Zou, and Bin Yuan, *Member, IEEE*

Abstract—DDoS attacks are rampant in cloud environments and continually evolve into more sophisticated and intelligent modalities, such as low-rate DDoS attacks. But meanwhile, the cloud environment is also developing in constant. Now container technology and microservice architecture are widely applied in cloud environment and compose container-based cloud environment. Comparing with traditional cloud environments, the container-based cloud environment is more lightweight in virtualization and more flexible in scaling service. Naturally, a question that arises is whether these new features of container-based cloud environment will bring new possibilities to defeat DDoS attacks. In this paper, we establish a mathematical model based on queueing theory to analyze the strengths and weaknesses of the container-based cloud environment in defeating low-rate DDoS attack. Based on this, we propose a dynamic DDoS mitigation strategy, which can dynamically regulate the number of container instances serving for different users and coordinate the resource allocation for these instances to maximize the quality of service. And extensive simulations and testbed-based experiments demonstrate our strategy can make the limited system resources be utilized sufficiently to maintain the quality of service acceptable and defeat DDoS attack effectively in the container-based cloud environment.

Index Terms—Container, microservice, DDoS attack, mitigation, cloud computing

1 Introduction

In this paper, we attempt to explore new solutions to overcome DDoS attacks in container-based cloud environment. Nowadays, the overall number of DDoS attacks is growing every year [1]. Not only that, with the development of cloud environment, the DDoS attacks also have lots of changes in scale, methods, and aims [2]. But in essence, the key issue of DDoS attack and defense is still the resources competition [3], [4], [5]: the party that can effectively control more resources (CPU, memory, and network bandwidth, etc.) is the winner of this battle. At present, due to the lightweight features, container-based cloud environment has been rapidly developed and widely applied. The combination of container technology and microservice architecture makes the container-based cloud environment more effective and agile in resources usage. In this case, we will discuss the new opportunities to mitiDDoS attacks in the container-based cloud environment.

In traditional cloud environment with the monolithic architecture, the web application is tightly coupled and runs as an independent instance on the virtual machine (VM) [6]. It means that if a component of the application experiences the influence of DDoS attacks, the entire instance must be scaled to overcome the DDoS attacks. Doubtlessly, this kind of resources usage mode is coarse-grained and could be a fatal issue to the individual cloud customers who have limited resources to fight with DDoS attacks. Also, to the other cloud customers who obtain resources on-demand with "pay-as-you-go" business model [7], it may sharply drive the progress of Economic Denial of Sustainability (EDoS) attack [8], [9].

Comparing with VM, the container with lightweight virtualization can use fewer resources to scale service instances and achieve the same effect of DDoS attack mitigation as VM-based cloud environments. Furthermore, different from monolithic architecture, the microservice architecture applied in container-based cloud environment also provides a flexible option to service scaling. In microservice architecture, an application is split into a set of components, and each of them run as an independent service called microservice. The function in application will be performed by a series of microservices forming a chain, and the chains often share microservices. Once a microservice experiencing the influence of DDoS attacks, its instances can be scaled independently [10]. In practice, considering the economic cost, the development teams will estimate the expected growth scale of the microservice based on quantitative and qualitative analysis, and reserve sufficient resources to support it scaling, which is called as *Capacity Planning* [11], [12]. According to the capacity planning, although the microservice has enough resources to cope with the spike in

• *The authors are with the National Engineering Research Center for Big Data Technology and System, Cluster and Grid Computing Lab, Services Computing Technology and System Lab, Big Data Security Engineering Research Center, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {lizhi16, hjin, Deqingzou, yuanbin}@hust.edu.cn.*

Manuscript received 4 Oct. 2018; revised 10 Sept. 2019; accepted 16 Sept. 2019. Date of publication 20 Sept. 2019; date of current version 10 Jan. 2020. (Corresponding author: Hai Jin.)
Recommended for acceptance by F. Qin.
Digital Object Identifier no. 10.1109/TPDS.2019.2942591

demand, it also means that each microservice only has limited resources to battle with DDoS attacks.

Fortunately, for flood-based DDoS attack, many intrusion detection mechanisms based on packets signature or network behavior are constantly evolving [13], [14], [15], [16], [17] and can also be applied to container-based cloud environment. Nevertheless, some intelligent attackers could launch the low-rate DDoS attack in a stealthy fashion to elude the detection mechanisms [18]. In this kind of attack, attackers exploit potential logic errors or vulnerabilities in the service to elaborately construct malicious requests [19]. Each malicious request can consume significantly greater resources than the benign, meanwhile, the rate and number of malicious requests are all slightly less than the thresholds of intrusion detection. On the basis of these characteristics, the low-rate DDoS attack can achieve DoS attack stealthily.

Limited by the effectiveness of detection mechanisms, many mitigation strategies have been proposed to cope with DDoS attacks. The resource-scaling mechanism [7], [20], [21] is one of them, which automatically extends additional resources to help services survive the DDoS attacks. Also, the isolation mechanism [22], [23] isolates the victim services from DDoS attacks to mitigate the influence of DDoS attacks. And the limitation mechanism [24] limits the resources usage of victim services to ensure the usability of other services. With these state-of-art solutions, it is possible to mitigate the low-rate DDoS attack successfully. However, the researches about these mitigation mechanisms are all based on VM-based cloud environments. Due to the difference between the container-based and the VM-based cloud environment, these mechanisms would not have the prospective effect or even can't deal with the low-rate DDoS attack in microservice architecture.

Specifically, the loosely coupled microservice architecture facilitates scaling the microservice affected by DDoS attacks independently. But in this case, constantly increasing the scale of a microservice will make its dependent services overload with great probability. And meanwhile, owing to the sharing of microservices [25], the consequence of DDoS attacks will affect the other microservices in a larger scope. Further, due to the OS-level virtualization of container, there may have serious competition for resources between plenty of microservice instances [26], [27]. However, the existing mitigation mechanisms in VM-based cloud environment didn't consider these issues in their theory models or solutions. Therefore, we focus on the low-rate DDoS attack in container-based cloud environment and explore the appropriate mitigation mechanism according to the actual situation in container-based cloud environment.

In this paper, we establish a mathematical model based on queueing theory to formalize and analyze the low-rate DDoS attack scenario in container-based cloud environment. Based on the results of these analyses, we point out the strengths and weaknesses of the container-based cloud environment in defeating the low-rate DDoS attack and propose a dynamic DDoS mitigation mechanism according to the features of container-based cloud environment.

With the mitigation mechanism, we divide requests to the microservice into two parts: the whitelist and the unknown part. To the whitelist requests, the target of mitigation mechanism is using the minimum amount of resources to maintain the acceptable QoS of them. Further, using the remaining resources increases the serving rate to the unknown requests as much as possible for helping them survive the low-rate DDoS attack. Specifically, to achieve these targets, we dynamically optimize the number of container instances serving for the whitelist and unknown requests, and regulate the resources allocation for each instance to make system resources be utilized effectively. Finally, we conduct a set of testbed-based and simulation-based experiments to demonstrate the correctness of our analysis model and the effectiveness of our DDoS mitigation mechanism.

To the best of our knowledge, this paper is the first work to study the superiorities and limitations for the new features of container-based cloud environment in defeating the low-rate DDoS attack. And we propose a dynamic DDoS mitigation mechanism to defeat the low-rate DDoS attack in container-based cloud environment. More specifically, our contributions are:

- We explore the possibility that utilizing the new features in container-based cloud environment defeats the low-rate DDoS attack. And we point out the strengths and weaknesses to mitigate low-rate DDoS attack in the container-based cloud environment.
- We establish a mathematical model based on queueing theory to formalize the low-rate DDoS attack scenario in container-based cloud environment and analyze the capacity of container-based cloud environment in defeating against low-rate DDoS attack.
- Guided by this model, we propose a dynamic mitigation mechanism to optimize and coordinate the resource allocation and the number of containers for mitigating the low-rate DDoS attack.

The remainder of this paper is organized as follows. We introduce the related work in Section 2. In Section 3, we review the features of container technology and microservice architecture in brief. In Section 4, we discuss the design of DDoS mitigation mechanism. We present system modeling detail in Section 5. Performance evaluations are shown in Section 6. And we discuss the future research points in Section 7. Finally, we summarize this paper in Section 8.

2 Related Work

2.1 DDoS Attacks Mitigation in Cloud Environment

The main aim of DDoS attacks is exhausting the resources of victims, such as networking resources or computing resources. When DDoS attacks spreading to cloud environment, a series of solutions have been proposed to defeat them based on the features of cloud environment.

Du et al. [28] explored an idea using abundant resources in cloud environment to defense DDoS attacks. And Yu et al. [7] proposed a dynamic resources allocation strategy based on queueing theory to mitigate the DDoS attacks with idle resources in cloud environment. Due to the resources are not free in cloud environment, the DDoS attacks gradually evolved into EDoS attacks that aim to the economic resources of victims. For the EDoS attacks, Sqalli et al. [29] proposed a filtering approach based on graphical Turing tests to create a virtual firewall for filtering traffic. Additionally, Amazon has provided a service, CloudWatch [30],

monitoring the cloud resources in real-time to limit the scaling upper and reduce the impact of EDoS attacks.

Except for investing the torrent of resources to mitigate DDoS attacks, the methods of *victim migration* and *resource management* are also used to mitigate the DDoS attacks in cloud environment. To the victim migration methods, Latanicki et al. [31] presented a cloud level detection method to identify malicious VMs and migrate the victim VM to other physical servers. To the resource management methods, Somani et al. [32], [33] proposed service resizing methods that use OS-level controls to constrain or isolate the system resources usage of the victim services.

Unfortunately, few studies focus on DDoS attacks mitigation methods in the container-based cloud environment. So far, we have found only one related work that Ye et al. [34] used deep learning to extract features of DDoS attacks in container-based cloud system and detect DDoS attacks based on it.

2.2 low-Rate DDoS Mitigation in Cloud Environment

The typical features of the flood-based DDoS attack, such as high-rate and heavy-flow, are not presented in low-rate DDoS attack, which makes traditional detecting mechanisms failure. For this problem, researchers have proposed many detecting mechanisms against the low-rate DDoS attack, and these detecting mechanisms can be classified into two categories: based on the network traffic and based on the application vulnerabilities.

To the mechanisms based on network traffic, Xiang et al. [35] used information metrics containing generalized entropy and information distance to detect low-rate DDoS attack. Luo et al. [36] proposed a mathematical model based on behaviors of victim TCPs congestion for detecting low-rate DDoS attack. Further, Wu et al. [37] established a mathematical model combined MF-DFA algorithm with Holder exponent to differentiate the malicious traffic and the normal traffic in low-rate DDoS attack.

An alternative approach against with low-rate DDoS attack is to nip the forming causes of the attacks in the bud. It means that the vulnerabilities and logical errors in the service, which can cause the low-rate DDoS attack, need to be detected and repaired before it runs. Burnim et al. [38] presented a complexity testing method based on dynamic analysis to detect logical errors in performance. Olivo et al. [19] proposed a static analysis method and implemented it into *Torpedo* to detect the vulnerabilities of DoS attacks in web applications. Although these methods can help us wipe out lots of low-rate DDoS attacks, they also exist false positives and false negatives. Thus, the mitigation mechanisms to the low-rate DDoS attack are still necessary.

2.3 Security of Container-Based Cloud Environment

Nowadays, security issues in the container-based cloud environment are focusing on the security drift problem and isolation problem. To the security drift problem, Byungchul et al. [39] performed analyses on the security issues brought by the high degree of agility, reusability, and portability with container. For the security issues about isolation, Xing et al. [40] systematically identified the information leakage problem and investigated potential container-based power attack threats built upon these leakage channels. Sergei et al. [41] used the SGX to enhance the isolation between containers and protect containers from outside attacks.

Previous researches about the security of containers focus on investigating and solving new security issues in container-based cloud environment, such as how to enhance the isolation to achieve the same security guarantees as hardware virtualization. However, there doesn't have researchers to study the new features of container-based cloud environment whether influence the traditional security issues. If so, are these influences positive or negative?

3 Preliminary Knowledge

In this section, we point out the key characteristics of container technology and microservice architecture, which make up the container-based cloud environment, to guide system modeling in next.

3.1 Container Technology

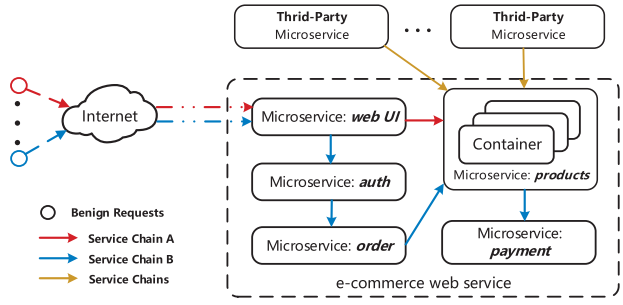
As the OS-level virtualization technology, container presents an alternative to the VM in cloud environment. Unlike VMs running the whole OS on virtual device, containers share kernel with the host system and support minimum runtime requirements of the application. Due to the difference in the level of virtualization between VM and container, containers depend more on kernel features such as namespace and control groups (cgroups) to achieve isolation and resource control instead of requiring hypervisors. With the namespace, processes in a container are isolated from other containers and host system. Moreover, cgroups use scheduler features in the kernel to control the amount and priority of resources usage for each container. Therefore, combining with these kernel features can achieve the fine-grained runtime isolation and resources control to containers.

Owing to lightweight features, container has faster startup times, superior performance of I/O throughput and lower latency than VM. Also, container needs fewer system resources to run, thus a single server can host far more containers than VMs. Besides the superiorities in performance, container also provides a consistent and portable software environment for development, testing, and production of applications. It can greatly shorten the release and update cycles of the application and make applications ignore the difference between cloud platforms to easily run and scale anywhere without compatibility issues.

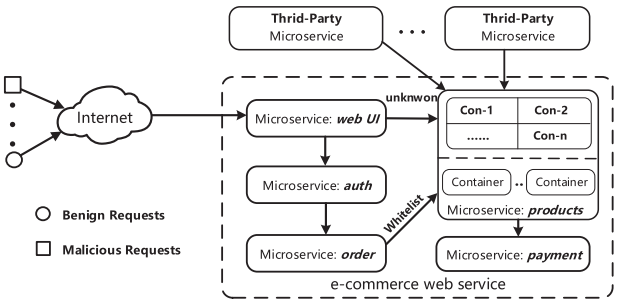
With the rapid popularity of containers and the continuous development of container ecosystem, container technology is widely used in various cloud platforms, including Amazon Web Services (AWS) [42], IBM Cloud [43] and Azure [44], etc. In the meantime, container has gradually become the basic unit of resource allocation and scheduling in the cloud platforms. To orchestrate containers in cloud environment more efficiently, many open-source projects, such as Docker Swarm [45] and Kubernetes [46], are developed to deploy, manage and schedule containerized applications.

3.2 Microservice Architecture

The aforementioned characteristics of container technology also promote the development of microservice architecture



(a) Cloud environment under nonattack



(b) Cloud environment under DDoS attack

Fig. 1. DDoS mitigation mechanism in the container-based cloud environment.

[47], where the application is decoupled into a set of independent services called microservices.

Comparing to traditional service architecture, the microservices are very loosely coupled with one another. In microservice architecture, each microservice is a simple RESTful web service [48], which performs a simple function and serves as a single purpose. In general, a set of microservices interact with each other using HTTP and form a service chain to provide an integrated function. In this case, the functions in a traditional monolithic application will be served by multiple service chains.

Therefore, in microservice architecture, the microservice can be scaled on demand to satisfy growing workloads in partial functions. However, most microservices do not live in isolation. Constantly scaling a microservice will overload its dependent services with great probability. One solution is ensuring that all dependencies of the microservice are scalable and will be scaled synchronously. However, considering economic cost, another solution is capacity planning. In this solution, development teams will foresee the expected growth of the microservice according to quantitative and qualitative analyses for the microservice function [12]. Further, they will reserve the resources that could meet expected growth of the microservice to limit its scale.

Moreover, due to the loosely coupled feature, each microservice is developed by a single team independently. In this way, the development and update for the microservices will have a lower failure rate and a shorter release cycle. And the development teams can delivery the release more flexible and frequently. Moreover, benefited from the certainty of microservices' function, each microservice can be developed with different tools or languages. As a consequence, the development teams have more choices to develop microservices, meanwhile, the microservice can also be easily reused by other services.

Furthermore, the fault-tolerant capability of the microservice architecture is stronger than the monolithic architecture where the failures in a single component will cause the entire application broken. In contrast, with microservice architecture, the failures shown in a single microservice only lead to the functionality degrading rather than crash the entire application. Due to these superiorities of microservice architecture, lots of companies, such as Uber and Netflix, etc., have transformed their business into the microservice architecture [49], [50].

4 DDoS Mitigation Mechanism

In this section, we propose a dynamic DDoS mitigation mechanism to maintain microservice availability under the low-rate DDoS attack and maximize its QoS with limited system resources.

First of all, we examine the features of container-based cloud environment. Fig. 1a shows partial microservices making up an e-commerce website. In the microservice system, users' requests will be served by a series of microservices forming a chain, and the chains often share microservices. Besides the microservices inside same application, each microservice can also be called by third-party services. In this case, the requests arrived at a microservice generally have different sources and credibility. As shown in Fig. 1a, chains A and B traverse the *products* microservice simultaneously. Any users can browse the products without authentication through chain A, but in chain B, only authenticated users can access the products to proceed payment.

When the DDoS attack occurs, the malicious requests can reach the target microservice with the service chains having weak authentication. And the performance of all service chains containing this microservice would be affected. To protect the benign users, we whitelist the requests from creditable service chains, which only serve for the authorized users, and allow them to access the attacked microservice with a higher priority. Unfortunately, due to the poor performance isolation between containers [26], [27], it is still inevitable that the whitelist requests in serving will compete for the resources with subsequent malicious requests under heavy workloads. Although throttling the requests outside whitelist may avoid the competition and block the DDoS attack with great probability, it will also drop lots of normal requests and cause the corresponding service chains cascading failure [48].

To this end, we propose a DDoS mitigation mechanism to dynamically regulate and divide resources for handling the whitelist and other unknown requests respectively. As presented in Fig. 1b, each microservice has a set of instances running in containers. When the microservice suffering from the low-rate DDoS attack, its instances will be divided into two isolated parts according to the assigned resources. One part handles the requests from whitelist, and another part serves for the unknown requests composed by malicious requests and benign requests.

To be specific, the mitigation mechanism is described in Algorithm 1. To the whitelist users, the mitigation mechanism will calculate the minimum resources R_{wc} and the optimum number of containers c_w needed for them, which can maintain its QoS acceptable and guarantee the resources being utilized

TABLE 1
Notations we Use when Modeling and Analyzing the System

R	total amount of available resources for service	μ	service rate of the queue under nonattack scenario
R_w	available resource for serving the whitelist users	μ_w	service rate for the whitelist users of the queue
R_u	available resource for serving the unknown users	μ_u	service rate for the unknown users of the queue
R_c	available resource for a single container	r	ratio of malicious requests to the total requests
λ	arrival rate of the queue under nonattack scenario	β	ratio of service time for malicious packets to service time of benign requests
λ_w	arrival rate of the whitelist users in queue	ρ	ratio of the arrival rate and the service rate under nonattack scenario
λ_u	arrival rate of the unknown users in queue	π_i	probability of the system when there are i requests in the system
c_w	number of containers serving for whitelist users	c_u	number of containers serving for unknown users
T_w	average staying time of whitelist requests in system	T	average staying time of requests in system under nonattack scenario
T_u	average staying time of unknown requests in system	T'	acceptable staying time for unknown users

effectively. Further, the system resources R_{wc} will be divided and isolated in unit of containers to avoid resources competition between the whitelist and unknown requests. If the number of containers being serving the whitelist less than c_w, the mitigation mechanism will create a set of containers c'_w to complement. Otherwise, the redundant containers will be assigned to handle unknown requests. The more details of the strategy about resources assigning and containers' number optimizing will be discussed in Section 5.2.

Algorithm 1. DDoS Mitigation Method

```
1: //To the whitelist users
2: R_wc, c_w = AssignResources (Whitelist)
3: CreateIsolatedArea(R_wc)
4: WC = GetServingContainer (Whitelist)
5: if [ c'_w = c_w − num (WC) ] > 0 then
6:    for C ∈ WC do
7:       Container = SplitFromOld (C)
8:       AdjustResources (Container, R_wc)
9:    end for
10:   CreateContainer (c'_w, R_wc)
11: else
12:   for C ∈ C' = Select (WC, |c'_w|) do
13:      Container = SplitFromOld (C)
14:      AdjustResources (Container, R_wc)
15:   end for
16: end if
17:
18: //To the unknown users
19: R_uc = R_total - R_wc
20: while CalculateWaitingTime(T) do
21:   c_u = OptimizeContainerNumbers(T,R_uc)
22:   UC = GetServingContainer (Unknown)
23:   for C ∈ UC do
24:      AdjustResources (C, R_uc)
25:   end for
26:   if [ c'_u = c_u − num (UC) ] > 0 then
27:      CreateContainer (c'_u, R_uc)
28:   else
29:      C'_u = DeleteContainer (|c'_u|)
30:   end if
31:   Waiting (BaseTime)
32: end while
```

On the other hand, to the unknown requests, the remaining resources R_{uc} are used to help them survive the DDoS attack. With the average waiting time T of requests, the mitigation mechanism will calculate the optimal number of containers c_u to maximize the QoS of unknown requests. The calculation details are demonstrated in Section 5.3. Similar to the process in whitelist, the mitigation mechanism will dynamically add or delete the containers serving for the unknown requests to meet c_u. Also, for the ever-changing attack, this mitigating process will repeat periodically.

Furthermore, to the low-rate DDoS attack, the basic idea is to send sophisticated requests with a low rate for consuming large amounts of system resources. For bypassing the traffic detection mechanism, the rate and number of malicious traffic are all close to benign traffics. In this case, the mitigation mechanism mentioned above will be triggered when the resources consumption of service over the threshold but the network traffic keeping stable. For simplicity, the trigger threshold in our experiments is the same as common auto-scaling threshold that the usage of system resources is over 80 percent.

5 System Modeling And Analysis

In this section, based on the queueing theory [51], we establish an executable mathematical model to formalize the microservice system in container-based cloud environment. And we list all the notations we use in Table 1. According to this model, we will tackle two issues for the proposed mitigation mechanism: (1) how to maintain the acceptable QoS of whitelist requests using the least resources and the optimal number of containers, and (2) how to regulate the number of containers to effectively utilize the remaining resources for improving the QoS of unknown requests as much as possible.

In general, each microservice has multiple instances running independently in containers. When requests arrive at a microservice, the load balancer will send the request to a specific instance to handle, and the handling process obeys the first-come-first-served basis. Moreover, concluded from observations, the serving time needed by the requests have identically power laws distribution and are independent of interarrival time. Under this scenario, we can use queueing theory model to approximate the average staying time of requests in system as the QoS of requests [7]. Further, based on this model, we approximate the performance impact caused by the DDoS attack and calculate the optimal parameters to implement DDoS mitigation mechanism effectively.

For making our analysis, modeling, and the following experiments feasible and practical, we make some reasonable assumptions as follows:

- Due to the characteristics of low-rate DDoS attack, the benign and the malicious requests have similar

behavior patterns in traffic. Therefore, we suppose the network traffic is stable and constant whether under nonattack scenarios or attack scenarios. And in general, the requests couldn't all be malicious requests under attack scenarios.

- We suppose the arrival rate of requests to the microservice follows the Poisson distribution under nonattack scenarios. It is widely considered that the arrival of requests follows the Poisson distribution [7], [52], [53]. Further, based on the discussion above, we can still be reasonable to make the assumption that the arrival rate of requests obeys the Poisson distribution under attack cases in this paper.
- Last, we suppose the service rate of each individual container follows an exponential distribution, which is common in queueing analysis [7], and we suppose the containers with same configurations have the same service rate. Moreover, due to the capacity planning, we suppose that each microservice only has limited system resources to meet requirements of users or overcome the DDoS attack.

Based on the analyses and assumptions mentioned above, it is reasonable to use M/M/c queueing model to formalize the system, which means the queueing process has a Poisson arrival, infinite buffer size, $c(c \geq 2)$ multiple servers each with an exponential service rate. To date, the M/M/c queueing model is still the mainstream method to analyze this kind of system and can offer a closed form result [7], [51].

5.1 Approximation of the Service Performance in Nonattack Scenario

In this subsection, we formalize the microservice system with a mathematic model. In order to quantify the performance of the microservice, we use the *average staying time* of requests in system as a metric of QoS and approximate it with the queueing model.

In container-based cloud environment, the rate of requests arriving at a microservice follows the Poisson distribution, and we denote the arrival rate of requests as λ. The service rate of each container is indicated as μ, and the service rate of whole system, where has c containers, is μ_n shown in Equation (1).

$$\mu_n = \begin{cases} n\mu & n \leq c \\ c\mu & n > c \end{cases}. \tag{1}$$

Moreover, we use ρ to denote the *utility rate* of system as the ratio of arrival rate and the service rate which is shown in Equation (2).

$$\rho = \frac{\lambda}{c\mu}. \tag{2}$$

When the system in stable state ($\rho < 1$), we get the probability π_n of n requests in the system, which can be calculated by Equation (3).

$$\pi_n = \begin{cases} \frac{(c\rho)^n}{n!}\pi_0 & n \leq c \\ \frac{\rho^n c^c}{n!}\pi_0 & n > c \end{cases}. \tag{3}$$

In the above formula, π_0 represents the probability of situation that there has no requests in queue, and it can be formalized by Equation (4) in the M/M/c model.

$$\pi_0(\rho, c) = \left[\sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{c!(1-\rho)} \right]^{-1}. \tag{4}$$

Moreover, based on the *Little formula*, we can get average staying time T of the requests in system, which is denoted as Equation (5). In more detail, we set T measured in nonattack cases as the baseline of QoS which can be accepted by users.

$$T = \frac{1}{\lambda}\left(c\rho + \rho \frac{(c\rho)^c}{c!} \frac{\pi_0(\rho, c)}{(1-\rho)^2} \right). \tag{5}$$

5.2 QoS to Whitelist Users under DDoS Attack

Under DDoS attack, we divide the requests into two parts: *whitelist requests* and *unknown requests*. At the same time, a set of containers with the number of c_w are used to handle whitelist requests exclusively. Each container is assigned with same amount of resources. To the requests in the whitelist, our target is to maintain the QoS of them with the least system resources and the proper number of containers. In the same way as the nonattack cases, we can formalize the serving process of whitelist requests with the M/M/c queueing model. When the whitelist requests arrive at system, it will enter a queue separated from unknown requests and wait for being served. Moreover, the arrival rate of the requests in whitelist also follows the Poisson distribution and is denoted as λ_w. And the container's service rate μ_w is related to the available resources R_w and the number of containers c_w, which is formalized as follow:

$$\mu_w = \sigma(R_w, c_w). \tag{6}$$

Based on the queueing theory model M/M/c, the average staying time of whitelist requests in system can be calculated by Equation (7).

$$T_w = \frac{1}{\lambda_w}\left(c_w\rho_w + \rho_w \frac{(c_w\rho_w)^{c_w}}{c_w!} \frac{\pi_0(\rho_w, c_w)}{(1-\rho_w)^2} \right), \tag{7}$$

where ρ_w is the utility rate in serving process of whitelist requests, which is defined as

$$\rho_w = \frac{\lambda_w}{c_w\mu_w}. \tag{8}$$

Based on the analysis mentioned above, for making QoS of the whitelist requests acceptable, we need to determine the resource R_w assigned to each container and the number of containers c_w to meet $T_w \leq T$. Therefore, we have inequality as follow:

$$f(c_w, R_w) = T_w - T \leq 0. \tag{9}$$

Combining (6), (7), (8) and (9), we get

$$f(c_w,R_w) = \frac{1}{\sigma(R_w,c_w)} + \frac{\pi_0(\rho_w,c_w)\lambda_w^{c_w}}{c_w!\sigma(R_w,c_w)^{c_w}} \frac{\sigma(R_w,c_w)c_w}{[\sigma(R_w,c_w)c_w - \lambda_w]^2} \leq T, \tag{10}$$

where has constrains shown as follow:

$$\begin{cases} \frac{\lambda_w}{c_w \mu_w} \leq 1 \\ R_w \geq 0 \\ c_w = 2, 3, 4, ... \end{cases}. \tag{11}$$

Normally, the system has sufficient idle or reserved resources to serve the whitelist requests. So it is reasonable that we suppose Equation (10) is solvable. And the optimal solution in the solution set is the least resources and the optimal number of containers which can make the QoS of whitelist requests acceptable. Specifically, this solution can satisfy Equation (12).

$$\mathcal{R}_w = min[c_w \cdot R_w], c_w, R_w \in \{c_w, R_w \mid f(c_w, R_w) \leq 0\}. \tag{12}$$

Moreover, in this paper, we don't consider the resources needed by the virtualization of containers, because of the resources for this lightweight virtualization are much less than the virtual machine.

5.3 QoS to Unknown Users under DDoS Attack

After determining the amount of resources assigned to the whitelist, we use the rest of system resources R_u to serve the requests from unknown users. With the finite resources, our target is to maximize the QoS of unknown requests as much as possible under the DDoS attack by optimizing the number of containers.

Similarly, we can treat the serving process of unknown requests as a standalone M/M/c queueing model. Based on our assumptions, the arrival rate λ_u of unknown requests also follows the Poisson distribution and is denoted as follow:

$$\lambda_u = \lambda - \lambda_w. \tag{13}$$

The unknown requests are the mixture of benign requests and malicious requests. Due to the properties of the low-rate DDoS attack, we suppose that the service time for malicious request is β times of the benign request in average, and the malicious requests account for r of all unknown requests. In this case, the service rate of the system will be lower than the normal. Moreover, the service rate is also related to the available resources R_u of each container and the total number of containers c_u. Therefore, the service rate μ_u under the low-rate DDoS attack can be calculated by Equation (14).

$$\mu_u = \varepsilon \cdot \sigma(R_u, c_u), \tag{14}$$

where ε is a factor to depict the impact of malicious requests on service rate and is denoted as follow:

$$\varepsilon = \frac{1}{1 - r + r * \beta}. \tag{15}$$

In the Equation (15), r meet the condition that $0 < r \leq 1$. Moreover, utility rate ρ_u of the system can be calculated by Equation (16).

$$\rho_u = \frac{\lambda_u}{c_u \mu_u}. \tag{16}$$

To make QoS of the unknown requests acceptable, the average staying time T_u of the unknown requests in the system should meet Equation (17) where T' is the acceptable staying time for requests.

$$T_u = \frac{1}{\lambda_u}\left(c_u \rho_u + \rho_u \frac{(c_u \rho_u)^{c_u}}{c_u!} \frac{\pi_0(c_u, \rho_u)}{(1 - \rho_u)^2}\right) \leq T'. \tag{17}$$

For simplicity, let

$$f(c_u) = T_u - T' \leq 0. \tag{18}$$

Combining Equations (13) (14) (15) (16) and (17), the $f(c_u, \mu_u)$ can be denoted as Equation (19). If Equation (17) does not hold, we will constant reduce the T' until this equation can be solved. With the optimal number of containers c_u, the finite resources can be utilized effectively to maximize the QoS of unknown requests.

$$f(c_u, \mu_u) = \frac{1}{\mu_u} + \frac{c_u \mu_u \pi_0(c_u, \rho_u)}{[(c_u \cdot \mu_u)^2 - \lambda_u]\mu_u!} \cdot \frac{(c_u \lambda_u)^{c_u} c_u!}{\mu_u^{c_u}} \leq T', \tag{19}$$

where has constrain as follow:

$$\begin{cases} \frac{\lambda_u}{c_u \mu_u} \leq 1 \\ c_u = 2, 3, 4, ... \end{cases}. \tag{20}$$

6 Performance Evaluation

In this section, we evaluate the performance of our DDoS mitigation mechanism through the combination of simulation and testbed-based experiments. First, we introduce the values of key parameters used in our experiments based on a set of preliminary experiments and the results of existing researches. And then, we study the effectiveness of DDoS mitigation mechanism to the whitelist users and the unknown users under the low-rate DDoS attack.

6.1 Key Parameters of Our Experiments

First of all, we study a set of key parameters in our experimental scenarios based on the previous works and preliminary experiments. To the preliminary experiments, we use *docker-ce* (version 17.09) to create a docker swarm on testbed equipped with 2.40 GHz 64-bit Intel Xeon CPU E5-2630 v3 processor with 32-cores, 64 GB RAM, 4T disks, and two network interfaces both with 1Gbps network speed. And we use the *MATLAB* [54] to run simulation experiments.

In testbed-based experiments, we build a Python-based *products* microservice with a ReDoS vulnerability [55], which is a part of the e-commerce website shown in Fig. 1a and provides a products-inquiring function to others. The instances of this microservice are deployed in a set of containers, and the requests will be dispatched to each container through the API gateway provided by docker swarm. Further, we capture real users' inquiring requests by *tcpdump* [56] as the benign traffic. And we modify the benign request with

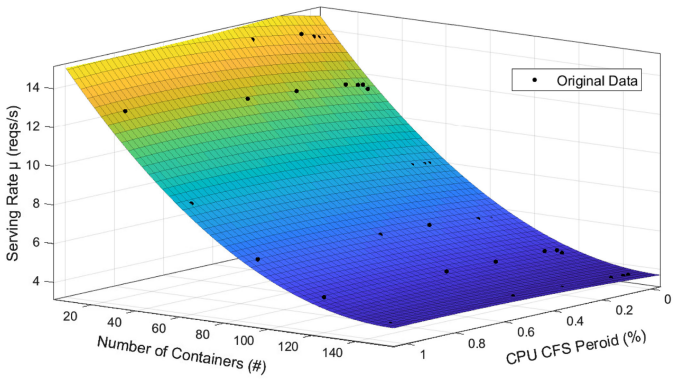


Fig. 2. Service rate of the microservice in different scenarios.

malicious payload, which can trigger the ReDoS vulnerability, to construct the attack traffic.

About the processing capacity of containers, as aforementioned, the container with different resource configurations will have different processing capacity μ, also called service rate. Moreover, due to the resources competition between containers, the number of containers is also a key parameter affecting the service rate of each container when system resources are limited. In our experiments, we only consider the CPU resource and use the *–cpu-period* option in docker to control container's CPU usage. With different proportions of CPU resource available, we test the number of requests can be handled per second by the microservice when it has a different scale of instances. In more detail, the relationship between service rate, resources configuration and the number of containers is shown in Fig. 2. In this figure, the black dots represent the service rate μ measured in the testbed-based experiments, and the fitting results of the relationship above are depicted with the 3D surface.

The fitting results show that the service rate of container decline with the increase in the number of containers or the decrease in the proportion of CPU resource. In contrast to the number of containers, the influence of available CPU resource to the service rate seemingly is slight. However, omitting either of them, we can find the influence of the other to the service rate is remarkable and can't be ignored. The relationships between service rate and the number of containers, and service rate and resource configurations are shown in Figs. 3 and 4 respectively.

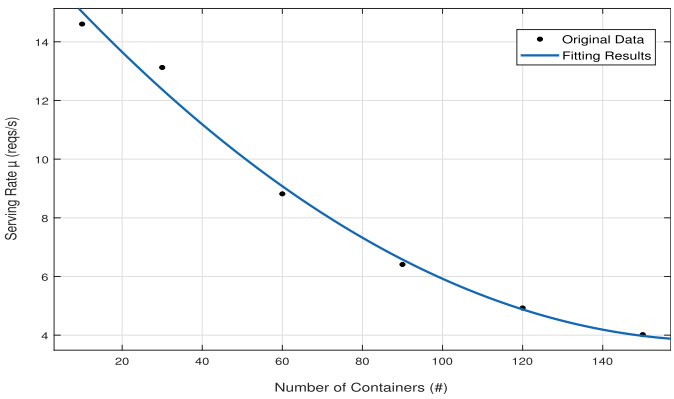


Fig. 3. Service rate of the microservice with a different number of containers.

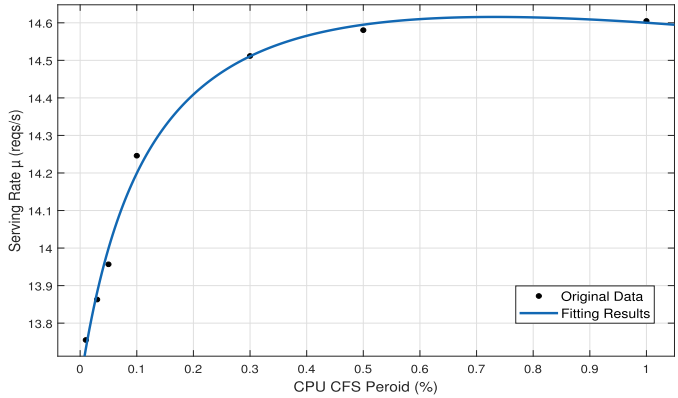


Fig. 4. Service rate of the microservice with different resource configurations.

The coefficient of determinations (*R-square*) of the fitting results mentioned above is all over the 0.995. Therefore, we believe that the fitting results are accurate and can be used in follow-up experiments.

Moreover, referring to the arrival rate of requests in a medium-sized e-commerce website [57], we set the arrival rate of requests λ in our experiments as 100 requests/s approximately. As discussed previously, the average staying time of requests in system is an important metric to reflect the quality of service. Based on the Equation (5) and the experimental results above, we can get that the average staying time T of requests is 0.067s under nonattack scenario. Further, we use the staying time to depict the capacity of our mitigation mechanism under different attack strengths.

To be specific, we generate a set of requests from 1,000 sources with an arrival rate of 100 requests per second, which follows the Poisson distribution. Moreover, according to the attack strength, we dynamically regulate the proportion of malicious requests in these requests by percentage. The number and arrival rate of the malicious requests are changed with attack strengths. And the attack duration lasts 5 minutes in our experiments.

Having verified the key parameters of our experiments, we now discuss the effectiveness of the mitigation mechanism on the whitelist and the unknown requests in Sections 6.2 and 6.3 respectively.

6.2 Performance of Whitelist Users

To the whitelist requests, we attempt to keep the average staying time of them acceptable and stable using a minimal amount of resources and the optimal number of containers. First, based on the prior study [58], we set that the trusted requests account for 30 percent of all requests and are tagged as the whitelist. Further, we study the conditions that could meet the performance requirements of whitelist requests according to Equation (10). And the results are shown in Fig. 5 which describe the relationship between the number of containers, amount of CPU resource and average staying time of requests in system.

From Fig. 5, when the number of containers is up to 6, we can observe that the average staying time of requests is the minimum. But exceeding that, the average staying time of requests increases as the number of containers increasing. At the same time, for reserving resource to unknown requests as much as possible, the least CPU resource required to whitelist

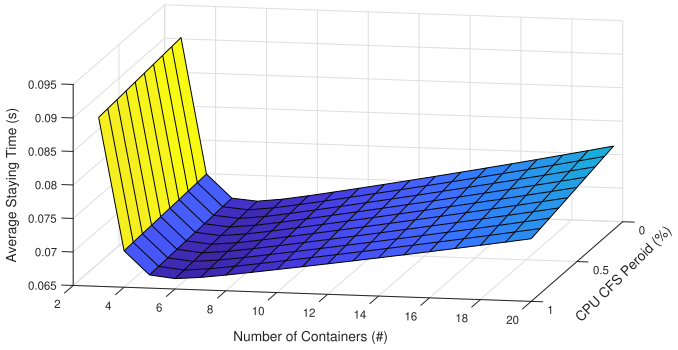


Fig. 5. Average staying time of whitelist in scenarios with a different number of containers and different configurations of CPU resource.

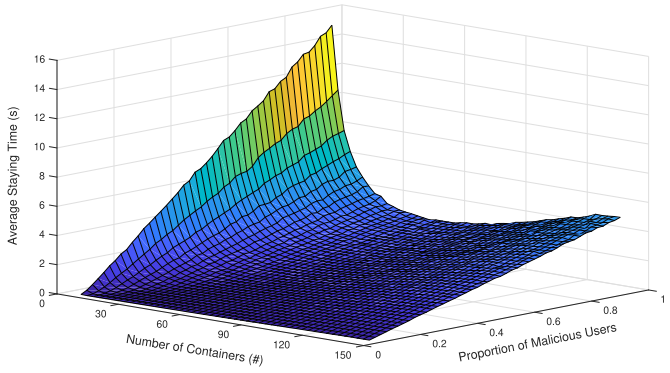


Fig. 7. Average waiting time of unknown requests in scenarios with a different number of containers and different attack strengths.

requests is also a key parameter. Thus, the optimal solution of these two parameters in Equation (10) will be used to ensure the QoS of whitelist requests acceptable. For demonstrating correctness of the solution, we apply the optimal parameters to run a set of simulation experiments and compare the results with the nonattack scenario, which is shown in Fig. 6.

Observing in Fig. 6, the staying time of requests in the system under nonattack scenario slightly exceed the staying time of whitelist requests under DDoS attack scenario. Due to the isolation of resources, the container instances serving for whitelist requests will not be impacted by the DDoS attack. Meanwhile, optimization to the number of containers can improve the QoS of whitelist requests with a certain degree. Based on the comparison results and the analyses above, we can verify the correctness of our theoretical model and the effectiveness of DDoS mitigation mechanism.

6.3 Performance of Unknown Users

To the unknown requests, the rest of resources will be assigned to them after meeting the requirement of whitelist requests. And we attempt to make the average staying time of unknown requests acceptable under the scenario that the proportion of malicious requests as high as possible. To achieve this target, we conduct an analysis of the relationship between the number of containers and average staying time of requests under different attack strengths according to the theoretical model described in Equation (19).

In this equation, the multiple β existing between the handling time of benign requests and malicious requests is an important parameter and fluctuate in a certain range. For simplifying the process of experiments, we suppose β as a deterministic value. According to the results of the research [59], we set the handling time of malicious requests is 25 times than the benign requests.

Now based on the theoretical model mentioned above, we depict the relationship between average staying time of requests in system, the number of containers and attack strengths in Fig. 7. From this figure, we can see that the staying time of requests grows linearly as the proportion of malicious requests increased. Moreover, when the number of containers is insufficient, the utility rate ρ of system is greater than 1. This means that the processing capacity of existing containers is not enough to tackle the requests arrived continually, which leads to the waiting time of subsequent requests sharply growing. Meanwhile, as we discussed earlier, the number of containers can't increase indefinitely. Thus, with the same proportion of malicious requests, if the number of containers continues to increase after exceeding a certain extent, the staying time of requests will continually increase rather than decrease. In this case, to maximize the QoS of unknown requests under different attack strengths, we need to dynamically regulate the number of containers for taking full advantage of resources.

Further, based on the results above, there exists the nonzero and unique solution to the Equation (19). Therefore, the DDoS mitigation mechanism will dynamically adjust the number of containers according to optimal results solved by the theoretical model to maximize the QoS of unknown requests. Moreover, under the attack scenarios, we set that the acceptable staying time for these requests is not more than 1.5 times to the normal requests, and this value is 1.005s in our experiments. To discuss the correctness and effectiveness of the mitigation mechanism, we conduct a set of simulation experiments to measure the average staying time of requests after DDoS mitigation mechanism optimizing under different attack strengths. And as the comparison group, we run the simulation experiments with the same configurations to obtain the staying time of requests in the worst case without optimization. In each experiment, we conduct 750,000 times simulations and then take the average as the final results. The details of the comparison results are shown in Fig. 8.

From Fig. 8, we observe that the average staying time of requests in worst case increases as the attack strength increased, and it is over the acceptable value when the proportion of malicious requests over 13 percent. Comparing with it, the growth trend of staying time of requests in the

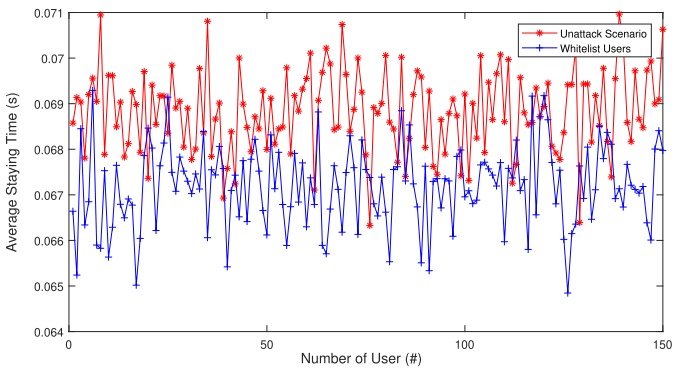


Fig. 6. Comparison of average staying time between the whitelist under attack scenario and the normal requests under nonattack scenario.

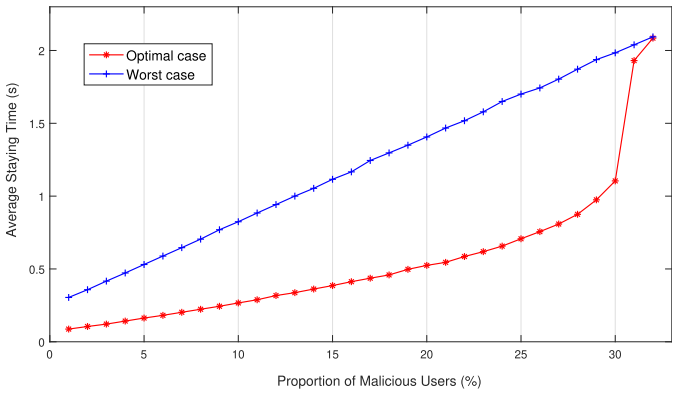


Fig. 8. Average waiting time of unknown users in scenarios with a different number of containers and different attack strengths.

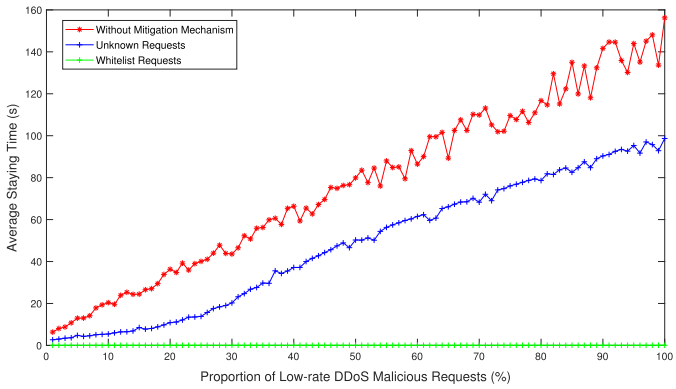


Fig. 9. Average waiting time of requests in complex DDoS scenarios with or without the DDoS mitigation mechanism.

optimized case is more gentle than the worst case, meanwhile, the configurations in the optimized case can help system survive the DDoS attack when its strength up to 30 percent. It means that blindly enlarging the number of containers to mitigate DDoS attack is not sensible in the container-based cloud environment. Moreover, based on these results, we believe our DDoS mitigation mechanism can keep the microservice available as long as possible when the low-rate DDoS attack occurs in container-based cloud environment.

6.4 Effectiveness on Complex DDoS Scenarios

At times, the low-rate DDoS attack is not launched independently and mixes with the flood-based DDoS attack. To evaluate the effectiveness of our mitigation mechanism in these complex scenarios, we simulate a flood-based DDoS attack and add it into the previous attack cases which only contain the low-rate DDoS attack. Specifically, the average attack rate of the flood-based DDoS attack is 5,000 requests per second, which is the 50 times workloads than the normal case [7]. And the evaluation results are shown in Fig. 9.

As observed in Fig. 9, with the mitigation mechanism, the QoS of whitelist requests can still be maintained in the normal level under the mixture DDoS attacks. Also, with the attack strength of the low-rate DDoS attack increasing, the QoS of unknown requests has a degree of improvement compared to the cases without our DDoS mitigation mechanism. However, the QoS of the unknown requests is still far away from the acceptable level, because the microservice only has limited resources to face the massive DDoS requests. In order to maximize the effectiveness of our mitigation mechanism, combining it with the traffic filtering mechanism will have greater capacity to defeat the DDoS attacks in these complex scenarios. In this case, the traffic filtering mechanism is a complement for our mitigation mechanism to filter the malicious requests from flood-based DDoS attack.

7 Further Discussion

To the best of our knowledge, this paper is the first work to discuss the DDoS mitigation mechanism in container-based cloud environment. Limited on the space, we have only discussed the influence of partial new features of container-based cloud environment for mitigating DDoS attack. As a new research field, a series of issues in there need to be investigated and optimized in the future. We list some research points following based on our understanding.

First, multiple scaling patterns in microservice architecture need to be considered in mitigating DDoS attacks. In this paper, we only consider the scenario using limited resources to scale a microservice to mitigate DDoS attacks. In practice, limiting scaling degree of the microservice is one way to keep its dependency chain stable. For example, the number of open database connections a microservice needs is limited. Once scaling this microservice infinitely, it inevitably leads to the database instances overload. Also, another way to avoid this issue is scaling all dependencies when a microservice being scaled. In this case, a microservice may need unlimited resources to be scaled. Therefore, how to effectively utilize the system resources to mitigate DDoS attacks with minimal economic cost in this scenario should be considered.

Second, our solution only focuses to mitigate the low-rate DDoS attack, without considering other kinds of DDoS attacks. As we know, the flood-based DDoS attack is also a common type of DDoS attacks in cloud environments [60]. Different from low-rate DDoS attack, the flood-based DDoS attack usually uses massive requests to inundate services. Under the monolithic application architecture, the attackers require investing significant resources to achieve their target. In contrast, in microservice architecture, we should discuss whether attackers can defeat key microservices with fewer resources achieving the same attack effect mentioned above. Or maybe, we should further discuss whether the loose-coupling feature of microservice architecture can help users dilute amounts of DDoS attack requests.

Third, for simplicity, we treat the container instance as a single thread service. Therefore, based on queueing theory, we use M/M/c model to formalize the serving process of container instances where each container instance is a server. In practice, due to different requirements, the container instance can also run multi-thread service. Accordingly, each container instance can be model as M/M/c queueing. Further, based on this model, the system resources can be assigned with a more fine-granularity way to defeat DDoS attacks. That is, we can calculate the accurate number of CPU, memory and I/O resources for each container instance based on this model.

Finally, our work focuses to increase tolerance of the microservice to DDoS attacks. But we are aware of the one-

sidedness of our work, because utilizing new features of the container-based cloud environment could produce more novel ideas to defeat DDoS attacks. For example, when the low-rate DDoS attack occurs, whether can we replace the microservice suffering from attack with another microservice with similar functions temporarily until the security issues solved. Researchers may take these new features into consideration to discover new DDoS mitigation methods.

8 Conclusion and Future Work

In this paper, we discuss whether the new features brought by container technology and microservice architecture can help the container-based cloud environment defeat low-rate DDoS attack more effectively. For this purpose, on the basis of queueing theory, we establish a mathematical model to formalize the low-rate DDoS attack scenario in container-based cloud environment. Based on this model, we explore the feasibility for defeating DDoS attack in container-based cloud environment with limited system resources. Further, we propose a strategy to mitigate low-rate DDoS attack according to the analysis results mentioned above. This strategy can dynamically reassign the system resources and optimize the number of containers to make the system resources be sufficiently utilized for defeating DDoS attack. Finally, we demonstrate the validity of this strategy through simulations and testbed-based experiments. And the experimental results demonstrate that our strategy can completely eliminate the influence of low-rate DDoS attack to the whitelist requests using minimal resources, and significantly improve the ability of other users to tolerate the DDoS attack.

As the new issues to research, there are a series of works need to be studied in the near future. As future works, we first want to explore the solutions to mitigate low-rate DDoS attack under the scenario that microservice can scale with unlimited resources. Further, under the unlimited resources scenario, we attempt to study the pricing issues in container-based cloud environment when defending DDoS attack. Finally, we will discuss defense schemes to more kinds of DDoS attacks in container-based cloud environment.

Acknowledgments

This paper was supported by the National Key Research & Development (R&D) Plan of China under grant No. 2017YFB0802205.

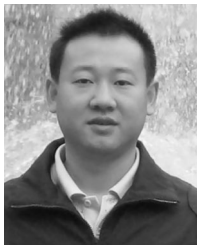
References

[1] "DDoS attacks in Q1 2018." [Online]. Available: https://securelist.com/ddos-report-in-q1-2018/85373/
[2] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, M. Rajarajan, and R. Buyya, "Combating DDoS attacks in the cloud: Requirements, trends, and future directions," *IEEE Cloud Comput.*, vol. 4, no. 1, pp. 22–32, Jan./Feb. 2017.
[3] S. Yu, S. Guo, and I. Stojmenovic, "Can we beat legitimate cyber behavior mimicking attacks from botnets?" in *Proc. IEEE Conf. Comput. Commun.*, 2012, pp. 2851–2855.
[4] Y. Chen, K. Hwang, and W.-S. Ku, "Collaborative detection of DDoS attacks over multiple network domains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1649–1662, 2007.
[5] J. François, I. Aib, and R. Boutaba, "Firecol: A collaborative protection network for the detection of flooding ddos attacks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1828–1841, Dec. 2012.
[6] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, et al., "Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures," *Service Oriented Comput. Appl.*, vol. 11, no. 2, pp. 233–247, 2017.
[7] S. Yu, Y. Tian, S. Guo, and D. O. Wu, "Can we beat DDoS attacks in clouds?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2245–2254, Sep. 2014.
[8] J. Idziorek, M. F. Tannian, and D. Jacobson, "The insecurity of cloud utility models," *IT Prof.*, vol. 15, no. 2, pp. 22–27, 2013.
[9] M. H. Sqalli, F. Al-Haidari, and K. Salah, "Edos-shield-a two-steps mitigation technique against edos attacks in cloud computing," in *Proc. 4th Int. Conf. Utility Cloud Comput.*, 2011, pp. 49–56.
[10] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *Proc. 11th Int. Andrei Ershov Memorial Conf. Perspectives Syst. Informat.*, 2017, pp. 95–104.
[11] "Benefits of Microservices." [Online]. Available: https://aws.amazon.com/cn/microservices/
[12] Fowler, Susan J, *Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization.*, Newton, MA, USA: O'Reilly Media, Inc., 2016.
[13] A. Bakshi and Y. B. Dujodwala, "Securing cloud from ddos attacks using intrusion detection system in virtual machine," in *Proc. 2nd Int. Conf. Commun. Softw. Netw.*, 2010, pp. 260–264.
[14] W. Dou, Q. Chen, and J. Chen, "A confidence-based filtering method for ddos attack defense in cloud environment," *Future Generation Comput. Syst.*, vol. 29, no. 7, pp. 1838–1850, 2013.
[15] T. Karnwal, S. Thandapanii, and G. Aghila, "A filter tree approach to protect cloud computing against XML DDoS and HTTP DDoS Attack," in *Proc. 2012 Int. Symp. Intell. Informat.*, 2012, pp. 459–469.
[16] M. N. Ismail, A. Aborujilah, S. Musa, and A. Shahzad, "Detecting flooding based DoS attack in cloud computing environment using covariance matrix approach," in *Proc. 7th Int. Conf. Ubiquitous Inf. Manage. Commun.*, 2013, pp. 36:1–36:6.
[17] Z. Chen, G. Xu, V. Mahalingam, L. Ge, J. H. Nguyen, W. Yu, and C. Lu, "A cloud computing based network monitoring and threat detection system for critical infrastructures," *Big Data Res.*, vol. 3, pp. 10–23, 2016.
[18] M. Ficco and M. Rak, "Stealthy denial of service strategy in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 3, no. 1, pp. 80–94, Jan.-Mar. 2015.
[19] O. Olivo, I. Dillig, and C. Lin, "Detecting and exploiting second order denial-of-service vulnerabilities in web applications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 616–628.
[20] G. Somani, A. Johri, M. Taneja, U. Pyne, M. S. Gaur, and D. Sanghi, "DARAC: DDoS mitigation using DDoS aware resource allocation in cloud," in *Proc. 11th Int. Conf. Inf. Syst. Secur.*, 2015, pp. 263–282.
[21] S. M. Alqahtani and R. F. Gamble, "DDoS attacks in service clouds," in *Proc. 48th Hawaii Int. Conf. Syst. Sci.*, 2015, pp. 5331–5340.
[22] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman, "CDN-on-demand: An affordable DDoS defense via untrusted clouds," in *Proc. 23rd Annu. Netw. Distrib. Syst. Secur. Symp.*, 2016.
[23] S. Zhao, K. Chen, and W. Zheng, "Defend against denial of service attack with VMM," in *Proc. 8th Int. Conf. Grid Cooperative Comput.*, 2009, pp. 91–96.
[24] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and M. Rajarajan, "DDoS victim service containment to minimize the internal collateral damages in cloud computing," *Comput. Electr. Eng.*, vol. 59, pp. 165–179, 2017.
[25] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," *Present and Ulterior Software Engineering.*, pp. 195–216, 2017.
[26] Y. Li, J. Zhang, C. Jiang, J. Wan, and Z. Ren, "PINE: Optimizing performance isolation in container environments," *IEEE Access*, vol. 7, pp. 30410–30422, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2900451. doi: 10.1109/ACCESS.2019.2900451.
[27] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI2: CPU performance isolation for shared compute clusters," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 379–391.
[28] P. Du and A. Nakao, "DDoS defense as a network service," in *Proc. 10th Netw. Operations Manage. Symp.*, 2010, pp. 894–897.
[29] M. H. Sqalli, F. Al-Haidari, and K. Salah, "EDoS-Shield - A two-steps mitigation technique against EDoS attacks in cloud computing," in *Proc. 4th Int. Conf. Utility Cloud Comput.*, 2011, pp. 49–56.

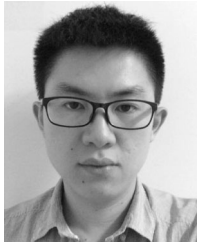
[30] "Amazon CloudWatch." 2018. [Online]. Available: https://aws.amazon.com/cloudwatch/
[31] J. Latanicki, P. Massonet, S. Naqvi, B. Rochwerger, and M. Villari, "Scalable cloud defenses for detection, analysis and mitigation of DDoS attacks," in *Proc. 2rd Int. Conf. Future Internet*, 2010, pp. 127–137.
[32] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "Service resizing for quick DDoS mitigation in cloud computing environment," *Annales des Télécommunications*, vol. 72, no. 5/6, pp. 237–252, 2017.
[33] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and M. Rajarajan, "DDoS victim service containment to minimize the internal collateral damages in cloud computing," *Comput. Electr. Eng.*, vol. 59, pp. 165–179, 2017.
[34] K. Ye, Y. Liu, G. Xu, and C. Xu, "Fault injection and detection for artificial intelligence applications in container-based clouds," in *Proc. 11th Int. Conf. Cloud Comput.*, 2018, pp. 112–127.
[35] Y. Xiang, K. Li, and W. Zhou, "Low-Rate DDoS attacks detection and traceback by using new information metrics," *IEEE Trans. Inf. Forensics Secur.*, vol. 6, no. 2, pp. 426–437, Jun. 2011.
[36] J. Luo, X. Yang, J. Wang, J. Xu, J. Sun, and K. Long, "On a mathematical model for low-rate shrew DDoS," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 7, pp. 1069–1083, Jul. 2014.
[37] Z. Wu, L. Zhang, and M. Yue, "Low-rate dos attacks detection based on network multifractal," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 5, pp. 559–567, Sep./Oct. 2016.
[38] J. Burnim, S. Juvekar, and K. Sen, "WISE: Automated test generation for worst-case complexity," in *Proc. 31st Int. Conf. Softw. Eng.*, 2009, pp. 463–473.
[39] B. Tak, C. Isci, S. S. Duri, N. Bila, S. Nadgowda, and J. Doran, "Understanding security implications of using containers in the cloud," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 313–319.
[40] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2017, pp. 237–248.
[41] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. M. Eyers, R. Kapitza, P. R. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 689–703.
[42] "AWS." 2018. [Online]. Available: https://aws.amazon.com/
[43] "IBM Cloud." 2018. [Online]. Available: https://www.ibm.com/cloud/container-service
[44] "Microsoft Azure." 2018. [Online]. Available: https://azure.microsoft.com/
[45] "Swarm mode overview." 2018. [Online]. Available: https://docs.docker.com/engine/swarm/
[46] "kubernetes." 2018. [Online]. Available: https://kubernetes.io/
[47] "Microservice." 2018. [Online]. Available: http://microservice.io
[48] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar, "Gremlin: Systematic resilience testing of microservices," in *Proc. 36th IEEE Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 57–66.
[49] J. Thones, "Microservices," *IEEE Softw.*, vol. 32, no. 1, pp. 116–116, May/Jun. 2015.
[50] A. Panda, M. Sagiv, and S. Shenker, "Verification in the age of microservices," in *Proc. 16th Workshop Hot Top. Operating Syst.*, 2017, pp. 30–36.
[51] D. P. Heyman, "Queueing systems," *Netw.*, vol. 7, no. 3, pp. 285–286, 1977.
[52] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2011, pp. 266–277.
[53] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *Proc. 13th USENIX Symp. Networked Syst. Des. Implementation*, 2016, pp. 501–521.
[54] "MATLAB." 2018. [Online]. Available: https://www.mathworks.com/
[55] "ReDoS." 2019. [Online]. Available: https://en.wikipedia.org/wiki/ReDoS
[56] "tcpdump." 2018. [Online]. Available: http://www.tcpdump.org/
[57] A. Bremler-Barr, E. Brosh, and M. Sides, "DDoS attack on cloud auto-scaling mechanisms," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[58] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 198–206.
[59] T. Petsios, J. Zhao, A. D. Keromytis, and S. Jana, "SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 2155–2168.
[60] J. Wang, X. Yang, M. Zhang, K. Long, and J. Xu, "HTTP-SoLDiER: An HTTP-flooding attack detection scheme with the large deviation principle," *Sci. China Inf. Sci.*, vol. 57, no. 10, pp. 1–15, 2014.

Zhi Li is currently working toward the PhD degree in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, China. His research interests include security in system architecture and cloud computing.

Hai Jin received the PhD degree in computer engineering from Huazhong University of Science and Technology, in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He is a Cheung Kung scholars chair professor of computer science and engineering with Huazhong University of Science and Technology. He worked with The University of Hong Kong between 1998 and 2000, and as a visiting scholar with the University of Southern California between 1999 and 2000. He is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. He has co-authored 22 books and published more than 700 research papers. He is a fellow of the IEEE, CCF fellow, and a member of the ACM.



Deqing Zou received the PhD degree from Huazhong University of Science and Technology, in 2004. He is a professor of computer science with Huazhong University of Science and Technology. His main research interests include software security, system security and cloud security. He has more than 50 high-quality papers, including papers published by NDSS, ACSAC, TPDS, TDSC and so on. He has always served as a reviewer for several prestigious Journals, such as the *IEEE Transactions on Parallel and Distributed Systems*, TOC, the *IEEE Transactions on Dependable and Secure Computing*, the *IEEE Transactions on Cloud Computing*, and so on. He is on the editorial boards of four international journals, and has served as PC chair/PC member of more than 40 international conferences.



Bin Yuan received the BS and PhD degrees in computer science and technology from HUST, in 2013 and 2018, respectively. He is currently a postdoc with Huazhong University of Science and Technology (HUST), Wuhan, China. His research interests include SDN security, NFV, cloud security, privacy and IoT security. He has published several technical papers in top journals, such as the *IEEE Transactions on Services Computing*, the *IEEE Transactions on Network and Service Management*, the *Future Generation Computing Systems* and the *IEEE Access*. He is a member of the IEEE.