SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

July 2023, Vol. 66 172103:1–172103:18
https://doi.org/10.1007/s11432-022-3593-7

Resource investment for DDoS attack resistant SDN: a practical assessment

Bin YUAN[1,2,3,4,5], Fan ZHANG[1,2,3,4], Jun WAN[1,2,3,4], Huan ZHAO[6,2,3,7], Shui YU[8], Deqing ZOU[1,2,3,4*], Qiangsheng HUA[6,2,3,7] & Hai JIN[6,2,3,7]

[1]*School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan* 430074*, China;*
[2]*National Engineering Research Center for Big Data Technology and System, Wuhan* 430074*, China;*
[3]*Services Computing Technology and System Lab, Wuhan* 430074*, China;*
[4]*Hubei Engineering Research Center on Big Data Security, Wuhan* 430074*, China;*
[5]*Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen* 518057*, China;*
[6]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074*, China;*
[7]*Cluster and Grid Computing Lab, Wuhan* 430074*, China;*
[8]*School of Computer Science, University of Technology Sydney, Ultimo* NSW 2007*, Australia*

Received 20 May 2022/Revised 4 July 2022/Accepted 23 September 2022/Published online 14 June 2023

Abstract Software-defined networks (SDNs) present a novel network architecture that is widely used in various datacenters. However, SDNs also suffer from many types of security threats, among which a distributed denial of service (DDoS) attack, which aims to drain the resources of SDN switches and controllers, is one of the most common. Once the switch or controller is damaged, the network services can be affected. Many defense schemes against DDoS attacks have been proposed from the perspective of attack detection; however, such defense schemes are known to suffer from a time consuming and unpromising accuracy, which could result in an unavailable network service before specific countermeasures are taken. To address this issue through a systematic investigation, we propose an elaborate resource-management mechanism against DDoS attacks in an SDN. Specifically, by considering the SDN topology, we leverage the M/M/c queuing model to measure the resistance of an SDN to DDoS attacks. Network administrators can therefore invest a reasonable number of resources into SDN switches and SDN controllers to defend against DDoS attacks while guaranteeing the quality of service (QoS). Comprehensive analyses and empirical data-based experiments demonstrate the effectiveness of the proposed approach.

Keywords SDN security, DDoS attacks, resource management, queueing theory, QoS

Citation Yuan B, Zhang F, Wan J, et al. Resource investment for DDoS attack resistant SDN: a practical assessment. Sci China Inf Sci, 2023, 66(7): 172103, https://doi.org/10.1007/s11432-022-3593-7

1 Introduction

Compared with a traditional network, a software-defined network (SDN) decouples the control plane from the data planes. The forwarding devices, such as switches on the data plane and the controller on the control plane, are important infrastructures, and the CPU is considered the most important resource because the CPU determines the processing capacity. Although the data and control planes have different responsibilities, they are closely related and communicate through the southbound protocols. OpenFlow is the most commonly applied orthodox standard SDN-based protocol [1, 2]. In addition, a flow table is filled with specific flow rules for each switch. The controller installs, modifies, and deletes the flow rules of the switch through the southbound protocol, whereas the switch forwards data packets according to the flow rules [1, 3].

By dividing the network into different planes, an SDN enables a programmatically efficient and dynamic network configuration to improve the network performance and monitoring, leading to a more elastic network management than that of a traditional network architecture. Although the centralized control and programmable features of an SDN provide many benefits to network management, such networks

* Corresponding author (email: deqingzou@hust.edu.cn)

still face several security threats [4]. SDNs have various security issues such as timing side channel attacks [5], persona hijacking [6], and network topology poisoning attacks [7]. Some countermeasures have been proposed to resist such attacks [6–9]. Meanwhile, the number of distributed denial of service (DDoS) attacks, one of the most common types of attacks having a significant impact on the quality of network services, is increasing each year and is becoming a major threat[1)]. In addition, with the development of network technology, DDoS attacks have different manifestations in SDN networks, such as attacking the flow table in the switch [10], controller, and link [11]. Irrespective of its type, a DDoS attack is essentially a competition between attackers and defenders for resources [12–14], and the party with the greater number of resources will be the final winner. The data and control planes are usually the targets of DDoS attacks. For instance, the processing of switches and controllers is inseparable during the installation of a new flow rule. If any of them fails to serve efficiently, the forwarding of network traffic will be affected, and the quality of service (QoS) can no longer be guaranteed.

Fortunately, for DDoS attacks in an SDN, prior studies have focused on massive intrusion detection and defense methods that exploit packet signatures or network behavior. Mousavi et al. [15] claimed that a state in which the controller cannot work normally owing to a DDoS attack is fatal to an SDN. The authors proposed an attack detection method that exploits the entropy variation of the destination IP address. Niyaz et al. [16] developed a DDoS detection system based on deep learning. Their implementation uses features extracted from network traffic headers, and then identifies the attack traffic based on the training model. Zheng et al. [17] presented an adaptive correlation analysis system for defense against a DDoS attack, by which the switches are instructed to collect partial information by the controllers, and the system is able to detect various types of DDoS attacks and throttle the attack traffic.

Most of these studies conducted to defend against DDoS attacks are based on attack detection, such as in [18], and are focused on blocking attack traffic after the detection of a DDoS attack. The main difference between their approaches is the diverse accuracy resulting from the various detection methods applied. Although these methods are useful for defending against a DDoS attack, a high cost of detecting such an attack is inevitable. As expected, the system may not be able to provide proper services during DDoS attack detection. Furthermore, if a sudden DDoS attack directly defeats the system, these detection systems cannot operate successfully or effectively.

In this paper, we propose an effective static resource investment method for defending against DDoS attacks aimed at affecting the QoS by delaying the installation time of the flow rules. As mentioned previously, the key issue in defeating DDoS attacks is the allocation of adequate resources. Provided a sufficient number of resources, the service will be unaffected. The installation of the flow rules involves switches and controllers. The CPU is one of the most important resources for the normal operation of the switches and controllers, and we therefore consider the number of CPUs when investing in switches and controllers. If a sufficient number of resources are invested in the switches and controllers, the time required to install the flow rules will not exceed a certain threshold. To deduce the resource requirements and guarantee the QoS when defending against DDoS attacks, we exploit queueing theory to conduct a performance evaluation, which is widely used in an SDN performance analysis, such as in [19, 20]. In addition, we consider the impact of the topology on the CPU resource investment in switches because switches in different locations undertake different traffic in a datacenter. Thus, it is rational to make different resource investments. As a result, if the network topology of a datacenter is determined, we can guarantee the QoS of the entire network through a reasonable resource investment based on our mathematical model. In addition, such a static resource allocation method can ensure continuous normal network services compared to a defense method based on attack detection.

Our contributions are as follows:

• We argue that the consumption of CPU resources must be considered to achieve a decent defense when facing DDoS attacks aimed at affecting the flow rule installation of the switches.

• We exploit queueing theory to build a mathematical model for estimating the resource allocation while guaranteeing the QoS of the network.

• We argue that the traffic flowing through the switches at different positions of the network topology differs, and thus we consider the CPU investment in switches according to their positions.

• We conducted an in-depth analysis on the model along with extensive experiments to evaluate our proposal, which was shown to be effective.

The remainder of this paper is organized as follows. Section 2 reviews prior efforts in this area. In

1) DDoS attacks in Q1 2020. https://securelist.com/ddos-attacks-in-q1-2020/96837/.

Section 3, we then describe the features of an SDN and the mathematical knowledge used in modeling the system. Section 4 describes the essential concept of investing resources to defend against a DDoS attack. Section 5 presents the details of the system model. Section 6 describes the results of the performance evaluations. Section 7 then discusses areas of future research, and Section 8 provides some concluding remarks.

2 Related work

Our approach was inspired by members' prior studies, which can be divided into the following three perspectives.

2.1 DDoS attacks in SDN

The purpose of a DDoS attack is to lead to an exhaustion of resources. Switches, controllers, and links are the main tools used in an SDN and therefore face serious threats from DDoS attacks.

Yuan et al. [10] pointed out that although switches are important forwarding nodes in an SDN, their flow-table space is usually limited. Attackers can launch a flow table overloading attack that causes the space to be exhausted. This is a transformed DDoS attack that can make the switch unable to consistently achieve data packet forwarding.

Cao et al. [11] proposed adversarial path reconnaissance, an original technique that identifies target paths and disrupts the control channel by generating low-rate TCP data traffic. They named the new attack a CrossPath attack, which utilizes the shared links in the paths of the control and data traffic to interrupt the SDN control channel. This leads to a failure of the normal transmission of messages in the data and control layers, which affects the upper-layer applications.

Wang et al. [21] proposed a novel attack in which attackers inject forging packets into the SDN network, compelling the switch to send `packet_in` messages and generate a resource exhaustion of the switch and controller. This is called a data-to-control plane saturation attack, which results in an overload of the SDN infrastructure.

Yuan et al. [22] indicated that cloud hosts can act as both attackers and victims in cybercrimes. An attacker can rent hosts from a cloud center and launch an attack targeting the host, making the target host unable to communicate properly.

Cao et al. [23] proposed a new attack called buffered packet hijacking, which is launched through malicious applications by exploiting a negligence in which SDN switches process the buffered packets of the responding buffer IDs following `flow_mod` messages without checking whether the match fields are matched with the buffered packets. An attacker can leverage such an attack to disrupt a TCP three-way handshake, which causes a saturation of the control channel.

2.2 DDoS attack defense in SDN

Compared with traditional networks, SDNs have more features that are helpful in resisting DDoS attacks. Using an SDN to ensure the network quality, researchers have proposed several methods for defending against DDoS attacks.

Zheng et al. [17] proposed a system for real-time attack defense based on a traffic data-information analysis. The system is divided into collectors, detectors, and locators. The major function of the collector is the collection of flow-related information. The detector obtains statistical information from the collector and then detects an attack by a correlation analysis. Finally, the locator locates the attack traffic and blocks it through the flow rules. Thus, several types of DDoS attacks can be detected.

Gillani et al. [24] proposed a resilient control network architecture (ReCON), which is a moving target technique, as a defense against a DDoS attack. The core of this scheme is to make full use of idle resources in the network. In the in-band mode of an SDN, ReCON can minimize the sharing of critical links between the control plane and the data plane and exploit the under-utilized OpenFlow agent (OFA) to elastically increase its capacity.

Zhang et al. [25] proposed SWGuard, a defense solution that detects and prioritizes abnormal control messages based on a host-application pair to defend against control plane reflection attacks. In the controller, a monitoring module collects the messages to be downlinked and assigns different priorities.

Multiple priority queues are set up on the switch. The scheduler on the switch conducts the scheduling and processing based on the different priorities of the messages.

Zhang et al. [26] proposed Poseidon, a defense method that exploits programmable switches to leverage abstracted policies of known DDoS defense strategies and dynamically manage the underlying resource management, providing a flexible and convenient method for implementing defense strategies.

Xie et al. [27] proposed CrossGuard, a defense system applied against CrossPath attacks, which mitigates congestion through protection rules that automatically activate a timeout mechanism to sustain the control flow process by reducing the bandwidth and detecting attack flows based on the flow statistics.

Many prior studies on DDoS attack defense methods require authority to control the entire system or extend the original switch or controller architecture. Undoubtedly, the method of expanding the original architecture to resist an attacks faces the problem of versatility in practical applications. In addition, it might be difficult for the system to provide the expected services when an attack occurs if the attack defense scheme is implemented after an attack has been detected.

2.3 Queueing model in SDN

In an SDN, the processing of the data packets must go through switches and controllers, and much of the work is done through queueing theory models for a related analysis.

Xiong et al. [19] proposed an OpenFlow network performance model based on queuing theory. They constructed a packet-forwarding process on the switches and a `packet_in` process on the controller through M/M/1 and M/G/1 queueing models, respectively, and solved the closed expression of the average packet sojourn time and the corresponding probability density function.

Singh et al. [28] mainly discussed the impact of the buffer on the switch according to the queueing theory. They compared the performance when the data and control plane traffic share a buffer, and the performance when these two types of traffic are processed by the switch in a priority queue. The results show that switching with the priority queuing buffer achieves a better performance.

Jarschel et al. [29] established an $M/M/1/\infty$ queuing theory model for the OpenFlow architecture based on the OpenFlow switch forwarding rate and blocking probability, which can help developers assess the sojourn time and packet loss rate of the data packets. This indicates that the processing time of the controller affects the sojourn time of the packets.

Shen et al. [30] proposed the use of queuing theory to estimate the minimum switch flow table entries, considering the flow distribution and path selection. They considered that the entire life cycle of a flow table entry comprises three processes: `packet_in`, handling, and serving. They established different queuing models for different processes and analyzed the impact on the flow table size.

Xiong et al. [19], Singh et al. [28], and Jarschel et al. [29] used a queuing model to construct a theoretical model to analyze the sojourn time or packet loss of data packets of the switches or controllers. Shen et al. [30] used queuing theory to analyze the flow table size. They did not consider that the switches and controllers in a real network environment are often multi-CPU; therefore, the results of the single-server queuing theory model are inaccurate. Unlike these studies, our queueing model estimates the time required to install the flow rules, considering the influence of the number of CPUs on the performances of the switches and controllers. Similar to Shen et al. [30], we considered the impact of the path selection on our model.

3 Preliminary knowledge

We briefly describe the essential features of an SDN and the queueing model.

3.1 SDN

An SDN has a novel architecture for depicting, constructing, and supervising the network and decouples the data and control planes with a flow-based forwarding strategy rather than a destination-based forwarding strategy [2, 31]. In addition, the network is programmable through protocols that interact with the underlying data plane devices. Figure 1 presents an overview of the SDN architecture. The data plane mainly consists of physical forwarding devices, such as switches, and the control plane where the controller runs is responsible for controlling the entire network based on a logically centralized network

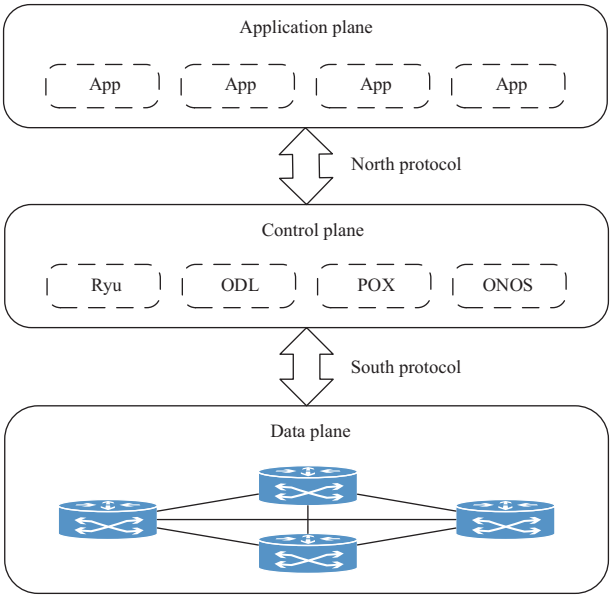

Figure 1 (Color online) SDN architecture.

view. Networking applications lie at the top of the SDN architecture, and are developed by network administrators for completing different types of tasks.

In the SDN paradigm, communication with applications exploits northbound APIs, which in practice prevalently choose the REST architectural style [32]. Network administrators can manage networks using northbound APIs. However, the control and data planes communicate with each other through southbound APIs. OpenFlow, one of the first standards in an SDN, is currently one of the most commonly used protocols.

The OpenFlow-enabled data plane forwards data packets according to the flow rules in the flow tables filled with flow entries. Match fields and action sets are the most important fields for a flow entry. The switch handles data packets based on the flow rule. If there is a match between the packets and entries, the packets are processed based on the action set. By contrast, a `packet_in` message is sent to the controller if no entries are matched. The controller receives the `packet_in` message, determines the packet process, and sends a `flow_mod` message in reply. Subsequently, a flow rule is installed on the switch. This process of treating new flows is reactive-based and is the current mainstream processing method [33].

3.2 Queueing model

The queuing theory model is a theoretical analysis model that is often used to estimate the average waiting time of a service and the length of the waiting queue. There are many different types of queuing theory models, including the birth and death processes, M/G/1 queue, and M/M/c queue. Most queueing theories deal with the system performance under a steady state. This means that both the arrival and service rates satisfy a certain distribution, and the ratio of the arrival rate to the service rate should be less than 1.

In queuing theory, the M/M/c queue is a queuing model with multiple servers [34], which indicates that there are c servers in the system. The service time obeys a negative exponential distribution with parameter μ, and the arrival rate λ of the customers follows a Poisson distribution. As shown in Figure 2, an M/M/c queue is a stochastic process whose state space is the number of all customers in the system, including customers currently being serviced. The process state changes from i to $i+1$ with a customer arrival, whereas it changes from i to $i+1$ with a customer departure. If the number of customers is less than c, there will be idle servers. Otherwise, extra customers temporarily remain in the buffer queue.

The possibility of stat i is π_i. Clearly, the sum of all values of π_i is 1.

$$\sum \pi_i = 1, \quad i \geqslant 0. \tag{1}$$

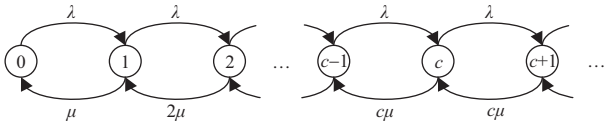

Figure 2 State space diagram of M/M/c model.

In addition, the system has a stationary distribution if $\lambda < c\mu$, which deduces the server utilization ρ as

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

The probability of each state is

$$\pi_k = \begin{cases} \pi_0 \dfrac{(c\rho)^k}{k!}, & 0 < k < c, \\ \pi_0 \dfrac{(c\rho)^k c^{c-k}}{c!}, & c \leqslant k. \end{cases} \tag{3}$$

In (3), π_0 denotes the probability of a situation occurring in which no customers are waiting to be served. Based on the M/M/c model, we can formalize π_0 as follows:

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

In addition, based on Little's formula, the average staying time T of the packets is

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

4 DDoS attack defense based on resource management

First, we consider the features of the switch and controller under a non-attack scenario, which means that all packets of benign users lie within the queue and are served by the switches and controllers. As long as the data flow does not overload the switch and controller, they can operate normally. In general, the QoS is adequate for users with a stable number of benign users.

When DDoS attacks arrive at the switches or controllers, a massive number of forged packets generated by botnets are injected into the SDN network. The processing capacity of the switches and controllers affects the installation of the flow rules. This indicates that to carry out a task and guarantee the QoS of the network, sufficient resources must be invested. If the network topology is provided, we can roughly estimate the upper limit of the traffic carried by each switch node. Once we know the traffic to be processed by the switch and controller and the threshold of the entire processing time, we can determine the number of CPUs that need to be invested.

We can estimate the appropriate number of CPUs invested in the switches and controllers through the load of the attack packets. It can be deduced that the number of resources required to defeat an attack is affordable because the attack load that a botnet can launch is limited. It is therefore reasonable to expect a resource-management mechanism to defend against a DDoS attack.

5 System modeling and analysis

To model the system, we first depict the influence of the topology on the traffic distribution. Later, a practical mathematical model was proposed based on queueing theory [35] to estimate the resource requirements for defeating a DDoS attack. According to this model, we solve the problem of maintaining an acceptable QoS, which is the time required to install flow rules with the least number of resources.

To ensure the practicality and feasibility of our modeling, analysis, and experiments, we make the following reasonable assumptions.

Table 1 Notations

Notation	Definition	Notation	Definition
a	Arrival rate of new packets on edge switch	p^{packetin}	proportion of `packet_in` in switch j
$\beta^{(s,d)}$	Number of paths from switch s to d	p^{flowmod}	Proportion of `flow_mod` in switch j
c	Number of messages in queue	π_k	Probability of k messages stored in controller
f_j	Arrival rate of `flow_mod` messages on switch j	π_{jk}	Probability of k messages stored in switch j
$\gamma_i^{(s,d)}$	Probability of choosing path i from s to d	$\sigma_j^{(s,d)}$	Probability of switch j receiving `flow_mod` of traffic s to d
λ_j	Total arrival rate of packets on switch j	t^{con}	Average waiting time on the controller
μ^{con}	Service rate of controller	t_{total}	Maximal total time for flow rule installation
μ^{flowmod}	Service rate for `flow_mod` messages	t_j	Average waiting time on switch j
μ^{packetin}	Service rate for `packet_in` messages	T_0	Upper limit of flow rule installation time
μ_j	Service rate of switch j	τ	Total arrival rate on the controller
n	Number of edge switches in the topology	$y_j^{(s,d)}$	Whether switch j is on the path from switch s to d
ρ	Utility rate	$\zeta^{(s,d)}$	Probability of source edge switch sending traffic to others

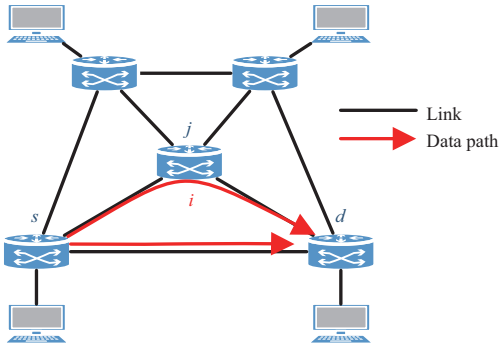

Figure 3 (Color online) Simple network topology of data center.

• We assume that the network traffic is constant and stable regardless of whether it is under a nonattack or attack scenario. In fact, not all packets are malicious under the attack scenarios.

• We assume that the packet arrival of the switches and controllers obeys a Poisson distribution. It is commonly believed that the packet arrival conforms to a Poisson distribution [36]. In addition, it is also rational to suppose that under DDoS attack scenarios, the packet arrival conforms to a Poisson distribution [12].

• We assume that the service rate of the switches and controllers follows an exponential distribution. In addition, we assume that the buffers of the switches and controllers are infinite.

Based on the above analyses and assumptions, it is sensible to formalize the system by exploiting an M/M/c queuing model consisting of a Poisson arrival queuing process, an infinite buffer, and c servers whose service rates follow an exponential distribution. To the best of our knowledge, the M/M/c queuing model is prevalent in the current SDN analyses.

For clarity, all notations mentioned in this paper are listed in Table 1.

5.1 Traffic distribution

The locations of the switches are determined in the given topology of the data center. As shown in Figure 3, there may be multiple reachable paths from one edge switching to another, and there are also different routing strategies [37] for the controllers.

In the network, data traffic flows from the source to the target host. According to the different routing strategies, the switches through which the traffic flows pass are also different. We use s to represent the edge switches connected to the source host and d to represent the edge switches connected to the destination host, as shown in Figure 3. From s to d, $\beta^{(s,d)}$ reachable paths exist. In accord with the flow rules on the switches, the data traffic from s to d flows through one of the $\beta^{(s,d)}$ paths each time. Let $\gamma_i^{(s,d)}$ denote the probability of choosing path i. If we simply consider that the probability of each

alternative path selected is the same, we have

$$\begin{cases} \gamma_i^{(s,d)} = \dfrac{1}{\beta^{(s,d)}}, \\ \displaystyle\sum_{i=1}^{\beta^{(s,d)}} \gamma_i^{(s,d)} = 1. \end{cases} \tag{6}$$

In general, a host can send traffic to multiple other hosts in a network. Therefore, we consider the source edge switch to be s, where d represents any other reachable destination-edge switch. If the number of edge switches in the network is counted as n, then edge switch s can send traffic to other $n-1$ edge switches. This implies that the source edge switch has the same probability of sending traffic to the other edge switches, denoted by $\zeta^{(s,d)}$. Thus, we have the following equation:

$$\begin{cases} \zeta^{(s,d)} = \dfrac{1}{n-1}, \\ \displaystyle\sum_{d=1}^{n} [d \neq s] \zeta^{(s,d)} = 1. \end{cases} \tag{7}$$

The traffic forwarding must pass through the switches, and each switch may be a forwarding node on multiple forwarding paths. Let $y_j^{(s,d)}$ denote whether switch j is on the path from s to d, and we obtain

$$y_j^{(s,d)} = \begin{cases} 1, & \text{if } j \text{ is on path}(s,d), \\ 0, & \text{if } j \text{ is not on path}(s,d). \end{cases} \tag{8}$$

We consider only the traffic that forces the switch to generate `packet_in` messages. The edge switch generates the `packet_in` message and sends it to the controller after the traffic sent by the host reaches the edge switch. The controller then replies to the corresponding `flow_mod` message of the switch after processing. Combining (6), (7), and (8), we can obtain the probability that a switch j receives the `flow_mod` message when s sends data packets to d as follows:

$$\sigma_j^{(s,d)} = \left(\sum_{i=1}^{\beta^{(s,d)}} y_j^{(s,d)} \times \gamma_i^{(s,d)} \right) \times \zeta^{(s,d)}. \tag{9}$$

5.2 System modeling

As mentioned earlier, the flow rule installation in an SDN network requires the cooperation of switches and controllers. When a new flow arrives at a switch, it generates a `packet_in` message and sends it to the controller. The controller processes the `flow_mod` message according to the routing policy. Finally, the flow rule is installed by the switch after receiving the message. Figure 4 shows the structure of the entire process, which mainly includes the queuing processes on the switch and controller. During the message processing, the QoS is closely related to the performance of the CPU, and we mainly discuss the number of CPUs on the impact of the process messages. In this subsection, we model and analyze both the switches and controllers.

5.3 Queueing process on switches and controller

When DDoS attacks occur, the traffic enters the network from the edge switch. According to the above assumptions, the arrival rate a_e of the new flow follows a Poisson distribution. Based on Burke's theorem [38], we can know that the sum of the arrival rates of new packets on each edge switch is the arrival rate of `packet_in` messages received by the controller. Thus, the arrival rate of controller τ is

$$\tau = \sum_{e=1}^{n} a_e. \tag{10}$$

The service rate of the CPUs on the controller is defined as μ^{con}, whereas ρ^{con}, the ratio of the arrival rate to the service rate, which is also known as the utility rate of the controller, is shown as

$$\rho^{\text{con}} = \frac{\tau}{\mu^{\text{con}}}. \tag{11}$$

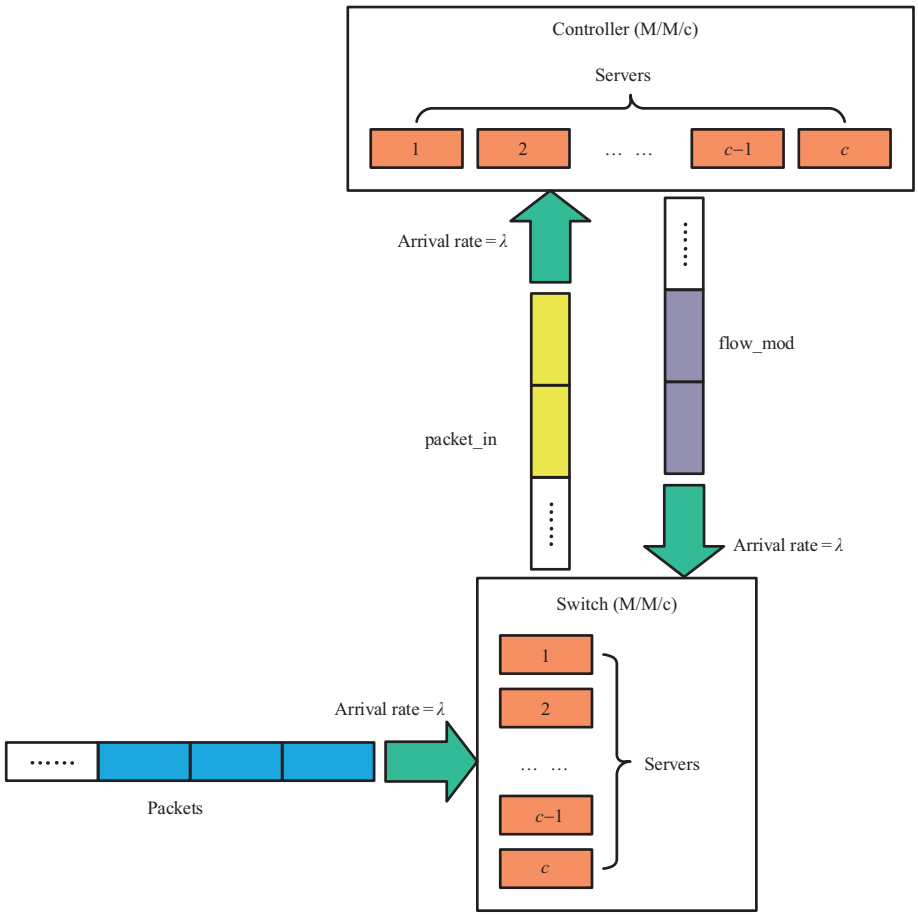

Figure 4 (Color online) Model of switches and controller.

If $\rho^{\text{con}} < 1$, which means that the system is in a stable state, we can calculate the probability that none of the packets will stay in the queue through (4). This leads us to calculate the average waiting time t^{con} for the data packet to be processed by the controller according to (5), which leads to

$$t^{\text{con}} = \frac{1}{\tau}\left(c\rho^{\text{con}} + \rho^{\text{con}}\frac{(c\rho^{\text{con}})^c}{c!}\frac{\pi_0}{(1-\rho^{\text{con}})^2}\right). \tag{12}$$

After processing the `packet_in` message, the controller delivered `flow_mod` messages to the corresponding switches. Because normal data packets with matching flow rules can be quickly processed through a high-speed network adapter, the impact on the CPU is minimal and can be ignored. Therefore, during the process of installing the flow rules, the traffic on the edge switch mainly includes the traffic generating `packet_in` messages and the `flow_mod` traffic from the controller. Meanwhile, the internal switch processes the `flow_mod` traffic. In the previous discussion, we mentioned that the switches through which the traffic flows vary with each routing strategy. Thus, the arrival rate of `flow_mod` messages on switch j is calculated using

$$f_j = \begin{cases} \displaystyle\sum_{s=1}^{n} \sigma_j^{(s,j)} a_s, & \text{if } j \text{ is an edge switch,} \\ \displaystyle\sum_{s=1}^{n}\sum_{d=1}^{n} [s \neq d]\sigma_j^{(s,d)} a_s, & \text{others.} \end{cases} \tag{13}$$

Based on the analysis mentioned above, the total arrival rate of the packets on switch j is

$$\lambda_j = \begin{cases} a_j + f_j, & \text{if switch } j \text{ is an edge switch,} \\ f_j, & \text{others.} \end{cases} \tag{14}$$

The switch has different service rates for processing `flow_mod` messages and generates `packet_in` messages. To facilitate the model construction, we simply regard the overall service rate on the switch as the weighted average of the two, which is defined as

$$\mu_j = \mu^{\text{packetin}} \times p^{\text{packetin}} + \mu^{\text{flowmod}} \times p^{\text{flowmod}}, \tag{15}$$

the constraints of which are as follows:

$$\begin{cases} p^{\text{packetin}} = \dfrac{a_j}{\lambda_j}, \\ p^{\text{flowmod}} = \dfrac{f_j}{\lambda_j}. \end{cases} \tag{16}$$

When $\rho_j = \frac{\lambda_j}{\mu_j} < 1$, the queueing system of the switch is under a steady state. Consequently, the waiting time t_j on switch j can be expressed as

$$t_j = \frac{1}{\lambda_j}\left(c\rho_j + \rho_j \frac{(c\rho_j)^c}{c!}\frac{\pi_{j0}}{(1-\rho_j)^2}\right). \tag{17}$$

The packet handling process consists of two parts: switch and controller processes. To successfully defend against a DDoS attack, the main task is to guarantee the QoS of both processes with the least investment of resources, which in our case is the smallest number of CPUs. Based on the above analysis, we need to determine the number of CPUs on the switch and controller and keep the total time for the flow rule installation less than the threshold so as to achieve an acceptable QoS. The installation of the flow rules mainly consists of three processes: generating a `packet_in` message on the switches, replying to the `flow_mod` message from the controller to the switches after processing, and installing the flow rule on the switches. Therefore, by combining (12) and (17), we obtain

$$t_{\text{total}} = 2 \times \max t_j + t^{\text{con}}. \tag{18}$$

Notably, for t_{total}, to prove that the flow rule installation time within the entire network is less than the threshold, we only need to consider the maximum t_j that belongs to the switch with the highest amount of traffic in the network.

To guarantee the QoS, that is, the total time must be less than a given threshold, the following constraints must be satisfied:

$$\begin{cases} t_{\text{total}} < T_0, \\ \dfrac{\lambda_j}{\mu_j} < 1, \\ \dfrac{\tau}{\mu^{\text{con}}} < 1. \end{cases} \tag{19}$$

The independent variable in the processing time on switches and controllers is the number of CPUs, which causes a change in the flow-rule installation time. Therefore, as long as we invest sufficient resources, we can ensure that the QoS will not be affected by a DDoS attack.

6 Performance evaluation

In this section, a series of experiments conducted to evaluate the performance of the proposed DDoS defense mechanism are described. In Subsection 6.1, we list the key parameters and experimental setup used in later experiments. We then verify the correctness of our analytical model in Subsection 6.2. Finally, in Subsections 6.3 and 6.4, we provide a feasible and practical evaluation through a discussion of the impacts of the arrival rate, service rate, and number of CPUs on the switches and controllers.

6.1 Key parameters and experiment setup

Key parameters. The key parameters in Table 2 were derived from the following experimental results and solid prior efforts published in top journals and conference proceedings.

Table 2 Value of key parameters

Feature	Value
Arrival rate (p/s)	100 to 290
Service rate per CPU on switch (p/s)	3530.57
Service rate per CPU on controller (p/s)	3227.25
Attack rate (p/s)	500, 5000, 50000
T_0 (ms)	0.957
c_{sw}, c_{con}	1 to 100

(1) Arrival rate. The waiting time is significantly affected by the arrival rate in both our model and practice. Therefore, to compare the theoretical and experimental results or show their impact, we either set the arrival rate from the empirical data or vary it for a comprehensive analysis.

(2) Service rate. To estimate the theoretical waiting time in a switch, we calculate the service rate of each CPU on the switch as 3530.57 p/s depending on a reasonable assumption and later experiment results, whereas we set the service rate of each CPU on the controller as 3227.25 p/s.

(3) Attack rate. Moore et al. [39] pointed out that the most common attack rate of a DDoS attack is generally 500 packets per second or higher. Thus, to conduct simulations under different attack strengths, we set the attack rate to 500, 5000, and 50000.

(4) Threshold of flow rule installation time (T_0). We set the upper limit of the flow rule installation time to 0.957 ms, which represents the threshold to ensure the QoS when maintaining a normal function for benign users when referring to empirical data.

(5) The numbers of CPUs on a switch and controller are (c_{sw} and c_{con}). To evaluate the impact of a resource investment on the QoS, we vary the number of CPUs on the switch and controller from 1 to 100.

Experiment setup. For the preliminary experiments, we used Mininet (version 2.1.0P1)[2)] for the SDN network emulation, and Ryu (version 4.15)[3)] as the SDN controller. We created the testbed and virtualized the experimental environment using Mininet on a virtual machine running Ubuntu Linux 18.04 with a single CPU core and 2 GB of RAM.

6.2 Correctness of analytical model

In this subsection, we present an evaluation of the correctness of our proposed strategy based on an SDN testbed. Specifically, we first conducted real simulations on different queuing processes, obtained the real average waiting time according to a statistical analysis, and substituted the measured values into a mathematical model to obtain the theoretical average waiting time.

6.2.1 *Service rate and actual waiting time of generating packet_in messages in switches*

When a new flow arrives at the switch, it generates a `packet_in` message. To measure the service rate for generating a `packet_in` messages on the switch, we first install the default flow rule on the switch, which directs the switch to generate a `packet_in` message if no rules match the packet. Meanwhile, we inject data packets into the network with a fixed packet sending frequency, and time stamp the data packets at the same time. In addition, TShark[4)] is used to capture the corresponding `packet_in` message and record the corresponding times. Finally, we obtained the experimental results shown in Figure 5 through simple mathematical calculations, which indicate the service rate by which the switch generates `packet_in` messages.

Figure 5 indicates that the service rate at which the switch generates `packet_in` messages changes dynamically as the frequency of packet sending increases each time. The actual measured average waiting time also fluctuates within a small range. Using the data in Figure 5, the average service rate and actual average waiting time were calculated as 3996.98 p/s and 0.2507 ms, respectively.

6.2.2 *Service rate and actual waiting time of processing flow_mod messages in switches*

For the switch, the flow rules are installed after receiving the `flow_mod` message from the controller. To measure the service rate of the switch processing `flow_mod` messages, we injected data packets into the

2) Mininet. https://mininet.org/.
3) Ryu. https://ryu-sdn.org/.
4) tshark. https://www.wireshark.org/docs/man-pages/tshark.html.

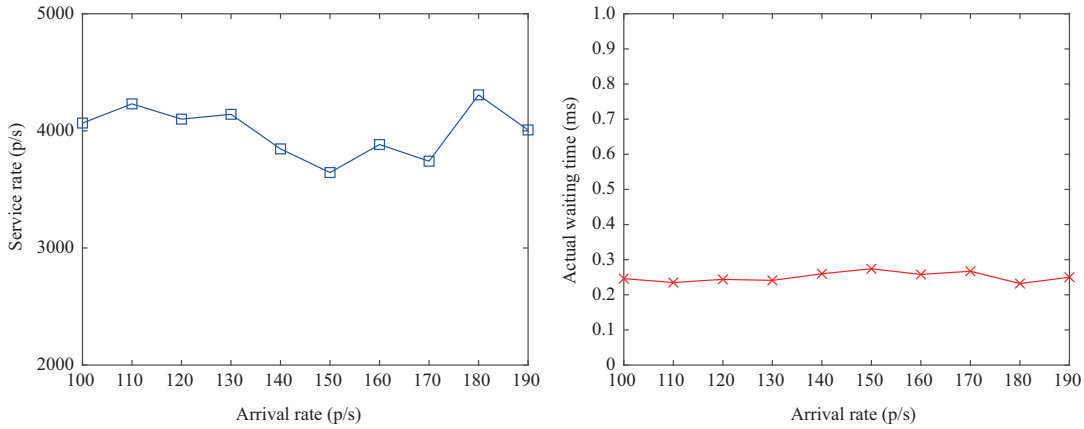

Figure 5 (Color online) Service rate and actual waiting time when switches generate `packet_in` messages.

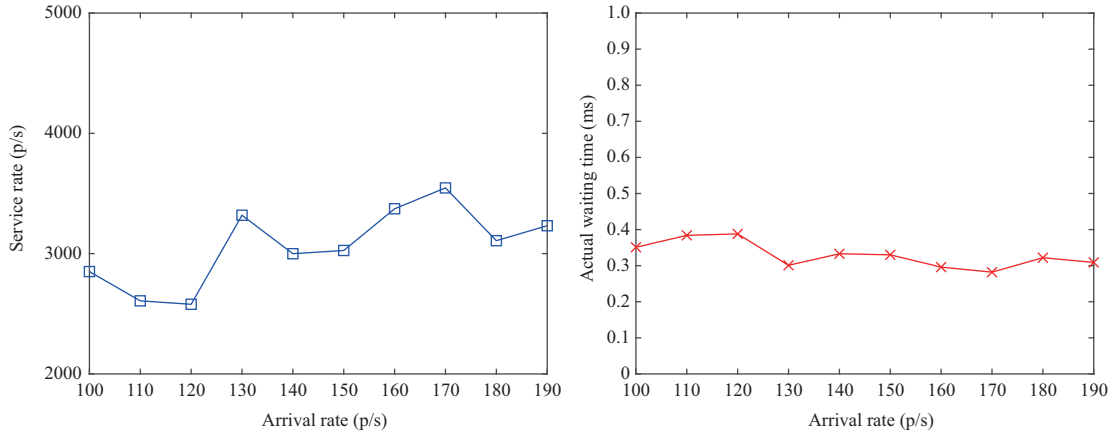

Figure 6 (Color online) Service rate and actual waiting time when switches process `flow_mod` messages.

network such that the controller received a `packet_in` message from the switch and sent a `flow_mod` message in return. We then use TShark to capture the `flow_mod` messages and record the entry time. In addition, we capture the forwarded packet following the newly installed rule and its exit time. Thus, we can obtain the results shown in Figure 6 by calculating the relevant data.

Although the service rate of processing `flow_mod` messages is also a fluctuating value, it is significantly lower than that of the switch generating `packet_in` messages. Thus, it is clear that the actual average waiting time for processing `flow_mod` messages is higher than that for generating `packet_in` messages. According to the data from Figure 6, the average service rate of the switch processing `flow_mod` messages and the average waiting times is 3064.16 p/s and 0.3296 ms, respectively.

6.2.3 *Service rate and actual waiting time of controller*

One of the main responsibilities of the controller is to process the received and uplink messages from the switch, and to then generate the corresponding message as a reply to the switch, such as the interaction of `packet_in` and `flow_mod`. The controller conducts various functions as a control layer. We applied a path calculation function on the controller to obtain the service rate of the controller. First, we inject data packets into the network at different packet-sending frequencies through the virtual host in Mininet. Because the process of the controller actually starts at the entry point of the function rather than the time when the packets are received, to capture the packets, we place timestamps on both the entrance and exit of the function that processes the `packet_in` message in the controller instead of using TShark. We use TShark to capture packets in the previous experiment scenario mainly to avoid modifying the source code of the OVS[5)] switch during the Mininet simulation experiment. Finally, we obtained the results shown in Figure 7.

5) OVS. http://www.openvswitch.org/.

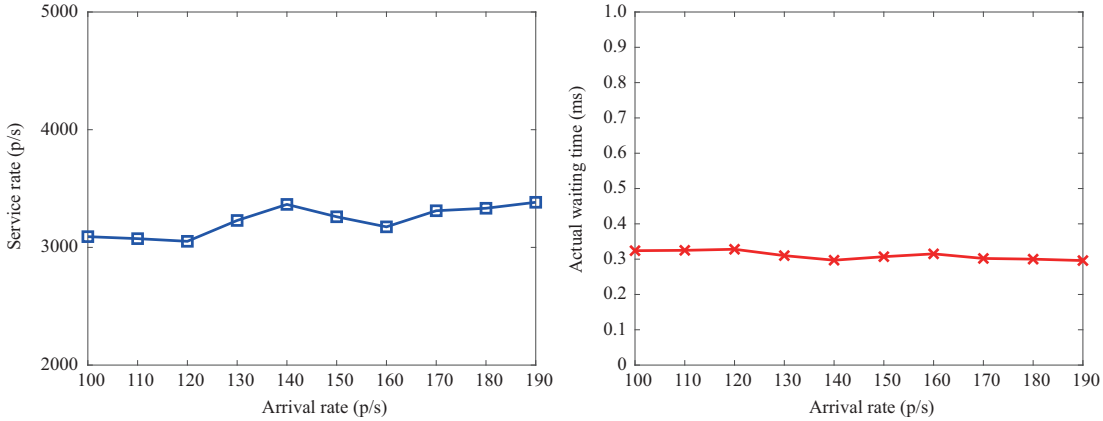

Figure 7 (Color online) Service rate and actual waiting time of the controller.

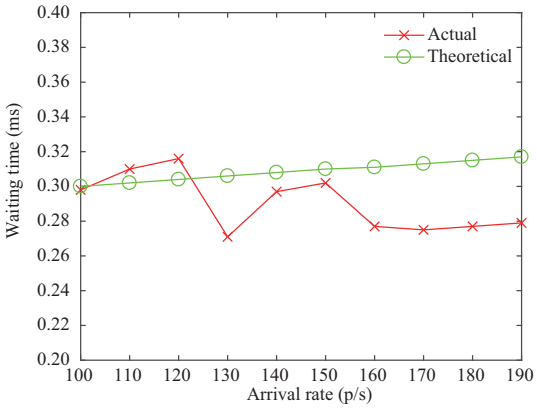

Figure 8 (Color online) Actual and theoretical waiting time in the switch.

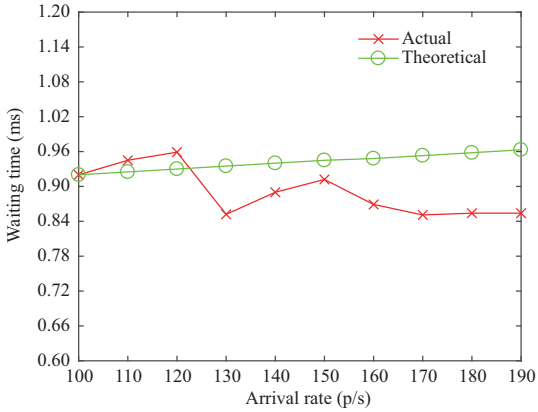

Figure 9 (Color online) Actual waiting time and theoretical waiting time for rule installation.

Using the data in Figure 7, we can calculate the average service rate and average actual waiting time of the controller as 3227.25 p/s and 0.3104 ms, respectively. It should be noted that the service rate on the controller is closely related to the specific services running on the controller.

6.2.4 *Comparison of theoretical and actual waiting times*

We previously pointed out that the service rate on the switch is the weighted average service rate of generating `packet_in` messages and processing `flow_mod` messages. During our experiment, we injected traffic from one host; therefore, the ratio of `packet_in` messages to `flow_mod` messages on the switch should be 1 : 1. Based on these analyses, we can calculate that the weighted average service rate of the switch is 3530.57 p/s using (15) and substitute it into (17) to obtain the theoretical average waiting time. In addition, we determined the actual average waiting time in the switch by summing the time required for the switch to generate `packet_in` messages and process `flow_mod` messages under each experimental scenario. Finally, we obtained the result shown in Figure 8.

From Figure 8, we can see that the difference between the theoretical waiting time and the actual waiting time does not exceed 0.04 ms when different types of messages are processed on the switch. On this basis, we can add the processing time in the controller to obtain the waiting time required for the entire flow-rule installation process, as shown in Figure 9.

The total theoretical waiting time for installing flow rules increases as the arrival rate (the load of CPUs) increases, indicating an increasing trend in terms of the actual time, as shown in Figure 9. The actual measurement value fluctuates because the data packet statistics obtained through the packet capture itself have a certain time overhead. However, from the actual results, the error was approximately 0.1 ms, which is considered acceptable.

To show that such results are acceptable, we conducted corresponding statistical analyses on these

Table 3 Average, maximal, and minimal waiting times under different situations

Type	Average (ms)	Max (ms)	Min (ms)
Actual `packet_in`	0.251	0.274	0.232
Theoretical `packet_in`	0.260	0.263	0.257
Actual `flow_mod`	0.330	0.388	0.282
Theoretical `flow_mod`	0.343	0.348	0.337
Actual switch	0.290	0.316	0.275
Theoretical switch	0.309	0.317	0.300
Actual controller	0.310	0.328	0.296
Theoretical controller	0.339	0.344	0.333

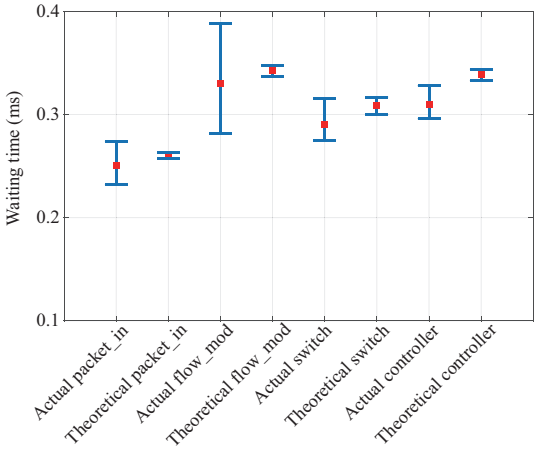

Figure 10 (Color online) Average-max-min of actual and theoretical waiting times under different scenarios.

data at various arrival rates and obtained the average waiting time under different scenarios alongside the minimum and maximum waiting times, which are listed in Table 3. To observe the difference in the range of values more intuitively, we plotted an average-max-min chart of the data in Figure 10, ranging from minimum to maximum and including the average point. Comparing the results, the error ranges between the actual and theoretical waiting times under the same scenarios are extremely small, which proves the correctness of our analysis model.

6.3 Influence of different parameters

We can see from Section 5 that the time required to install the flow rules is affected by several system parameters. Therefore, in this section, we study the influence of different parameters to evaluate the proposed method.

6.3.1 *Influence of arrival rate*

To study the influence of the arrival rate, we evaluated the impact on the waiting time. We modeled the process on the switch and controller. Because they are both M/M/c models, their analysis results were similar. We therefore only analyzed the queuing process on the switch as an example. By increasing the arrival rate of data packets with a fixed service rate of the switch, we can observe the change in the average waiting time, which is shown in Figure 11.

Figure 11 shows that the average waiting time in the switch increases with an increase in the arrival rate, which means that if we have high-quality QoS requirements, they will be difficult to meet when the arrival rate reaches a certain value. DDoS attacks can be regarded as an increase in the arrival rate. Based on our analysis model, we know that to resist the impact of a DDoS attack, we need to invest more resources to improve the processing capabilities of the switches and controllers.

6.3.2 *Influence of service rate*

As described in Subsection 6.3.1, we fixed the other variables and observed the change in average waiting time as the service rate changed.

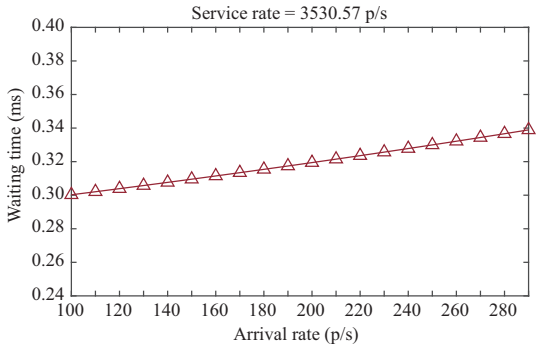

Figure 11 (Color online) Influence of arrival rate.

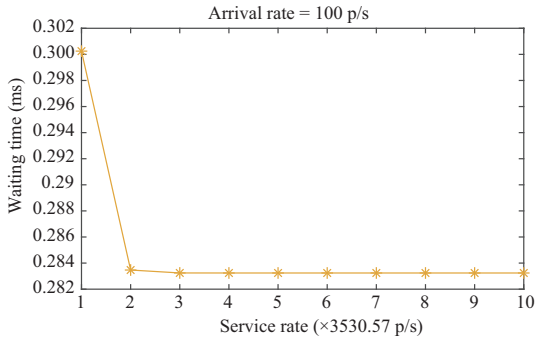

Figure 12 (Color online) Influence of service rate.

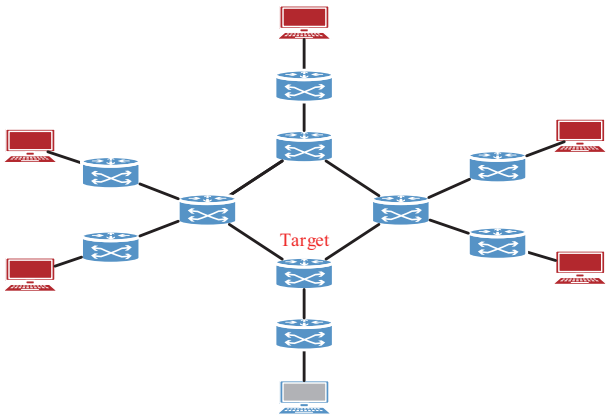

Figure 13 (Color online) Network topology under the attack scenario.

As shown in Figure 12, when the arrival rate is fixed, if we increase the number of CPUs, that is, enhance the service rate of the server, we can see that the waiting time initially decreases significantly. However, as the service rate increases, the decrease in waiting time slows and stabilizes. This is because, when the arrival rate is fixed, the number of packets to be processed within a certain period of time can be determined. Increasing the service rate can effectively alleviate the processing pressure. However, the number of tasks to be processed remains the same while the number of CPUs increases, which leads to a surplus in the processing capacity of the server, and thus the waiting time will not change significantly later on. The line chart in Figure 12 shows that the slope changes significantly at the beginning, and then gradually becomes more gentle.

6.4 Resource investment for QoS guarantee

In this subsection, we discuss whether switches and controllers can resist DDoS attacks under different resource investments; that is, we determine whether the total waiting time is less than a given threshold. Here, the resources we refer to are mainly the number of CPUs on the switch and the controller.

We consider the topology shown in Figure 13 as an example. We assume that the attacker compromises five hosts. The attack rate at which each compromised host sends attack packets to an innocent host in the network is fixed. It is assumed that the probabilities of each alternative path selected are equal. According to the traffic distribution analysis, we found that the switch marked as the target in Figure 13 has the most traffic. Given that the theoretical upper limit of the flow rule installation time T_0 is double the theoretical average waiting time in the switch 0.309 ms plus the theoretical average waiting time in controller 0.339 ms, which is 0.957 ms, we analyze how the number of CPUs on the switch and controller affects the QoS when the attack strength varies. Specifically, we evaluate whether the resource investment of the switches and controllers can meet the conditions in (19) with attack rates of 500 p/s, 5000 p/s, and 50000 p/s. The x-axis represents c_{sw}, which is the number of CPUs on the switch, whereas the y-axis represents c_{con}, which is the number of CPUs on the controller. A value of -1 on the z-axis indicates that the QoS cannot be satisfied, and a value of 1 indicates that the QoS can be satisfied.

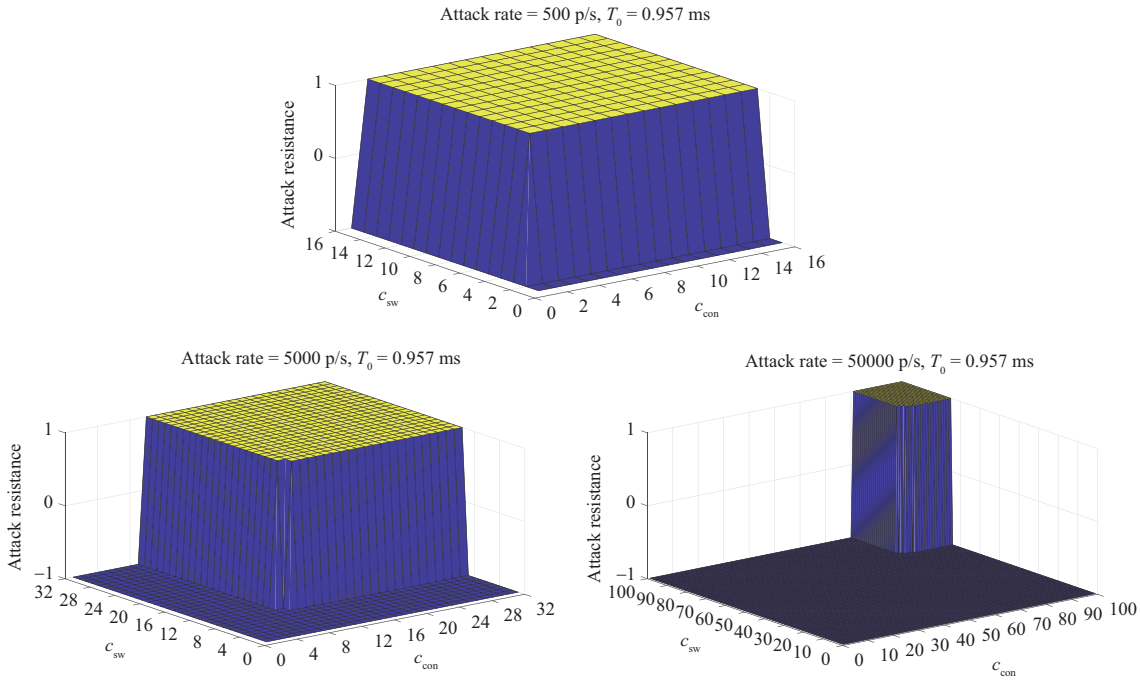

Figure 14 (Color online) Attack resistance under different attack rates.

From Figure 14, it is clear that the numbers of CPUs on the switch and controller determine whether the QoS can be satisfied under different attack strengths and the given topology. When the attack strength is low, a small number of resources for the switch and controller are sufficient to maintain the QoS requirement. However, the higher the attack strength, the more resource switches and controllers are needed to achieve a resistance to DDoS attacks.

7 Further discussion and future work

To the best of our knowledge, this study is the first to combine the M/M/c queue with the influence of switches and controllers on the installation process of flow rules. Owing to space and time limitations, the discussion is limited to the scheme proposed in this paper. However, there are still many issues that require improvement and future research. We discuss some of these in this section.

First, we did not consider how to dynamically adjust and allocate resources, which are less adaptable because the intensity of the attack is continuously changing. If resources can be allocated dynamically, resource wastage can be effectively avoided. In a future study, we will investigate how to implement a dynamic configuration of the network resources when considering the load of the controllers and switches by exploiting network function virtualization.

Second, for the resources targeted by a DDoS attack, we only considered the CPU. However, many essential resources, such as the memory and network, can easily become potential targets. In a future study, we will work on an extension of the proposed model, apply the method to the management of other resources, and optimize it into a more elaborate resource investment and all-around attack defense scheme.

Finally, when we consider the impact of the CPU on the service rate, we did not consider that the configuration of the CPU itself may differ, which means that different types of CPUs may have different service rates. Therefore, we will consider this in more detail in the future.

In future research, we will adapt our model to overcome these limitations. In addition, we intend to improve the resource investment mechanism by reducing the calculation complexity.

8 Conclusion

In this study, we indicated that a flow rule installation involves the processing of switches and controllers in a software-defined network, and that the installation time of the flow rules affects the normal forwarding of network data packets. To address this problem, we propose a resource allocation method that considers the impact of the network topology on the resource investment based on queueing theory. By adopting the M/M/c queuing model, the CPU resources invested in switches and controllers for ensuring that the installation time of the entire process flow rule does not exceed a given threshold can be calculated. Consequently, an effective defense against DDoS attacks can be achieved based on the premise of minimal resource investment. In addition, the network topology can fully help the network administrator construct the entire network reasonably while predicting the network load. The results of the performance evaluation show that our solution can guarantee the quality of network services through reasonable resource investment.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 61902138), Key-Area Research and Development Program of Guangdong Province (Grant No. 2019B010139001), and Shenzhen Fundamental Research Program (Grant No. JCYJ20170413114215614).

References

1 McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput Commun Rev, 2008, 38: 69–74
2 Kreutz D, Ramos F M V, Verissimo P E, et al. Software-defined networking: a comprehensive survey. Proc IEEE, 2015, 103: 14–76
3 Li D, Wang S, Zhu K, et al. A survey of network update in SDN. Front Comput Sci, 2017, 11: 4–12
4 Zhang H G, Han W B, Lai X J, et al. Survey on cyberspace security. Sci China Inf Sci, 2015, 58: 110101
5 Cui H, Karame G O, Klaedtke F, et al. On the fingerprinting of software-defined networks. IEEE Trans Inform Forensic Secur, 2016, 11: 2160–2173
6 Jero S, Koch W, Skowyra R, et al. Identifier binding attacks and defenses in software-defined networks. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, 2017. 415–432
7 Hong S, Xu L, Wang H, et al. Poisoning network visibility in software-defined networks: new attacks and countermeasures. In: Proceedings of the 22nd Annual Network and Distributed System Security Symposium, San Diego, 2015
8 Sonchack J, Dubey A, Aviv A J, et al. Timing-based reconnaissance and defense in software-defined networks. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, 2016. 89–100
9 Liu S, Reiter M K, Sekar V. Flow reconnaissance via timing attacks on SDN switches. In: Proceedings of the 37th IEEE International Conference on Distributed Computing Systems, Atlanta, 2017. 196–206
10 Yuan B, Zou D, Yu S, et al. Defending against flow table overloading attack in software-defined networks. IEEE Trans Serv Comput, 2019, 12: 231–246
11 Cao J, Li Q, Xie R, et al. A crosspath attack disrupts the SDN control channel via shared links. In: Proceedings of the 28th USENIX Security Symposium, Santa Clara, 2019. 19–36
12 Yu S, Guo S, Stojmenovic I. Can we beat legitimate cyber behavior that mimics botnet attacks? In: Proceedings of the IEEE Conference on Computer Communications, Orlando, 2012. 2851–2855
13 Chen Y, Hwang K, Ku W S. Collaborative detection of DDoS attacks over multiple network domains. IEEE Trans Parallel Distrib Syst, 2007, 18: 1649–1662
14 Francois J, Aib I, Boutaba R. FireCol: a collaborative protection network for the detection of flooding DDoS attacks. IEEE ACM Trans Networking, 2012, 20: 1828–1841
15 Mousavi S M, St-Hilaire M. Early detection of ddos attacks against SDN controllers. In: Proceedings of the International Conference on Computing, Networking and Communications, Garden Grove, 2015. 77–81
16 Niyaz Q, Sun W, Javaid A Y. A deep learning based DDoS detection system in software-defined networking (SDN). ICST Trans Security Saf, 2017, 4: 153515
17 Zheng J, Li Q, Gu G, et al. Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis. IEEE Trans Inform Forensic Secur, 2018, 13: 1838–1853
18 Guo Y, Miao F, Zhang L C, et al. CATH: an effective method for detecting denial-of-service attacks in software defined networks. Sci China Inf Sci, 2019, 62: 032106
19 Xiong B, Yang K, Zhao J, et al. Performance evaluation of OpenFlow-based software-defined networks based on queueing model. Comput Networks, 2016, 102: 172–185
20 Muhizi S, Shamshin G, Muthanna A, et al. Analysis and performance evaluation of SDN queue model. In: Proceedings of the Wired/Wireless Internet Communications, St. Petersburg, 2017. 26–37
21 Wang H, Xu L, Gu, G. FloodGuard: a DoS attack prevention extension in software-defined networks. In: Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, 2015. 239–250
22 Yuan B, Zou D, Jin H, et al. HostWatcher: protecting hosts in cloud data centers through software-defined networking. Future Generation Comput Syst, 2020, 105: 964–972
23 Cao J, Xie R, Sun K, et al. When match fields do not need to match, buffered packets are hijacked in the SDN. In: Proceedings of the 27th Annual Network and Distributed System Security Symposium, San Diego, 2020
24 Gillani F, Al-Shaer E, Duan Q. In-design resilient SDN control plane and elastic forwarding against aggressive DDoS attacks. In: Proceedings of the 5th ACM Workshop on Moving Target Defense, Toronto, 2018. 80–89
25 Zhang M, Li G, Xu L, et al. Control plane reflection attacks in SDNs: new attacks and countermeasures. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, Heraklion, 2018. 161–183
26 Zhang M, Li G, Wang S, et al. Poseidon: mitigating volumetric DDoS attacks with programmable switches. In: Proceedings of the 27th Annual Network and Distributed System Security Symposium, San Diego, 2020

27 Xie R, Cao J, Li Q, et al. Disrupting the SDN control channel via shared links: attacks and countermeasures. IEEE ACM Trans Networking, 2022, 30: 2158–2172
28 Singh D, Ng B, Lai Y, et al. Modelling software-defined networking: switch design with finite buffer and priority queuing. In: Proceedings of the 42nd IEEE Conference on Local Computer Network, Singapore, 2017. 567–570
29 Jarschel M, Oechsner S, Schlosser D, et al. Modeling and performance evaluation of the open flow architecture. In: Proceedings of the 23rd International Teletraffic Congress, San Francisco, 2011. 1–7
30 Shen G, Li Q, Ai S, et al. How powerful switches should be deployed: a precise estimation based on queuing theory. In: Proceedings of the 2019 IEEE Conference on Computer Communications, Paris, 2019. 811–819
31 Gong Y, Huang W, Wang W, et al. A survey on software defined networking and its applications. Front Comput Sci, 2015, 9: 827–845
32 Zhou W, Li L, Luo M, et al. REST API design patterns for the SDN northbound API. In: Proceedings of the 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, 2014. 358–365
33 Huang G, Youn H Y. Proactive eviction of flow entry for SDN based on hidden Markov model. Front Comput Sci, 2020, 14: 144502
34 Harrison P G, Patel N M. Performance Modelling of Communication Networks and Computer Architectures. Boston: Addison-Wesley Longman Publishing Co., Inc., 1993
35 Zukerman M. Introduction to queuing theory and stochastic teletraffic models. 2013. ArXiv:1307.2968
36 Raiciu C, Barré S, Pluntke C, et al. Improving datacenter performance and robustness by using multipath TCP. In: Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications1, Toronto, 2011. 266–277
37 Echenique P, Gómez-Gardeñes J, Moreno Y. Improved routing strategies for Internet traffic delivery. Phys Rev E, 2004, 70: 056105
38 Walrand J. A probabilistic look at networks of quasi-reversible queues. IEEE Trans Inform Theor, 1983, 29: 825–831
39 Moore D, Shannon C, Brown D J, et al. Inferring Internet denial-of-service activity. ACM Trans Comput Syst, 2006, 24: 115–139