

## 基于 API 调用管理的 SDN 应用层 DDoS 攻击防御机制

王洋<sup>1</sup>, 汤光明<sup>1</sup>, 王硕<sup>2</sup>, 楚江<sup>2</sup>

(1.信息工程大学, 河南 郑州 450001; 2.中国西安卫星测控中心, 陕西 西安 710043)

**摘要:** 软件定义网络(SDN, software defined network)针对北向接口安全研究少, 加之缺乏严格的访问控制、身份认证及异常调用检测等机制, 导致攻击者有机会开发恶意的应用程序, 造成北向应用程序接口(API, application programming interface)的滥用, 不利于 SDN 的全面推广。针对应用层的分布式拒绝服务(DDoS, distributed denial-of-service)主要有两种样态: 一是攻击者设计恶意 App, 绕过北向接口的安全审查, 对某些 API 进行短时间大量调用, 进而导致控制器崩溃, 使整个网络瘫痪; 二是攻击者以某个合法 SDN 应用程序作为攻击目标, 对该应用程序所需用的特定 API 进行短时间大量调用, 使该合法 App 无法正常调用 API, 进而使该合法 App 无法正常工作。与第一种攻击相比, 第二种攻击更为隐蔽。因而, 如何分辨 App 是恶意的还是合法的、如何对受攻击控制器上运行的 App 进行快速清洗以分离出恶意 App、如何对合法 App 重新分配控制器以保证其正常运行, 成为必须。在深入分析当前北向接口发展趋势的基础上, 模拟并实践了对可能的 DDoS 攻击样态, 并据此提出了基于 API 调用管理的 SDN 应用层 DDoS 防御机制。该机制在 SDN 应用层和控制层之间增加了一层 App 管理层。通过对 App 的信誉管理、初始审查、映射分配、异常检测和识别迁移, 来预判和抵抗恶意 App 对 SDN 的攻击。机制侧重于在攻击发生前对恶意 App 进行事先审查, 以避免攻击的发生。若攻击已然发生, 则对合法 App 和恶意 App 进行清洗分离。理论与实验验证表明, 所提安全机制有效避免了 SDN 应用层的 DDoS 攻击, 且算法运行效率高。

**关键词:** 拒绝服务攻击; 网络安全; 软件定义网络; 北向接口

**中图分类号:** TP393

**文献标志码:** A

**DOI:** 10.11959/j.issn.2096-109x.2022017

## Defense mechanism of SDN application layer against DDoS attack based on API call management

WANG Yang<sup>1</sup>, TANG Guangming<sup>1</sup>, WANG Shuo<sup>2</sup>, CHU Jiang<sup>2</sup>

1. Information Engineering University, Zhengzhou 450001, China

2. China Xi'an Satellite Control Center, Xi'an 710043, China

**Abstract:** Due to the lack of strict access control, identity authentication and abnormal call detection, attackers may

收稿日期: 2021-03-29; 修回日期: 2021-10-20

通信作者: 王硕, Waltshuo@163.com

基金项目: 国家自然科学基金(61802438)

**Foundation Item:** The National Natural Science Foundation of China (61802438)

**引用格式:** 王洋, 汤光明, 王硕, 等. 基于 API 调用管理的 SDN 应用层 DDoS 攻击防御机制[J]. 网络与信息安全学报, 2022, 8(2): 73-87.

**Citation Format:** WANG Y, TANG G M, WANG S, et al. Defense mechanism of SDN application layer against DDoS attack based on API call management[J]. Chinese Journal of Network and Information Security, 2022, 8(2): 73-87.

develop malicious applications easily and then it leads to the abuse of the northbound interface API (application programming interface) accordingly. There are mainly two patterns of DDoS (distributed denial-of-service) attacks against application layer. 1) malicious App bypass the security review of the northbound interface and make a large number of calls to some API in a short time, thus causing the controller to crash and paralyzing the whole network; 2) attackers take a legitimate SDN (software defined network) application as the target and make a large number of short-time calls to the specific API needed by the application, which makes the legitimate App unable to call the API normally. Compared with the first pattern, the second one is more subtle. Therefore, it's necessary to distinguish whether the App is malicious or not, effectively clean the App running on the attacked controller, and redistribute the controller to the legitimate App. Based on the in-depth analysis of the development trend of the current northbound interface, the possible DDoS attack patterns were simulated and practiced. Then a DDoS defense mechanism for SDN application layer was proposed. This mechanism added an App management layer between SDN application layer and control layer. Through reputation management, initial review, mapping allocation, anomaly detection and identification migration of the App, the malicious App attack on SDN can be predicted and resisted. The proposal focused on pre-examination of malicious App before attacks occur, so as to avoid attacks. If the attack has already happened, the operation of cleaning and separating the legitimate App from the malicious App is triggered. Theoretical and experimental results show that the proposed mechanism can effectively avoid DDoS attacks in SDN application layer, and the algorithm runs efficiently.

**Keywords:** DDoS, network security, SDN, northbound interface

## 0 引言

软件定义网络 (SDN, software defined network) 及其相关技术已经成为未来网络发展的重要方向之一<sup>[1-3]</sup>。然而,随着软件定义网络的不断发展和实践应用,SDN 面临诸多安全问题,极大地影响了其发展进程<sup>[4]</sup>。事实上,由于 SDN 架构的核心在于将网络控制任务交给控制器处理,控制器自然而然成为整个网络架构的大脑,但这同时导致控制器成为网络攻击的焦点。特别地,攻击者往往会发动针对控制器的分布式拒绝服务 (DDoS, distributed denial-of-service) 攻击,从而使控制器资源耗尽,最终导致整个网络瘫痪。因此,研究抵抗面向 SDN 的 DDoS 攻击及检测技术成为当前 SDN 安全问题的研究热点<sup>[5]</sup>。

分析现有研究可知,针对 SDN 的 DDoS 攻击及检测技术集中于数据层和控制层,其发动攻击的原理在于利用 OpenFlow 协议中的漏洞,通过持续发送特定数据包使交换机流表溢出或控制器资源耗尽。具体地,在数据层,攻击者通过增加僵尸节点个数等手段使交换机流表空间溢出而无法正常工作<sup>[6]</sup>。在控制层,攻击者通过向控制器持续发送 packet-in 数据包,导致控制器资源耗尽

而拒绝服务<sup>[7]</sup>。针对该类型的 DDoS 检测方法可分为三类 (基于规则的检测方法、基于统计信息的检测方法和基于机器学习的检测方法)<sup>[8]</sup>,其本质是对 SDN 中的流量进行约束和测量。近年来,随着深度学习技术的不断发展,其相关技术广泛应用于 DDoS 攻击检测<sup>[9-12]</sup>。而针对该种类型的防御机制,主要依托于限速和迁移。Kandoi 等<sup>[13]</sup>提出利用一种通过限制数据传输速率来缓解数据层或控制层 DDoS 攻击的方法。乔思祎等<sup>[14]</sup>对 OpenFlow 交换机流表溢出问题的缓解机制进行了深入研究,提出流表共享 (FTS, flow table sharing) 方法,完善了流表溢出处理机制。武泽慧等<sup>[15]</sup>提出基于 OpenFlow 交换机洗牌的 DDoS 攻击动态防御方法,来缓解交换机遭遇的 DDoS 攻击。

事实上,SDN 控制层北向接口 (NBI, northbound interface) 的安全性同样对 SDN 的正常运行起关键作用。北向接口,作为控制层与应用层的通道,随着 SDN 应用的广泛普及,必然会受到越来越多攻击者的关注。然而,针对北向接口的防御研究还比较少,主要集中在利用访问控制来隔离攻击者。Wen 等<sup>[16]</sup>将控制器上的应用程序和控制器内核进行隔离,提出一个应用程序访

权限管理系统, 实现对应用程序权限的细粒度访问控制。与之类似的, 还有 Klaedtke 等<sup>[17]</sup>的工作。然而, 一些攻击者会通过研究访问控制规则, 设计相应的绕过机制, 从而使该类型的防御机制失效。鉴于此, 本文研究主要针对 SDN 应用层 DDoS 攻击的防御, 结合北向接口发展特点<sup>[18-20]</sup>, 提出一种基于应用程序接口 (API, application programming interface) 恶意调用的 SDN 应用层 DDoS 攻击场景。进一步地, 为了防御该种类型攻击, 提出基于 API 调用管理的 SDN 应用层 DDoS 防御机制, 该机制通过在 SDN 应用层和控制层之间增加一层 App 管理层, 进而通过对 App 的信誉管理、初始审查、映射分配、异常检测和识别迁移, 来抵抗恶意 App 对 SDN 的攻击。

## 1 SDN 架构及应用层 DDoS 攻击场景

SDN 架构的核心思想是通过分离网络控制与数据处理, 赋予网络可编程性, 提高网络的智能化与灵活性, 最终提高网络的整体运行效率。当前, SDN 的经典架构分为应用层、控制层和数据层。

应用层主要指实现特定功能的 SDN 应用程序。从当前发展来看, SDN 应用程序大多是为了实现网络的负载均衡、拓扑构建以及安全管理等功能。控制层作为 SDN 的核心, 其主要依据 SDN 应用需求, 通过生成能够用来控制细粒度网络流量编排的流表, 进而实现对网络的控制。数据层主要负责基于控制层下发流表的数据转发。上述三层由两个接口连接, 北向接口用于连接应用层与控制层, 而南向接口 (SBI, southbound interface) 用于连接控制层与数据层。

事实上, OpenFlow 已成为当前南向接口的标准, 针对南向接口的研究相对成熟, 其安全性也得到很多学者的关注。然而, 学者对北向接口的研究较少, 目前还没有形成得到广泛认可的标准, 其安全威胁更令人担忧。随着 SDN 技术的不断发展, 其应用必定会更加丰富多样, 而北向接口作为开放给用户的端口, 是用户通过可编程来实现自定义功能的重要通道。依据当前 SDN 技术的发展潮流, 灵活性将是 SDN 技术的灵魂。因此, 设

计北向接口的关键在于接口的可编程性与调用管理的便捷性。表征状态转移 (REST, representational state transfer) 作为设计开放灵活接口的约束准则, 受到研究者的普遍关注。越来越多的学者倾向于利用 REST 架构来定义 SDN 应用所需的各种 API, 进而使 SDN 应用层的应用程序能够直接调用 API 来实现应用程序的高效开发, 最终推动 SDN 技术的快速普及。基于该技术, SDN 应用层可事先实现安全状态信息收集、链路流量信息收集、流量规则冲突分析、流规则超时回调、流规则修改、交换机配置信息获取、控制器配置信息获取、IP 地址跳变等 API, 进一步以这些 API 为接口提供给 SDN 应用调用。软件定义网络中应用层工作时 API 调用时间序列示例如图 1 所示。

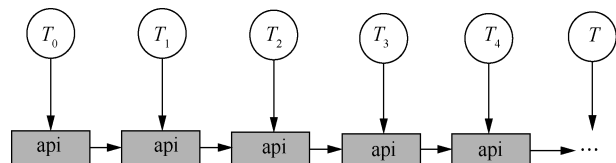


图1 软件定义网络中应用层工作时 API 调用时间序列示例  
Figure 1 Example of API call time sequence when SDN application layer works

由以上分析可知, SDN 北向接口的安全性直接关系到整个 SDN 正常运行。当前, 针对 SDN 北向接口安全性的研究还比较少, 缺乏严格的访问控制、身份认证及异常调用检测等机制, 这可能导致一些攻击者故意开发恶意的应用程序, 造成北向接口 API 的滥用, 以此实现对 SDN 的攻击。鉴于此, 本文研究主要针对 SDN 应用层 DDoS 攻击的防御。结合 SDN 网络架构, 模拟两种攻击者利用恶意的 API 调用实现 SDN 应用层的 DDoS 攻击场景, 如图 2 所示。

攻击场景 1: 在该攻击场景下, 攻击者设计恶意 App, 绕过北向接口的安全审查, 对某些 API 进行短时间大量调用, 进而导致控制器崩溃, 使整个网络瘫痪。

攻击场景 2: 在该攻击场景下, 攻击者以某个合法 SDN 应用程序作为攻击目标, 对该应用程序所需用的特定 API 进行短时间大量调用, 使该合法 App 无法正常调用 API, 进而使该合法 App 无法正常工作。与场景 1 相比, 该种攻击相对比较隐蔽。

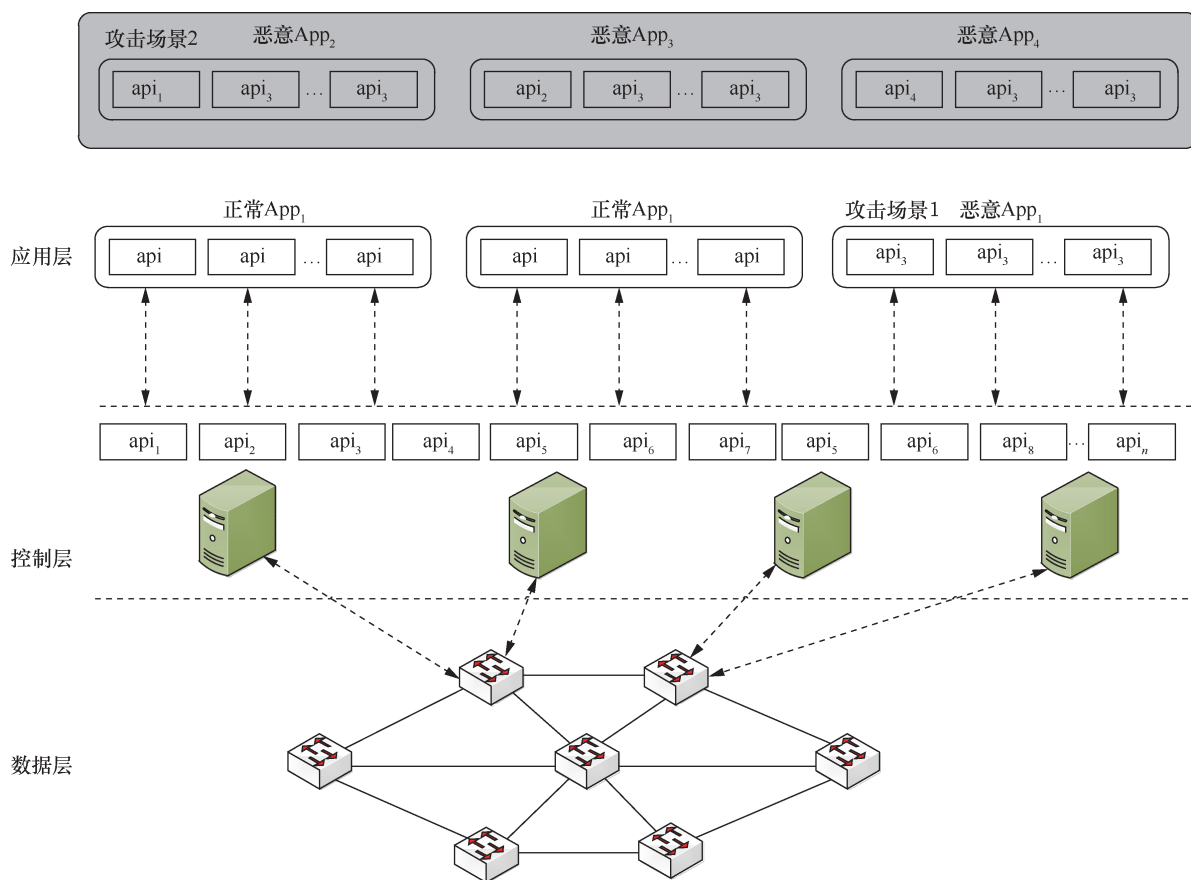


图2 软件定义网络中应用层 DDoS 攻击场景  
Figure 2 DDoS attack scenario in SDN application layer

## 2 API 可信调用机制

由上文可知,为抵抗攻击者利用恶意的 API 调用实现 SDN 应用层的 DDoS 攻击,对北向接口的安全管理成为防御的关键。鉴于此,本文提出一种基于 API 调用管理的 SDN 应用层 DDoS 防御机制,该机制通过在 SDN 应用层和控制层之间增加一层 App 管理层,进而通过对 App 的信誉管理、初始审查、映射分配、异常检测和识别迁移,来抵抗恶意 App 对 SDN 的攻击,整个架构如图 3 所示。App 管理层各个模块之间的协同关系如图 4 所示。

由图 3 和图 4 可知,为了实现对 SDN 北向接口的安全调用,提出的 App 管理层主要分为 App 运行前的合法审查、App 与控制器映射分配、App 运行时实时的异常检测、非法 App 的识别迁移以及 App 信誉度评分 5 个主要模块。其中,App 运行前的合法审查、App 与控制器映射分配、App 运行时实时的异常检测与恶意 App 的识别迁移

是按 App 能够正常运行在 SDN 中的逻辑先后顺序划分的:只有通过合法审查的 App 才能运行在 SDN 中;运行之前,需要 App 与控制器映射分配,以实现运行效率的最大化;运行过程中,需要对其进行实时的异常检测;一旦发现异常,则需要恶意 App 的识别迁移,以保证网络重新正常运行。App 信誉度评分则能够为 App 运行前的合法审查以及 App 与控制器映射分配提供依据。

### (1) App 运行前的合法审查

对于 SDN 应用层中的每一个 App,在其运行之前,均需要对其的 API 调用序列进行合法性检测,检测结果判定为合法或不合法。经判定不合法的 App 无法在应用层运行。

### (2) App 与控制器映射分配

随着 SDN 的不断发展,SDN 应用不断丰富,单一的控制网络架构往往无法满足网络实际需求,而多个控制器协同工作是 SDN 控制层发展的必然趋势。在此情况下,SDN 的应用层存在 App

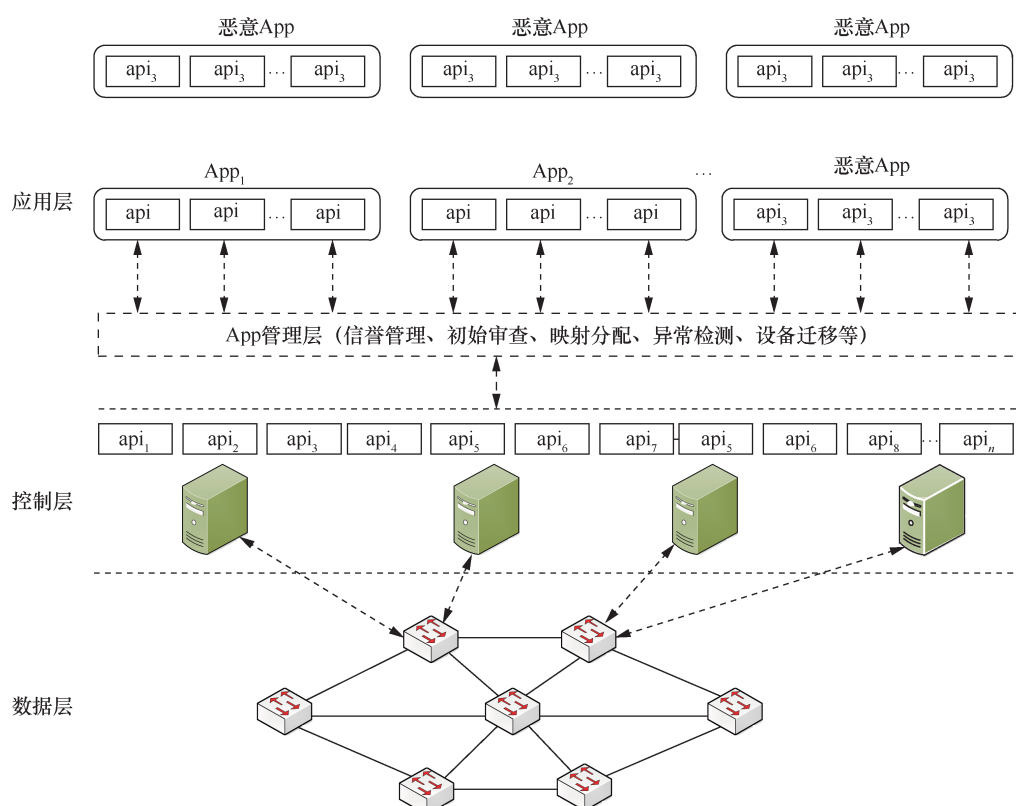


图 3 软件定义网络中应用层 DDoS 防御架构  
Figure 3 DDoS defense architecture of SDN application layer

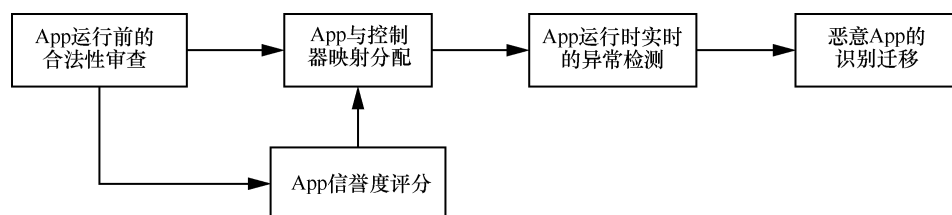


图 4 App 管理层各个模块之间的协同关系  
Figure 4 Synergy between modules of App management layer

与控制器映射分配的问题，即通过将当前网络所有运行的 App 合理分配给不同的控制器，实现控制器资源的最大化利用。

### (3) App 运行时实时的异常检测

尽管每一个 App 在 SDN 应用层运行之前均需进行合法性审查，然而事实上，任何审查均存在一定的漏报率和误报率。此外，由上文描述的攻击场景 2 可知，一些攻击者会精心设计恶意 App，使它们能够通过合法性审查而依然可对 SDN 实施攻击。鉴于此，SDN 需要实时对 App 运行状况进行异常检测。进一步分析可知，SDN 的应用层 DDoS 攻击最终均会导致控制器资源耗尽，从而使合法用户无法正常调用 API。因此，

通过对控制器运行状态的监视，能够及时发现当前 SDN 状态的异常。

### (4) 恶意 App 的识别迁移

由合法审查和异常检测模型可知，当恶意 App 绕过审查而运行在 SDN 应用层时，最终会导致控制器运行异常。本文仅考虑 SDN 应用层的攻击，即假定 SND 的控制层以及数据层不再遭遇其他类型的攻击。由映射分配模块可知，运行在同一个控制器上的 App 可能有多个，当该控制器异常时，反映出连接在该控制器上的 App 存在恶意 App，则非法 App 的识别迁移模块对这些 App 进行识别，即判断出哪些 App 是合法的而哪些 App 是恶意的，在此基础上，进一步暂停恶意 App 运

行而保证合法 App 的正常运行。

#### (5) App 信誉度评分

通过对上述模块分析可知,增加 App 管理层的目的是对恶意 App 的识别封禁,以保证合法 App 在 SDN 应用层的正常运行。在此过程中,防御者可对所有 App 进行统一的信誉度评分,对恶意 App 的来源进行标记,通过信誉度评分来提高对恶意 App 的检测效率。

通过以上分析可知,App 运行时的异常检测可通过对控制器运行状态的监视来实现,而当前针对 SDN 控制器异常检测的研究很多,本文不做重点研究。此外,针对 App 信誉度评分的研究,可参考类似的信誉度评价相关模型<sup>[21-23]</sup>,与 SDN 特性关联性不强。本文将 App 信誉度评分设定为 0 到 10 分,且将小于 5 分的 App 设为恶意 App,在此基础上,分别提出基于滑动时间窗口的 API 调用合法性审查、基于信誉度排序的 SDN 应用映射分配和恶意 App 清洗分离,以此实现完整的 App 管理层模块。针对多控制器之间的管理与通信研究较多<sup>[3,24]</sup>,这里不做深入探讨。

需要注意的是,软件定义网络应用层 DDoS 攻击与软件定义网络中 SDN 控制器是否是分布式的没有任何关系。即无论软件定义网络中 SDN 控制器是单个还是多个,攻击者均可对其发动应用层 DDoS 攻击。而本文强调的多控制器的目的是防御该种 DDoS 攻击的需要:当软件定义网络中的控制器仅有一个时,一旦攻击者对其发动应用层 DDoS 攻击,整个网络便会瘫痪;而当软件定义网络中的控制器有多个时,防御者可通过对恶意 App 清洗分离来保证整个网络的正常运行。

### 2.1 基于滑动时间窗口的 API 调用合法性审查

由上文可知,SDN 应用通过北向接口调用丰富的 API 以实现不同特定的功能。而针对 SDN 应用层的 DDoS 攻击,往往会开发恶意的 App,进而对某些 API 进行短时间大量调用,以此耗尽控制器资源,使整个网络崩溃。通过深入分析攻击过程可知,为了避免 API 被恶意调用而导致 SDN 的崩溃,需要使所有 App 在调用 API 时满足以下两个约束条件。

(1) 同一个 API 之间的调用周期约束  $T_{api_x} \geq$

$$T_{api_x}^{\min}$$

不妨假设北向接口中共有  $n$  个可供调用的 API,  $API = \{api_1, api_2, api_3, \dots, api_n\}$ 。为了使 SDN 正常运行,每个 API 设定一个最小调用周期限制,以保证应用的正常运行,即  $api_x$  的最小调用周期为  $T_{api_x}^{\min}$ ,如对于“IP 地址跳变”API 来说,当过于频繁调用该 API 时,SDN 中的节点地址跳变过于频繁,不能保证正常业务的运行。

(2) 不同 API 之间的调用资源约束  $R(T) \leq R^{\max}(T)$

在 SDN 中,由于控制器的高度集中性,特别对于单控制器 SDN,控制器具有一定的资源上限,若时间间隔  $T$  内,调用的 API 所需资源高于控制器资源上限,则控制器将无法正常运行,即  $R(T) \geq R^{\max}(T)$ 。不妨对每个 API 调用时所需资源进行量化,设调用  $api_x$  时所需资源为  $r_{api_x}$ 。进一步假设 SDN 正常运行时,某一  $T$  时间内调用的 API 序列为  $API(T) = \{api_1, api_2, api_3, \dots, api_m\}$ ,则易知

$$R(T) = r(api_1) + r(api_2) + \dots + r(api_m) \leq R^{\max}(T) \quad (1)$$

以上两个约束条件从 API 本身特性及其对资源需求角度对其正常调用进行了规定。显然,符合两个约束条件的 App 为合法 App,不符合的则为恶意 App。

进一步,本文以上述两个约束条件为依据,提出了基于滑动时间窗口的 API 调用合法性审查算法,如算法 1 所示。

**算法 1** 基于滑动时间窗口的 API 调用合法性审查算法

**输入** 某个 App<sup>m</sup> 的 API 调用时间序列  $App^m = \{api_1^m, api_2^m, api_3^m, \dots, api_k^m\}$ , 各个 API 的最小调用周期  $T_{api_x}^{\min}$ , 控制器调用资源约束  $R^{\max}(T)$

**输出** Result=Yes or No—App<sup>m</sup> 是否合法

1) Result=Yes//初始化

2) for 每一个  $api_x^m \in App^m = \{api_1^m, api_2^m, api_3^m, \dots, api_k^m\}$

3) if  $\text{sum}(t(api_x^m), t(api_x^m) + T_{api_x^m}^{\min}) > 1$ //表示从时

刻  $t(\text{api}_x^m)$  到  $t(\text{api}_x^m) + T_{\text{api}_x^m}^{\min}$  调用的  $\text{api}_x^m$  个数大于 1

4) Result=No

5) end if 3)

6) end for 2)

7) for 每一个  $\text{api}_x^m \in \text{App}^m = \{\text{api}_1^m, \text{api}_2^m, \text{api}_3^m, \dots, \text{api}_k^m\}$

8) if  $R(t(\text{api}_x^m), t(\text{api}_x^m) + T) > R^{\max}(T)$  //表示从时刻  $t(\text{api}_x^m)$  到  $t(\text{api}_x^m) + T$  调用的 API 所需要花费的资源之和大于调用资源约束

9) Result=No

10) end if 8)

11) end for 7)

12) Return Result

显然，参数  $T_{\text{api}_x}^{\min}$  和  $R^{\max}(T)$  的值直接影响了算法 1 的输出结果，即准确设定参数  $T_{\text{api}_x}^{\min}$  和  $R^{\max}(T)$  的值是算法 1 正确识别 App 是否合法的前提。需要说明的是，对于不同的 SDN，作为网络的防御参数， $T_{\text{api}_x}^{\min}$  和  $R^{\max}(T)$  的值不一定相同，其值一般可依据具体的 SDN 进行实际运行测试得来，而该测试可事先实施，不会造成算法效率降低。

## 2.2 基于信誉度排序的 SDN 应用映射分配

为满足日益增长的 SDN 应用层需要，SDN 控制平面往往会采取多个控制器协同工作的方式，且多个控制器实现负载均衡<sup>[25-26]</sup>。在此情况下，需要将所有 SDN 应用映射分配到不同的控制器，使整个 SDN 负载均衡，最终实现网络资源利用的最优化，SDN 应用映射分配示例如图 5 所示。

由图 5 可知，同一个控制器上往往分配有多个 App，显然，这些 App 同时运行时需要满足算法 1 的两个约束条件，才能保证控制器的正常运行。事实上，对 App 的合法性审查本质上是对其调用的 API 时间序列进行审查。由图 1 可知，将多个 App 的 API 调用时间序列进行合并，并依据 API 调用时间由小到大排列，可形成多个 App 共同运行时的 API 调用时间序列。由此可知，利用算法 1 能够实现对多个 App 能否共同运行做出判断。此外，为了使整个控制平面实现负载均衡，从资源利用的角度，各

个控制器映射分配过程需要考虑控制器负载。鉴于此，本文提出基于信誉度排序的 SDN 应用层映射分配算法。

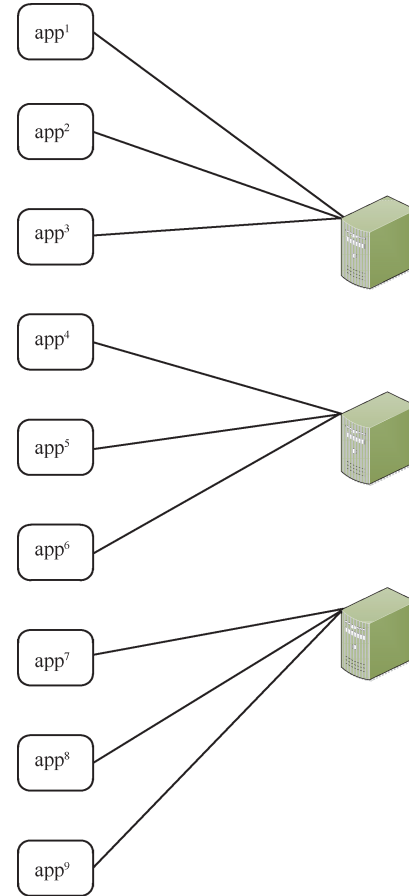


图 5 SDN 应用映射分配示例

Figure 5 Example of SDN application mapping allocation

**算法 2** 基于信誉度排序的 SDN 应用层映射分配算法

**输入** 目前待映射分配的 SDN 应用集合  $\text{APP} = \{\text{app}^1, \text{app}^2, \text{app}^3, \dots, \text{app}^m\}$ ，控制器集合  $C = \{c_1, c_2, c_3, \dots, c_n\}$ ，控制器调用资源约束  $R^{\max}(T)$ 。

**输出** SDN 应用集合与控制器集合间的映射分配关系  $\text{App} \rightarrow C$

1) 将待映射分配的 SDN 应用集合中的 app 按信誉度排序

2) for 每一个  $\text{app}^i \in \text{APP}$  //令  $\text{Seq}(\text{app}^i)$  表示应用  $\text{app}^i$  的 API 调用时间序列

3) for 每一个  $c_i \in C$  //假定将  $\text{app}^i$  映射分配至控制器  $c_i$ ，此时控制器  $c_i$  运行的 app 集合记

为  $c_i(\text{app})$ , 则其 API 调用时间序列记为  $\text{Seq}(c_i)$

4) for 每一个  $\text{api}_x \in \text{Seq}(c_i)$

5) if  $\text{sum}(t(\text{api}_x), t(\text{api}_x) + T_{\text{api}_x}^{\min}) > 1$  // 表示从时刻  $t(\text{api}_x)$  到  $t(\text{api}_x) + T_{\text{api}_x}^{\min}$  调用的  $\text{api}_x^m$  个数大于 1

6)  $C = C \setminus c_i$  // 说明控制器  $c_i$  不适合, 将其去除

7) end if 5)

8) end for 4)

9) end for 3)

10) for 每一个  $c_j \in C$  // 假定将  $\text{app}^l$  映射分配至控制器  $c_j$

11) for 每一个  $\text{api}_x \in \text{Seq}(c_j)$

12) if  $R(t(\text{api}_x), t(\text{api}_x) + T) > R^{\max}(T)$  //

13)  $C = C \setminus c_j$  // 说明控制器  $c_j$  不适合, 将其去除

14) end if 12)

15) end for 11)

16) end for 10)

17) if  $C \neq \emptyset$

18) for 每一个  $c_k \in C$  假定将  $\text{app}^l$  映射分配至控制器  $c_k$

19)  $R_k = \emptyset$

20) for 每一个  $\text{api}_x \in \text{Seq}(c_j)$

21)  $R_k = [R_k, R(t(\text{api}_x), t(\text{api}_x) + T)]$

22) end for (20)

23) end for 18)

24)  $\text{app}^l \rightarrow c_{\arg \min_k \text{diff}(\text{App}^l \rightarrow c_k)}$  // 将  $\text{App}^l$  分配至负载均衡最好的控制器中

25) end if 17)

26) if  $C = \emptyset$  // 说明当前没有合适的控制器

27) Pause and remove the app with the smallest 信誉值 // 强制暂停并从控制器中去除当前信誉度最低的 SDN 应用

28) end if 26)

29) Go to 2)

30) end for 2)

31) Return  $\text{App} \rightarrow C$

在算法 2 中, 整个 SDN 中的负载均衡用  $\text{diff}$  表示, 其可从两个方面进行量化: 一是同一个控制器中运行的 App 构成的 API 序列调用相对平滑; 二是不同控制器间资源利用相对均衡。具体

量化方法如下。

(1) 设  $R_k$  为时间序列, 用于表示控制器在不同时间间隔  $T$  所占用的资源, 用  $\text{diff}_1(R_k)$  来量化该时间序列的平稳度, 其值越小, 说明控制器运行越平稳, 负载均衡越好。通过实际 SDN 运行分析可知,  $\text{diff}_1(R_k)$  的确定方法如下。

$$\text{diff}_1(R_k) = \frac{\sum (x - \mu_k)^2}{|R_k|} \quad (2)$$

其中,  $|R_k|$  表示时间序列  $R_k$  中的元素个数,  $x$  表示时间序列  $R_k$  中的各个元素,  $\mu_k$  表示时间序列  $R_k$  中的元素均值。

(2) 用  $\text{diff}_2(C)$  表示不同控制器中负载均衡程度。为简单起见, 本文采用不同控制器的时间序列  $R_k$  中的元素均值之间方差来进行量化, 即

$$\text{diff}_2(C) = \frac{\sum (\mu - \mu_c)^2}{|C|} \quad (3)$$

其中,  $|C|$  表示控制器的个数,  $\mu$  表示不同控制器的时间序列  $R_k$  中的元素均值,  $\mu_c$  表示所有控制器的时间序列  $R_k$  中元素均值的均值。

(3) 依据不同 SDN 实际需求, 分别为上述两个要素设定不同的权重, 进而利用权重和得到  $\text{diff}$ 。

$$\text{diff} = \alpha \cdot \text{diff}_1(R_k) + \beta \cdot \text{diff}_2(C) \quad (4)$$

其中,  $\alpha$  和  $\beta$  分别为上述两要素的权重, 满足  $\alpha + \beta = 1$ 。

### 2.3 恶意 App 清洗分离

当一个 SDN 控制器上运行有恶意 App 时, 该 SDN 控制器由于资源耗尽而无法正常工作, 该 SDN 控制器被称为受攻击控制器。在该种情况下, 由于该控制器上运行着多个 App, 防御者无法分辨哪些 App 是恶意的, 哪些 App 是合法的。鉴于此, 如何对受攻击控制器上运行的 App 进行快速清洗以分离出恶意 App, 并进一步对合法 App 重新分配控制器以保证其正常运行, 是一个值得研究的问题。SDN 应用层恶意 App 清洗分离过程示例如图 6 所示。

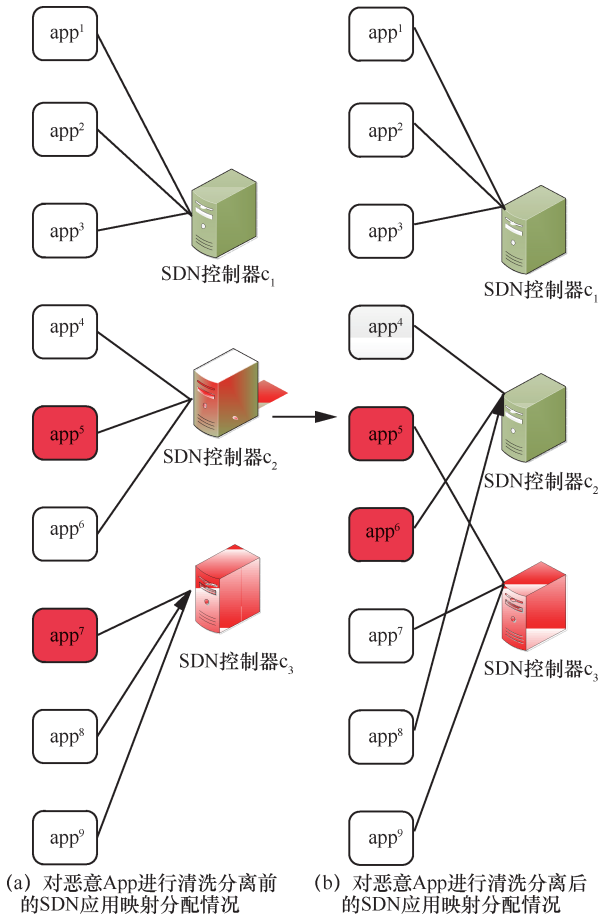


图6 SDN应用层恶意App清洗分离过程示例  
Figure 6 Example of cleaning and separating malicious App in SDN application layer

图6中共有9个SDN应用,其中包含两个恶意的App,即app<sup>5</sup>和app<sup>7</sup>,控制平面有3个控制器。经过初始的映射分配,app<sup>1</sup>、app<sup>2</sup>和app<sup>3</sup>分配到控制器c<sub>1</sub>,app<sup>4</sup>、app<sup>5</sup>和app<sup>6</sup>分配到控制器c<sub>2</sub>,app<sup>7</sup>、app<sup>8</sup>和app<sup>9</sup>分配到控制器c<sub>3</sub>。当SDN运行时,由于恶意App的存在,控制器c<sub>2</sub>和c<sub>3</sub>无法正常工作。而防御者无法识别连接到这两个App上的应用程序哪些是合法的,哪些是恶意的,如图6(a)所示。在此情况下,防御者实施恶意App清洗分离过程,如图6(b)所示,随机将app<sup>5</sup>分配到控制器c<sub>3</sub>,而将app<sup>8</sup>分配到控制器c<sub>2</sub>。在此情况下,控制器c<sub>2</sub>将恢复正常运行,而控制器c<sub>3</sub>将依然无法正常运行。由此可知,在SDN遭遇攻击时,恶意App清洗分离能有效分离出合法App和恶意App。为了最大化清洗分离效率,本文对此过程进行如下建模。

设需要清洗分离的App集合为A,可供清洗

分离的SDN控制器集合为C,控制器编号为1到|C|,其中|C|为控制器数量。每一轮的清洗分离可看作一个映射f。对于任一个控制器c<sub>i</sub>∈C,分配给该控制器的App集合记为A<sub>i</sub>=f(c<sub>i</sub>),显然A<sub>i</sub>∈A。由于每一个App只能分配给一个控制器,则

$$A_i \cap A_j = \emptyset, \forall 1 \leq i, j \leq |C| \text{ 且 } i \neq j \quad (5)$$

$$\bigcup_{1 \leq i \leq |C|} A_i = A \quad (6)$$

进一步,设恶意App数量为N<sub>v</sub>,设经过一轮清洗分离操作后,仍有N<sub>a</sub>个合法的App无法正常运行(被攻击),而有N<sub>u</sub>个合法App可以正常运行,显然,|A|=N<sub>a</sub>+N<sub>u</sub>+N<sub>v</sub>,其中|A|为所有待分配App数量。本文的目标是设计最优的清洗分离方法f,以使在各种条件下N<sub>u</sub>的E(N<sub>u</sub>)期望最大。如前文所述,对于任一个控制器c<sub>i</sub>∈C,分配给该控制器的App集合记为A<sub>i</sub>=f(c<sub>i</sub>),进一步,不妨设A<sub>i</sub>={app<sub>i</sub><sup>1</sup>,app<sub>i</sub><sup>2</sup>,...,app<sub>i</sub><sup>|A<sub>i</sub>||</sup>}。由于只有当一个控制器上分配的所有App均为合法App时,该控制器才能正常工作。相反,只要一个控制器上分配有一个恶意App,该控制器便无法正常工作,则控制器c<sub>i</sub>能正常工作的概率记为

$$p_i = \frac{\text{credit}^{\text{app}_i^1}}{10} \cdot \frac{\text{credit}^{\text{app}_i^2}}{10} \cdots \frac{\text{credit}^{\text{app}_i^{|A_i|}}}{10} \quad (7)$$

经过A<sub>i</sub>=f(c<sub>i</sub>)操作后,能够分离出来的合法App数量的期望为p<sub>i</sub>·|A<sub>i</sub>|。进一步,易知

$$E(N_u) = \sum_{i=1}^{|C|} p_i \cdot |A_i| \quad (8)$$

由于控制器资源的限制,当将App集合A<sub>i</sub>分配给该控制器c<sub>i</sub>时,存在即使所有A<sub>i</sub>中的App均为合法App而控制器也无法正常工作的情况,即控制器c<sub>i</sub>无法满足App集合A<sub>i</sub>中所有App同时正常运行。鉴于此,在清洗分离过程中,需要保证分配给控制器的App集合不能超出该控制器的资源限制。该条件可由算法1来判断,即可表示为

$$\text{Algorithm 1 } (c_i, A_i) == \text{Yes} \quad (9)$$

综上所述可知,对恶意App清洗分离问题可视为一个最优化问题,其目标函数为方程(8),变量为分配方案f,约束条件为式(9)。直观上,可供清洗分离的SDN控制器数量越多,则经过每一

轮清洗分离后合法 App 被分离出来的数量越多。若  $|C| \geq |A|$ ，则每一个 App 都能被分配到一个专有的 SDN 控制器上，在该种情况下， $E(N_u) = |A| - N_v$ ，它意味着所有合法的 App 均能正常运行，而所有恶意的 App 均被一轮清洗分离而分离出来。然而，在实际的 SDN 中，控制器的数量一般较少，甚至是一个控制器的情况，而 App 的个数一般远远多于控制器的个数。考虑到此，如何设计清洗分离算法使尽快识别出恶意 App 是十分重要的。对于  $|C| \geq 1$  时，目标函数是非线性的，且该类问题是 NP 难问题。对于该问题，本文利用贪心思想计算得出该方案的接近最优解，其具有多项式时间。该恶意 App 清洗分离算法如下。

### 算法 3 恶意 App 清洗分离算法

**输入** 目前待清洗分离的 SDN 应用集合  $A$ ，可用的控制器集合  $C$

**输出** 各个控制器上分配的 SDN 应用集合，用  $f$  表示，即  $f(c_i)$  表示控制器上分配的 App 集合

- 1) 将待映射分配的 SDN 应用集合  $A$  中的所有 app 按信誉度排序形成 AppList
- 2) if  $\text{length}(\text{AppList}) \leq |C|$
- 3) 为每一个 App 分配一个控制器
- 4) else
- 5) for  $i=1: |C|$  do //  $|C|$  表示集合  $C$  中的元素总个数

- 6)  $f(c_i) = \text{Appassign}(c_i, \text{AppList})$

- 7)  $\text{AppList} = \text{AppList} \setminus f(c_i)$  // 将当前已分配完成的 App 从列表中删除

- 8) end

1) Function Appassign (ControlID, AppList) // ControlID 表示需要分配 App 的控制器，AppList 表示当前所有待清洗分离的 App 列表，该列表以 App 信誉度评分从高到低排列

- 2)  $\text{Save} = []$ ,  $\text{AppSet} = []$

- 3) for  $j=1: \text{length}(\text{AppList})$  do

- 4)  $\text{Save}(j) = \frac{\text{credit}^1}{10} \cdot \frac{\text{credit}^2}{10} \cdots \frac{\text{credit}^j}{10} \cdot j$

- 5) end

- 6)  $U = \arg \max_{1 \leq j \leq \text{length}(\text{AppList})} \text{Save}(j)$

- 7) If Algorithm 1 (ControlID, AppList( $U$ )) == Yes // 通过算法 1 能够得出分配的 App 集合所调用的 API 序列是否适应控制器 ControlID 的正常工作要求

- 8)  $\text{AppSet} = \text{AppList}(U)$  // 将当前所有待清洗分离的 App 列表中的前  $U$  个 App 分配给控制器 Control ID

- 9) end 7)

- 10) else if Algorithm 1 (ControlID, AppList( $U$ )) == No

- 11)  $\text{Save}(U) = 0$

- 12) Go to 6)

- 13) end 10)

- 14) Return AppSet

由此，可依据每一轮分配后的控制器能否正常工作，来清洗分离出恶意 App。经过多轮的清洗分离操作，最终将会使所有恶意 App 均被分离识别。

此外，SDN 中的恶意 App 数量越多，越要保证清洗分离尽快完成，则需要越多可供分配的控制器。因此，清洗之前，对 SDN 中的恶意 App 数量进行评估，能够为设定合适的控制器数量提供依据。需要注意的是，若一个控制器上所有的 App 均为合法 App，则该控制器正常运行，不参加恶意 App 的清理分离任务；若一个控制器上至少存在一个恶意 App，则该控制器上的其他合法 App 也无法正常工作，而通过不断的清洗分离，这些合法的 App 将被分配至其他的控制器，最终得以正常工作。由此可见，恶意 App 清洗分离过程可能会导致无法正常工作的合法 App 频繁切换控制器，但不会对本身正常运行的合法 App 造成影响。

不妨设  $\text{app}^i$  的信誉度评分为  $\text{credit}^{\text{app}^i}$ ，则一个 App 为恶意的概率可记为

$$P(\text{app}_{\text{vicious}}^i) = \frac{10 - \text{credit}^{\text{app}^i}}{10} \quad (10)$$

一个 App 为正常的概率为

$$P(\text{app}_{\text{normal}}^i) = \frac{\text{credit}^{\text{app}^i}}{10} \quad (11)$$

其中， $\text{app}_{\text{vicious}}^i$  表示  $\text{app}^i$  为恶意 App， $\text{app}_{\text{normal}}^i$  表示

$app^l$  为正常 App。 $credit^l$  表示  $app^l$  的信誉度评分。不妨设某个受攻击的 SDN 控制器  $c_i$  上运行有  $M$  个 App,  $app^l$  的信誉度评分记为  $credit^l$ , 其中  $1 \leq l \leq M$ 。进一步, 设该控制器上的恶意 App 数量为变量  $X$ , 显然  $1 \leq X \leq M$ 。不妨设  $X=x$ , 即该控制器上的恶意 App 数量为  $x$  个。令  $U_x^1 = \{app_{vicious}^1,$

$app_{vicious}^2, app_{vicious}^3, \dots, app_{vicious}^x\}$  表示其中的一种恶意 App 集合, 则其对应的正常 App 集合可记为  $U_{M-x}^1 = \{app_{normal}^1, app_{normal}^2, app_{normal}^3, \dots, app_{normal}^{M-x}\}$ 。显然,  $U_x^1$  是从  $M$  个 App 中随机选取  $x$  个 App 作为恶意 App 而得来的, 则类似这样的  $U_x^1$  共有  $C_M^x$  个, 即  $U_x^1, U_x^2, \dots, U_x^{C_M^x}$ 。由排列组合可知

$$\Pr(X = U_x^1) = \frac{10 - credit^{App_{vicious}^1}}{10} \cdot \frac{10 - credit^{App_{vicious}^2}}{10} \cdot \frac{10 - credit^{App_{vicious}^3}}{10} \cdots \frac{10 - credit^{App_{vicious}^x}}{10} \cdot \frac{credit^{App_{normal}^1}}{10} \cdot \frac{credit^{App_{normal}^2}}{10} \cdot \frac{credit^{App_{normal}^3}}{10} \cdots \frac{credit^{App_{normal}^{M-x}}}{10} \quad (12)$$

$x$ 个  
 $(M-x)$ 个

于是

$$\Pr(X = x) = \underbrace{\Pr(X = U_x^1) + \Pr(X = U_x^2) + \Pr(X = U_x^3) + \cdots + \Pr(X = U_x^{C_M^x})}_{C_M^x \text{个}} \quad (13)$$

进一步, 利用最大似然估计原理, 可得

$$X^* = \arg \max_{1 \leq x \leq M} \Pr(X = x) \quad (14)$$

其中,  $X^*$  表示控制器  $c_i$  上恶意 App 数量的估计值。

### 3 实验与验证

针对 SDN 应用层的研究比较少, 基于 REST API 调用的 SDN 应用较缺乏。本文借鉴已被学者认可的大规模网络端地址跳变实现机理<sup>[27]</sup>, 通过仿真实现了一个使目标 SDN 所有终端地址跳变的 API。具体地, 借助 Mininet 创建 SDN 拓扑, RYU 作为控制器, 如图 7 所示。

图 7 所示实验网络中的终端数量可动态调整, 正常情况下, 各终端之间能够维持通信, 从而保证正常的业务活动。当 SDN 应用程序调用跳变 API 时, 该网络中所有终端的 IP 地址会发生跳变, 而后该 SDN 仍能正常工作。然而事实上, 若攻击者制作一个恶意 App, 该 App 短时间大量调用跳变 API, 显然会造成网络崩溃, 这种情况相当于攻击者从 SDN 应用层发起了 DDoS 攻击。本文实验部分所配置的控制器个数是动态变化的, 而图 7 为了便于画图需要, 又为体现多个物理控制器的逻辑集中, 故用一个控制器代替。为了进一步验证 SDN 应用层攻击会为网络正常运行造成影响, 首先

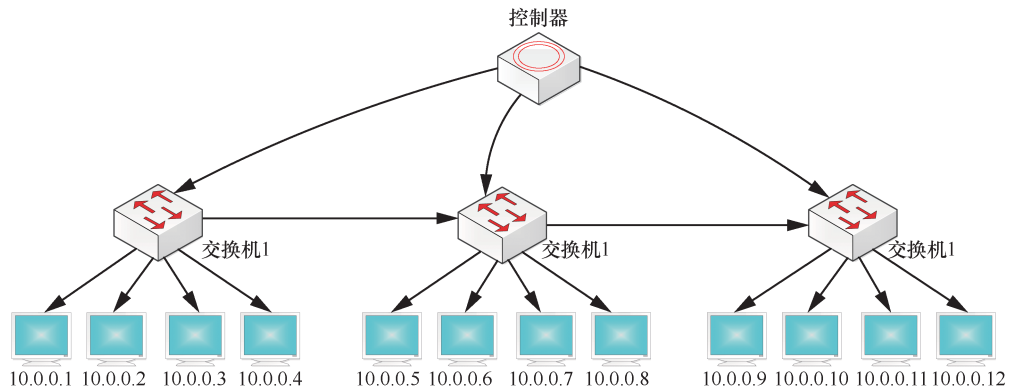


图 7 实验网络的拓扑结构  
Figure 7 Topology of experimental network

对网络正常运行的条件进行定量表示：模拟 SDN 业务需求，设定两台主机维持运行服务 4 h，每隔 10 min 服务一次，每次服务需要维持一定的服务时间 (ST, service time)。若网络达到该要求，即可认为网络正常运行。进一步，通过仿真得到，不同条件下 SDN 服务正常运行率与跳变 API 调用频率 (次数/小时) 之间的关系如图 8 所示。

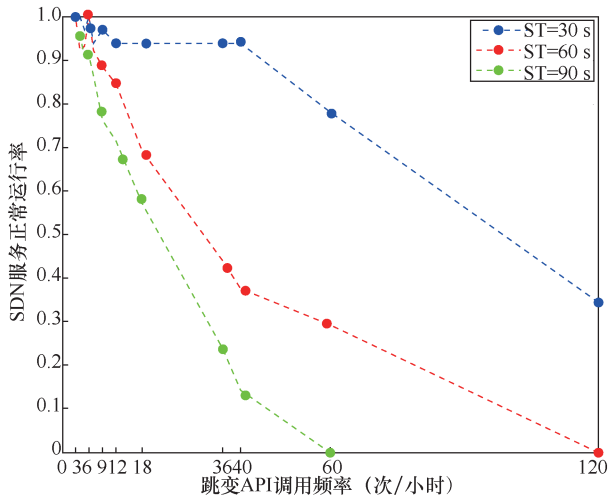


图 8 不同条件下 SDN 服务正常运行率与跳变 API 调用频率之间的关系

Figure 8 Relationship between normal operation rate of SDN service and call frequency of hopping API under different conditions

由图 8 可知，对于不同的服务时间要求，SDN 服务正常运行率均随跳变 API 调用频率的增大而减小，且当跳变 API 调用频率达到一定值后，SDN 服务正常运行率会变得很小，即不能正常工作。由此可见，当 SDN 应用通过短时间内高频率的调用 API 能够使 SDN 运行的正常服务瘫痪，即造成 SDN 应用层 DDoS 攻击。由此，可通过对具体的 SDN 进行实际运行测试，以此来准确设定参数  $T_{api_i}^{\min}$  和  $R^{\max}(T)$  的值。

通过对现有文献进行分析，针对 SDN 北向接口安全性的研究比较少，Oktian 等<sup>[28]</sup>提出了一种针对 SDN 北向接口 API 身份认证的方法。但其存在两方面缺陷：一是认证方法仅是理论上的且设计相对简单，实际的应用效果并不明确；二是对于前文提出的攻击场景 2 中攻击方法无效，一些攻击者会采用分布式的攻击方法，通过构造多个看似合法的 App（每个 App 都能通过身份认证），并利用这些 App 同时调用同一个 API，以实现对 SDN 应用层的攻击。事实上，本文提出的基

于 API 调用管理的 SDN 应用层 DDoS 攻击防御机制，侧重于在攻击发生前对恶意 App 进行事先审查，以避免攻击的发生。其次，若攻击发生，需要对合法 App 和恶意 App 进行快速清洗分离。本文算法对上文提出的两个攻击场景均有较好的防御效果。此外，算法的运行效率是验证该防御机制有效性的关键。鉴于此，对上一节所示 3 个算法运行效率进行了仿真测试。API 序列长度对 API 调用合法性检查所用时间的影响如图 9 所示。

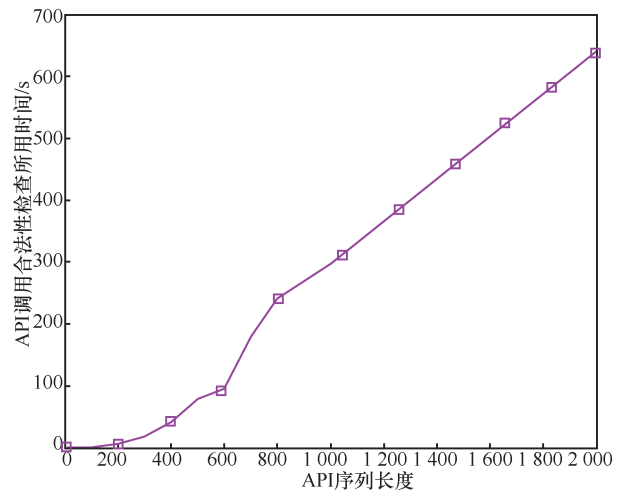


图 9 API 序列长度对 API 调用合法性检查所用时间的影响

Figure 9 The influence of API sequence length on the time for checking the validity of API calls

在本文实验中，App 序列长度指的是 SDN 应用程序所调用的 API 数量，如某一 App 的调用 API 序列为  $API = \{api_1, api_2, api_3, \dots, api_m\}$ ，则称该 App 序列长度为  $m$ ，即 App 序列长度可认为是该 App 调用的 API 序列长度。显然，App 序列长度对算法运行效率有较大的影响。由图 9 实验结果知，当 API 序列长度小于 600 时，API 调用合法性检查算法所需时间在 100 s 以内。事实上，该审查算法一般在 App 运行之前进行，对实时性要求不高；其次，通过分析已有的 SDN 应用，其 API 调用长度一般在 100 以内。综上所述，本文所提算法能够满足实用要求。

进一步，当设定 App 数量为 100、SDN 控制器数量为 4 时，API 序列长度对映射分配所用时间的影响如图 10 所示；当设定 App 数量为 100、App 序列长度为 10 时，SDN 控制器个数对映射分配所用时间的影响如图 11 所示；当设定 App 序列长度 10、SDN 控制器数量为 4 时，

App 个数对映射分配所用时间的影响如图 12 所示。

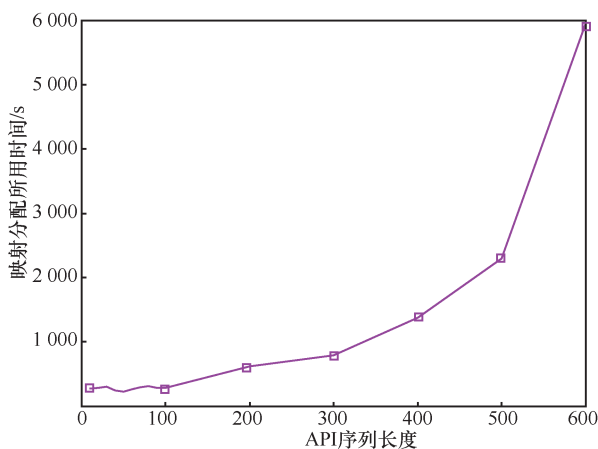


图 10 API 序列长度对映射分配所用时间的影响

Figure 10 Influence of API sequence length on mapping allocation time

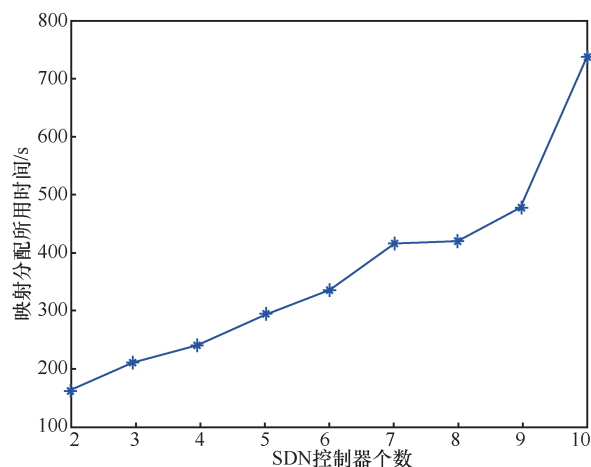


图 11 SDN 控制器个数对映射分配所用时间的影响

Figure 11 Influence of SDN controller number on mapping allocation time

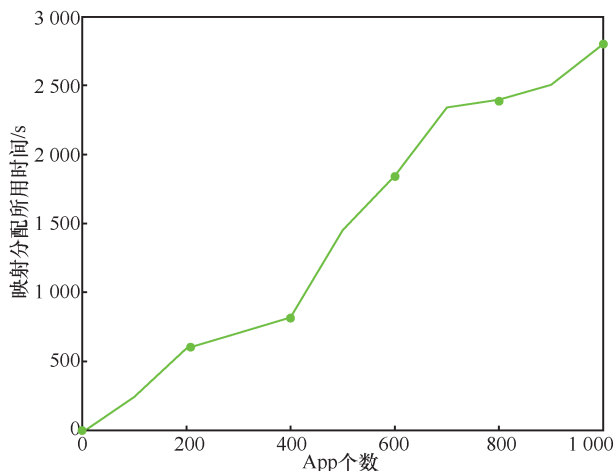


图 12 App 个数对映射分配所用时间的影响

Figure 12 Influence of the number of Apps on the time of mapping and allocation

图 10~图 12 中所示的 API 序列长度指的是所有 App 序列长度均保持一致。由图 10 可知, 当 API 序列长度维持在 300 以内时, 映射分配所用时间维持在 1 000 s 以内, 而当 API 序列长度继续增长时, 映射分配所用时间呈指数增加; 由图 11 可知, 当 API 序列长度、App 数量一定时, 映射分配所用时间随 SDN 控制器数量近似呈线性增加; 同样地, 由图 12 可知, 当 API 序列长度、SDN 控制器数量一定时, 映射分配所用时间随 App 数量近似呈线性增加。由上述 3 个图可知, App 序列长度对映射分配所用时间的影响最大。事实上, 对于 SDN 来说, 正常运行时, App 数量一般不会超过 200, API 序列长度不会超过 100。在该种情况下, 无论 SDN 控制器数量多少, 映射分配所用时间均在可接受时间内, 能够满足实际网络需求。

最后, 本文对恶意 App 清洗分离算法进行了仿真测试, 设定正常 App 数量为 100, 恶意 App 数量为 10, App 序列长度为 10, 则得出 SDN 控制器个数对清洗分离次数的影响如图 13 所示, SDN 控制器个数对清洗分离所需时间的影响如图 14 所示。

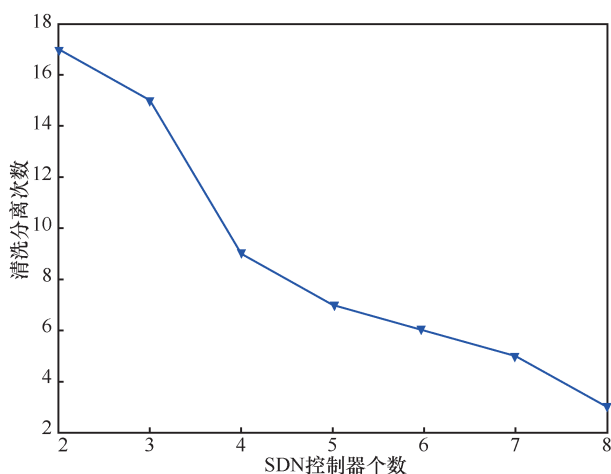


图 13 SDN 控制器个数对清洗分离次数的影响

Figure 13 Influence of the number of SDN controllers on cleaning and separation times

由图 13 和图 14 可知, 当 SDN 中正常 App 数量、恶意 App 数量以及 App 序列长度一定时, 清洗分离次数以及清洗分离所需时间均随 SDN 控制器个数的增加而降低, 且清洗分离所需时间低于 1 000 s。由此可知, 在 SDN 中, 通过部署

多个 SDN 控制器，能够加快清洗分离所需时间，降低 DDoS 攻击对目标网络的影响。

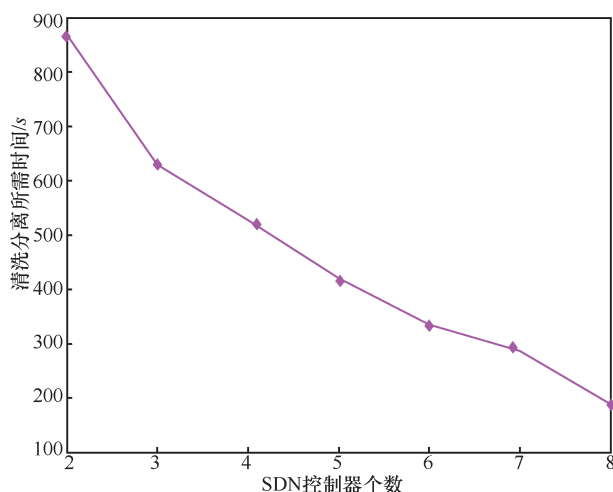


图 14 SDN 控制器个数对清洗分离所需时间的影响

Figure 14 Influence of the number of SDN controllers on the time required for cleaning and separation

## 4 结束语

针对 DDoS 攻击对北向接口造成的威胁日益扩大的问题，本文模拟了对其可能的 DDoS 攻击样态，并据此建立了基于 API 调用管理的 SDN 应用层 DDoS 防御机制。该机制通过在 SDN 应用层和控制层之间增加一层 App 管理层，进而通过对 App 的信誉管理、初始审查、映射分配、异常检测和识别迁移，来抵抗恶意 App 对 SDN 的攻击。实验结果表明，本文提出的安全防御方案能够有效地对 SDN 应用层的 DDoS 攻击进行防御。

## 参考文献:

- [1] JAIN R. Internet 3.0: ten problems with current Internet architecture and solutions for the next generation[C]//Proceedings of MILCOM 2006 IEEE Military Communications Conference. 2006: 1-9.
- [2] 王涛, 陈鸿昶, 程国振. 基于网络资源管理技术的 SDN DoS 攻击动态防御机制[J]. 计算机研究与发展, 2017, 54(10): 2356-2368.  
WANG T, CHEN H C, CHENG G Z. A dynamic defense mechanism for SDN DoS attacks based on network resource management technology[J]. Journal of Computer Research and Development, 2017, 54(10): 2356-2368.
- [3] 张朝昆, 崔勇, 唐嵩祯, 等. 软件定义网络(SDN)研究进展[J]. 软件学报, 2015, 26(1): 62-81.  
ZHANG C K, CUI Y, TANG H Y, et al. State-of-the-art survey on software-defined networking (SDN)[J]. Journal of Software, 2015, 26(1): 62-81.
- [4] 王蒙蒙, 刘建伟, 陈杰, 等. 软件定义网络: 安全模型、机制及研究进展[J]. 软件学报, 2016, 27(4): 969-992.  
WANG M M, LIU J W, CHEN J, et al. Software defined networking: security model, threats and mechanism[J]. Journal of Software, 2016, 27(4): 969-992.
- [5] 龚庆祥. 软件定义网络中分布式拒绝服务攻击研究[D]. 深圳: 深圳大学, 2017.  
GONG Q X. Research on distributed denial of service attacks in software defined networking[D]. Shenzhen, China: Shenzhen University, 2017.
- [6] WU X T, LIU M, DOU W C, et al. DDoS attacks on data plane of software-defined network: are they possible[J]. Security and Communication Networks, 2016, 9(18): 5444-5459.
- [7] 徐玉华, 孙知信. 软件定义网络中的异常流量检测研究进展[J]. 软件学报, 2020, 31(1): 183-207.  
XU Y H, SUN Z X. Research development of abnormal traffic detection in software defined networking[J]. Journal of Software, 2020, 31(1): 183-207.
- [8] DAYAL N, MAITY P, SRIVASTAVA S, et al. Research trends in security and DDoS in SDN[J]. Security and Communication Networks, 2016, 9(18): 6386-6411.
- [9] 张龙, 王劲松. SDN 中基于信息熵与 DNN 的 DDoS 攻击检测模型[J]. 计算机研究与发展, 2019, 56(5): 909-918.  
ZHANG L, WANG J S. DDoS attack detection model based on information entropy and DNN in SDN[J]. Journal of Computer Research and Development, 2019, 56(5): 909-918.
- [10] 田俊峰, 齐鏊岭. SDN 中基于条件熵和 GHSOM 的 DDoS 攻击检测方法[J]. 通信学报, 2018, 39(8): 140-149.  
TIAN J F, QI L L. DDoS attack detection method based on conditional entropy and GHSOM in SDN[J]. Journal on Communications, 2018, 39(8): 140-149.
- [11] 李传煌, 吴艳, 钱正哲, 等. SDN 下基于深度学习混合模型的 DDoS 攻击检测与防御[J]. 通信学报, 2018, 39(7): 176-187.  
LI C H, WU Y, QIAN Z Z, et al. DDoS attack detection and defense based on hybrid deep learning model in SDN[J]. Journal on Communications, 2018, 39(7): 176-187.
- [12] BRAGA R, MOTA E, PASSITO A. Lightweight DDoS flooding attack detection using NOX/OpenFlow[C]//Proceedings of IEEE Local Computer Network Conference. Piscataway: IEEE Press, 2010: 408-415.
- [13] KANDOI R, ANTIKAINEN M. Denial-of-service attacks in OpenFlow SDN networks[C]//Proceedings of 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). 2015: 1322-1326.
- [14] 乔思伟, 胡成臣, 李昊, 等. OpenFlow 交换机流表溢出问题的缓解机制[J]. 计算机学报, 2018, 41(9): 2003-2015.  
QIAO S Y, HU C C, LI H, et al. A mechanism of taming the flow table overflow in OpenFlow switch[J]. Chinese Journal of Computers, 2018, 41(9): 2003-2015.
- [15] 武泽慧, 魏强, 任开磊, 等. 基于 OpenFlow 交换机洗牌的 DDoS 攻击动态防御方法[J]. 电子与信息学报, 2017, 39(2): 397-404.  
WU Z H, WEI Q, REN K L, et al. Dynamic defense for DDoS at-

- tack using open flow-based switch shuffling approach[J]. Journal of Electronics & Information Technology, 2017, 39(2): 397-404.
- [16] WEN X T, CHEN Y, HU C C, et al. Towards a secure controller platform for OpenFlow applications[C]//Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. 2013: 171-172.
- [17] KLAEDTKE F, KARAME G O, BIFULCO R, et al. Access control for SDN controllers[C]//Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. 2014: 219-220.
- [18] 崔乾. SDN 北向资源访问安全方案研究与实现[D]. 北京: 北京邮电大学, 2019.
- CUI Q. Research and implementation of SDN northbound resource security access scheme[D]. Beijing: Beijing University of Posts and Telecommunications, 2019.
- [19] 王阿林. 基于 REST 开放标准的北向接口动态 API 研究与实现[D]. 北京: 北京邮电大学, 2017.
- WANG A L. Research and implementation of dynamic API based on REST northbound interface[D]. Beijing: Beijing University of Posts and Telecommunications, 2017.
- [20] 任开磊. 软件定义网络北向 REST 接口安全防护关键技术研究[D]. 郑州: 信息工程大学, 2017.
- REN K L. Research on key technologies of SDN northbound REST API security defense[D]. Zhengzhou: Information Engineering University, 2017.
- [21] 成静, 薛峰, 张逸飞, 等. 移动应用众包测试人员信誉度的模糊评估方法研究[J]. 西北工业大学学报, 2018, 36(4): 800-806.
- CHENG J, XUE F, ZHANG Y F, et al. A reputation assessment approach based on fuzzy mathematics mobile application crowd-sourced testers[J]. Journal of Northwestern Polytechnical University, 2018, 36(4): 800-806.
- [22] 沈丛麒, 陈双喜, 吴春明, 等. 基于信誉度与相异度的自适应拟态控制器研究[J]. 通信学报, 2018, 39(S2): 173-180.
- SHEN C Q, CHEN S X, WU C M, et al. Adaptive mimic defensive controller framework based on reputation and dissimilarity[J]. Journal on Communications, 2018, 39(S2): 173-180.
- [23] 李婕, 王兴伟, 刘睿. 社群智能系统中基于用户信誉度的激励机制[J]. 计算机科学与探索, 2015, 9(12): 1471-1482.
- LI J, WANG X W, LIU R. User reputation- based participatory incentive mechanism in social and community intelligence systems[J]. Journal of Frontiers of Computer Science and Technology, 2015, 9(12): 1471-1482.
- [24] 于洋, 王之梁, 毕军, 等. 软件定义网络中北向接口语言综述[J]. 软件学报, 2016, 27(4): 993-1008.
- YU Y, WANG Z L, BI J, et al. Survey on the languages in the northbound interface of software defined networking[J]. Journal of Software, 2016, 27(4): 993-1008.
- [25] 鲁垚光, 王兴伟, 李福亮, 等. 软件定义网络中的动态负载均衡与节能机制[J]. 计算机学报, 2020, 43(10): 1969-1982.
- LU Y G, WANG X W, LI F L, et al. Dynamic load balancing and energy saving mechanism in software defined networking[J]. Chinese Journal of Computers, 2020, 43(10): 1969-1982.
- [26] 胡涛, 张建辉, 马腾, 等. SDN 中基于可靠性评估的多控制器均衡部署策略[J]. 通信学报, 2017, 38(11): 188-198.
- HU T, ZHANG J H, MA T, et al. Multi-controller balancing deployment strategy based on reliability evaluation in SDN[J]. Journal on Communications, 2017, 38(11): 188-198.
- [27] JAFARIAN J H, AL-SHAER E, DUAN Q. An effective address mutation approach for disrupting reconnaissance attacks[J]. IEEE Transactions on Information Forensics and Security, 2015, 10(12): 2562-2577.
- [28] OKTIAN Y E, LEE S, LEE H, et al. Secure your northbound SDN API[C]//Proceedings of 2015 Seventh International Conference on Ubiquitous and Future Networks. 2015: 919-920.

#### [作者简介]



王洋(1985-), 女, 陕西西安人, 信息工程大学博士生, 主要研究方向为网络与信息安全。



汤光明(1963-), 女, 湖南常德人, 信息工程大学教授、博士生导师, 主要研究方向为网络与信息安全、信息安全管理、信息隐藏。



王硕(1991-), 男, 河南南阳人, 中国西安卫星测控中心工程师, 主要研究方向为网络与信息安全、机器学习。



楚江(1980-), 男, 江苏南京人, 中国西安卫星测控中心高级工程师, 主要研究方向为网络与信息安全、密码应用安全。