doi: 10.3969/j.issn.1001-0505.2017.S1.001

针对SDN基础设施的拒绝服务攻击防护框架

张梦豪[1,2,3] 毕 军[1,3] 白家松[1,2,3] 王旸旸[1,3]

([1]清华大学网络科学与网络空间研究院,北京 100084)
([2]清华大学计算机科学与技术系,北京 100084)
([3]清华信息科学与技术国家实验室,北京 100084)

摘要: 为了进一步缓解针对SDN基础设施的典型拒绝服务攻击(数据-控制平面饱和攻击),提出了一种控制器和交换机协作式的安全防护框架,即FloodShield. 在该攻击中,攻击者通过洪泛伪造数据包,使数据平面产生大量表项缺失和packet-in数据包,从而消耗SDN基础设施不同层次的资源:数据平面的TCAM资源、控制通道的带宽资源和控制器的CPU资源等. 通过对这种攻击的详尽分析,提出并设计了易部署、全面性和轻量化的FloodShield防护框架. 该框架主要使用了2个关键技术:1) 源地址验证,用于在数据平面过滤源地址伪造的数据包;2) 有状态数据包监控,用于监控源地址真实流量的状态,并基于流量评价打分和网络资源使用量采用动态的防护措施. 实验结果表明,相比于之前的防护框架,FloodShield在消耗更少系统资源的同时,对SDN架构的数据平面、控制通道和控制平面都提供了有效的保护.

关键词: 软件定义网络;拒绝服务攻击;源地址验证;有状态数据包监控
中图分类号: TP393.0 **文献标志码:** A **文章编号:** 1001-0505(2017)S1-0001-08

DoS attack defense framework for SDN infrastructure

Zhang Menghao[1,2,3] Bi Jun[1,3] Bai Jiasong[1,2,3] Wang Yangyang[1,3]

([1]Institute for the Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China)
([2]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
([3]Tsinghua National Laboratory for Information Science and Technology (TNLIST), Beijing 100084, China)

Abstract: To further mitigate the dedicated denial-of-service (DoS) attack (data-to-control plane saturation attack) against SDN infrastructure, a controller and switch cooperative defense framework, that is, FloodShield, is presented. In this attack, a large number of packets are flooded by attacker to trigger massive table-misses and packet-in messages in the data plane, which would exhaust resources of different components of SDN infrastructure, including TCAM in the data plane, bandwidth of the control channel, and CPU cycles of the controller. With the extensive analysis of the vulnerability of SDN against the saturation attack, a deployable, comprehensive and lightweight SDN defense framework, FloodShield, is proposed and designed. The following two techniques are combined with FloodShield: 1) source address validation for filtering forged packets directly in the data plane, and 2) stateful packet supervision for monitoring traffic states of real addresses and performing dynamic countermeasures based on evaluation scores and network resource usage. Experimental results show that, compared with previous defense framework, FloodShield provides an effective protection for data plane, control channel and control plane of SDN infrastructure, meanwhile consuming less resource.

Key words: software-defined networking; denial-of-service attack; source address validation; stateful packet supervision

---

收稿日期: 2017-06-19. **作者简介:** 张梦豪(1994—),男,博士生;毕军(联系人),男,博士,教授,博士生导师,junbi@tsinghua.edu.cn.
基金项目: 国家重点研发计划"网络空间安全"重点专项资助项目(2017YFB0801701)、赛尔网络下一代互联网技术创新资助项目(NGII20160123)、国家自然科学基金资助项目(61472213).
引文格式: 张梦豪,毕军,白家松,等. 针对拒绝服务攻击的SDN基础设施安全防护框架[J]. 东南大学学报:自然科学版,2017,47(S1):1-8. [doi: 10.3969/j.issn.1001-0505.2017.S1.001]

软件定义网络(software-defined networking,SDN)是近年来一种新兴的网络架构,其通过细粒度、控制转发分离和集中控制来简化网络管理和优化.典型的SDN架构包括三个主要组件:控制平面、数据平面和允许两个平面以标准协议进行通信的控制通道.

OpenFlow协议将SDN思想标准化,基于"匹配-动作"机制,OpenFlow引入了数据包被动(reactive)处理机制:数据包进入OpenFlow交换机后,首先在本地流表中进行查询和匹配.如果没有流表项可以匹配到该数据包,则会产生一次表项缺失(table-miss),交换机将此数据包的包头(或者数据包本身)封装在一个packet-in报文中,并转发给控制器.控制器收到数据包后,会计算得出对应的流规则,并通过下发flow-mod消息将新规则安装到相应的交换机上.

数据包被动处理机制使得SDN可以迅速适应网络的动态变化,但同时该机制也可能会成为SDN架构新的脆弱点.攻击者可以利用上述缺陷来进行数据-控制平面饱和攻击[1-4]——一种针对SDN架构的拒绝服务攻击(denial-of-service attack,DoS Attack).通过控制若干僵尸主机,攻击者可以产生大量数据包头伪造的恶意数据包,这些数据包往往不会在交换机本地流表中得到匹配,因此会触发大量的表项缺失,从而产生大量的packet-in数据包.处理这些packet-in数据包会消耗数据平面、控制平面及控制通道的可用资源,进而造成SDN整体架构的瘫痪.

针对这种攻击,已有一些工作提出过相应的防御措施.Avant-Guard[2]通过修改交换机设计来缓解TCP SYN洪泛攻击.FloodGuard[3]通过引入数据平面缓存来保护控制器.FloodDefender[4]采取了三种技术,在控制器中实现了四个模块来缓解这种针对SDN的DoS攻击.然而,根据作者的调查,还没有一个工作能同时满足以下3点要求:

(1)全面保护SDN架构:Avant-Guard和FloodGuard都主要着眼于SDN控制器的保护,忽视了对数据平面和控制通道的保护.然而,即使控制器得到很好的保护,如果数据平面和控制通道资源被攻击所消耗,攻击仍然可以发挥作用.此外,由于交换机中三元内容寻址存储(ternary content addressable memory,TCAM)资源较为昂贵且能耗较高[5-6],保护交换机资源不受饱和攻击和溢出攻击[7]的影响对服务质量和节能有重要意义.

(2)易于部署:由于SDN已经在许多企业/校园/服务提供商处得到部署[8-9],在不更改现有SDN架构、不引入新的设备前提下完成对攻击的防御变得十分重要,也更容易控制整体代价.Avant-Guard和FloodGuard都没有满足此项要求,这两项工作都引入了额外设备来防御攻击.

(3)轻量化,不带来过多控制器开销:作为SDN架构的核心组件,控制器在很多网络功能中都起着关键性的作用,因此也承担着较大的负荷.FloodDefender将防御功能全部放在控制器,由控制器上的模块处理所有消息,这样不可避免地会使控制器负荷过大,从而损害保护的效果.因此防御系统要满足轻量化的要求,对控制器不带来过多开销.

针对上述要求,本文提出了FloodShield框架——一个针对数据-控制平面饱和攻击的SDN防御框架.结合源地址验证和有状态数据包监控两项技术,FloodShield对SDN三层资源提供了全面的保护.同时,FloodShield不会修改现有OpenFlow交换机,也不引入新的设备,因此易于部署在已有OpenFlow网络中.此外,系统通过数据平面和控制平面协调工作来完成对攻击的防御,不会对控制器引入过多的额外负荷,满足轻量化的要求.

1 研究问题与敌手模型

这里用一个简化的网络场景来描述SDN架构被攻击的过程.当一个没有相应匹配流表项的数据包到达交换机时,交换机会将该数据包的内容存储在交换机缓存区,并向控制器发送packet-in数据包.如果缓存区未满,packet-in消息只包含数据包头;如果缓存区已满,则包含整个数据包的内容.控制器接收到该消息后,会计算对应的路径,并将相应的规则以flow-mod和packet-out数据包的形式发送到交换机.控制消息到达交换机后,交换机解析消息并将流表项安装在流表中.攻击者可以利用上述被动处理机制,通过大量伪造数据包来消耗系统资源.这些伪造数据包的包头都用蓄意伪造的数值填充,几乎不会在已有表项中得到匹配.由此交换机产生大量表项缺失,大量packet-in数据包被发送至控制器,使得整个SDN系统资源被消耗.

上述敌手模型中,SDN的三层资源受到损害.

(1)OpenFlow交换机:一方面,过多恶意流量会使交换机的缓存区域被占满.另一方面,即使控制器有足够的吞吐量能力去处理所有的packet-in数据包,大量流表表项仍然会超出交换机流表的容量,因此会有一部分流表表项被丢弃.在Open-

Flow1.4 及之后的版本[10] 中，交换机可以移除已有表项来为新下发的流表规则提供存储空间，但这也使得攻击者可以进行流表溢出攻击[7]. 利用交换机表项替换机制的缺陷，恶意流量产生的无效表项可以侵略性的替换掉已安装的正常表项，从而导致正常流量出现更多的表项缺失.

(2) 控制通道：根据 packet-in 的触发机制，当交换机的缓存区域存满时，发送到控制器的消息会包含整个数据包的内容，从而放大了攻击流量. 同时，控制器接收到这些 packet-in 消息后，会下发例如 flow-mod 和 packet-out 控制消息，从而使得控制通道更加拥塞.

(3) 控制器：控制器接收到 packet-in 消息后必须进行一定操作，包括计算路由路径、封装路径信息到控制消息中以及下发控制消息到数据平面. 因此，恶意流量导致的 packet-in 消息会消耗大量的控制器计算和存储资源.

2　系统设计

2.1　系统架构

FloodShield 可以直接在现有 SDN 网络操作系统(如 Floodlight，OpenDaylight，ONOS 和 RYU 等)的基础上实现，系统向控制器平台引入了源地址验证和有状态数据包监控两个模块. 源地址验证模块是防御数据-控制平面饱和攻击的第一道屏障，模块对入口处流量的来源进行验证，并在数据平面过滤掉伪造数据包. 在此基础上，有状态数据包监控模块以不同源地址为单位，分别对相应流量进行监测，并根据流量行为和网络资源使用量来对流量采取不同的抑制措施. 为了使两个模块更好地在不同网络环境下工作，所有的模块参数均可以以 REST API 的形式来外部定义.

如图 1 所示，源地址验证模块在一个主机接入 SDN 网络时工作，模块基于网络配置来监听地址分

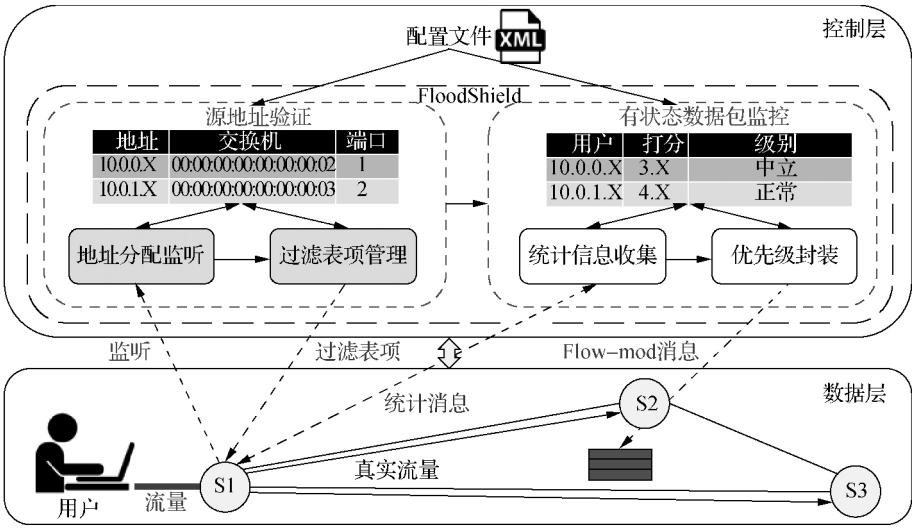


图 1　总体架构图

配过程，并向入口处的交换机下发过滤表项. 该模块使得伪造数据包在边缘交换机的入口端口处被过滤，使得基于源地址的流量统计和追责变得可行. 有状态数据包监控模块从边缘交换机处收集流量数据，并定期更新每个用户的评价分数. 根据该评价打分，系统会对不同用户采取不同措施，来实现动态化的网络区分服务.

2.2　源地址验证模块

为了掩盖意图和自我保护，攻击者往往倾向于使用伪造的源地址来实施攻击. 然而，由于僵尸主机的地址是由网络预先进行分配的，并且是事先确定的，尤其是在基于 SDN 的数据中心或校园网这类强管控的网络环境中. 因此，本文将源地址验证引入了对饱和攻击的防御方案中，并将其作为一个控制器中的一个基础模块. 然而，实施源地址验证的位置和方法还未确定. 为了解决这个问题，作者利用现有流表的特点，借助控制器的智能管控和数据平面的线速处理，提出了一种有效的协调机制，直接在数据平面中过滤伪造的数据包. 以上设计可以直接用于支持 OpenFlow 协议的交换机，不需要对协议进行修改，也不需要引入额外的设备(如专用的 SAVI 交换机).

2.2.1　部署源地址验证的位置

理论上，可能需要在 SDN 的不同层次部署源地址验证机制，但实际上，只要将源地址验证机制部署在与终端主机或服务器相连的交换机的端口

处就可以保证对数据包的验证. 为方便论述,首先提出以下两条假设:

假设1(数据包生成) SDN数据平面中除控制消息外的所有数据包都来自与边缘交换机相连的主机或服务器.

假设2(数据包传递) 在SDN数据平面,交换机的作用就是将数据包从一个端口转发到另一个端口(包含修改数据包的动作)或直接丢弃数据包.

数据平面中不同交换机与不同的设备相连,本文将交换机的端口分为以下3类. 交换机端口:指与其他交换机相连的端口;主机端口:指与终端主机或服务器相连的端口;可信端口:指与DHCP服务器等可信赖的服务器相连的端口.

基于以上假设和端口分类,可以推论出网络中除控制消息外的所有流量都是由主机端口或可信端口产生,并由交换机端口转发. 来自可信端口的流量源地址真实,而来自主机端口的流量源地址未必真实. 因此只需要在主机端口部署源地址验证机制,就能保证数据平面的流量均来源真实.

2.2.2 实施源地址验证的方法

如上节所述,在主机端口处验证即可保证进入网络流量的来源真实,在控制器中维护了一个全局的绑定表,每个绑定表项都将一个源地址和一个交换机端口绑定,记录为<源地址,交换机,端口>的形式,即来自该源地址的数据包只能由对应交换机的对应端口进入网络,否则就被视为伪造的数据包.

主机要进入SDN网络需要先分配到地址,然而,真实场景下会存在多种地址分配协议(静态/动态),如何在复杂的网络环境中获取地址分配信息并建立绑定关系成为面临的第一个挑战. 基于前人的SDN-SAVA[11]工作,设计了可以接收外部配置文件及相应参数的接口. 配置文件以XML的格式存在,指定了具体的地址分配协议,以及建立绑定表所需的其他参数信息.

(1) 对于动态地址分配协议(如DHCP协议),如图2所示,为了监听地址分配(address assignment mechanism, AAM)过程,在交换机连入网络时,控制器会在其主机端口和可信端口安装监听规则. 之后所有的AAM数据包会以packet-in消息的形式发送至控制器,控制器可以监控AAM的整个过程. 在控制器上对每个地址分配协议维持一个状态机(比如,RFC 7513就描述了DHCP的源地址验证状态机),每个IP地址都有一个状态机,记录当前地址分配阶段,状态由新来的AAM数据报文或计时器超时等事件来触发. 当状态转变为BIND时,该地址的绑定项会被加入到绑定表中,而当状态变为NO_BIND时,该地址对应绑定项从表中移除. 需要考虑的是,在攻击者发送伪造AAM数据包时,以上机制可能会成为系统新的脆弱点. 对此,使用交换机的meter表来限制AAM数据包的发送速率,同时在DHCP服务器上也限制AAM数据包的发送速率,因此正常环境下AAM流量速率都会较低.

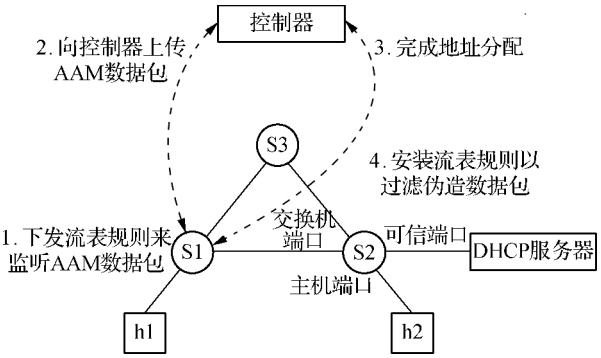


图2 动态地址分配的SAVI

(2) 对静态地址分配协议,系统通过直接读取网络配置来建立绑定表.

在绑定表的基础上,如何实施源地址验证机制成为了第二个挑战. 实施源地址验证的最简单的方式是在控制器上,即对于每一个新流,控制器都检查该流的第一个报文. 对于源地址伪造的报文,具体的处理方法有两种. (1) 忽略伪造包,该方案的缺陷是攻击者可以重复发送同一伪造数据包,以此来消耗控制器的计算资源和控制通道的带宽资源;(2) 向交换机安装对应该流的丢弃规则,这样就把检查和过滤的任务卸载到了数据平面,保护了控制器的资源. 然而,只要攻击者不断尝试随机的源地址,就会不断的产生不同的新流,仍然会有效的消耗控制器的资源;不仅如此,由于这种方式要求控制器下发丢弃规则,控制器上的资源消耗更大,交换机上的资源消耗也很大. 为解决以上问题,源地址验证模块向交换机显式地安装过滤表项,直接过滤掉未被绑定的源地址的数据包. 这种方式下,伪造的报文就可以在不消耗更多交换机或控制器资源的情况下,仅在数据平面就可以完成对报文源地址的验证和对伪造报文的丢弃.

现有的OpenFlow交换机只支持“匹配-动作”的形式,即如果数据包在流表中有匹配表项的时候,交换机会根据表项指定的指令来处理该数据包. 然而,源地址验证中的过滤表项遵循的却是

“不匹配-动作”的形式，只有数据包不匹配已有绑定关系时，才会执行对应的丢弃动作. 如何设计可以表达“不匹配-动作”逻辑的流表规则成为第 3 个挑战. 为了防止出现流表爆炸问题，同时保证流表的分离性和独立性，利用了 OpenFlow 的多级流表机制，过滤表项安装在第 0 级流表，而其他应用程序的表项都安装在之后的流表中. 受 Maple[12] 中算法策略的启发，使用优先级搭配的通配规则来实现过滤，由于源地址验证只部署在主机端口处，因此来自交换机端口和可信赖端口的数据包会被直接转发到后续流表中. 具体过滤表项包括 6 组，如表 1 所示. 第 1 和第 2 组使所有给定 IP 地址和 MAC 地址的数据包转发到之后流表; 之后两组流表使得所有伪造数据包被丢弃; 最后两组流表使得来自交换机端口和可信赖端口的数据包直接转发到之后流表.

表 1　数据平面第 0 级(table 0) 流表(过滤流表) 结构

组名	匹配域	优先级	动作
主机端口	in port = hport, mac src = given mac src, ipv4, ipsrc = given ip src	2	跳转到 1 级流表
主机端口	in port = hport, mac src = given mac src, arp	2	跳转到 1 级流表
主机端口的过滤表项	in port = hport, ipv4	1	丢弃
主机端口的过滤表项	in port = hport, arp	1	丢弃
交换机端口	in port = sport	1	跳转到 1 级流表
交换机端口	in port = tport	1	跳转到 1 级流表

2.3　有状态数据包监控模块

尽管源地址验证模块可以过滤掉伪造的数据包，攻击者仍然可以通过源地址真实的数据包来执行攻击. 因此，引入了有状态数据包监控来进行基于源地址的统计和追责. 来自不同源地址的数据流会根据流量特征被分为不同等级，系统有差别地对这些用户提供服务. 系统首先根据采集的流量特征对源地址(用户) 进行评价，评价结果按公式被量化为分数，正常的流量特征会使分数增加，而恶意的特征会使分数降低. 基于这些评价分数和网络资源使用量，系统动态的对不同数据流采用不同的措施. 正常用户的数据流会得到更好的服务，而被怀疑是攻击者的用户会得到较差的服务，甚至直接被丢弃. 有状态数据包监控的思想很朴素，但如何实施仍然有两个主要挑战. 1) 如何设计基于流量行为的评价准则以区分用户服务; 2) 如何对不同流动态提供有差别的服务. 针对这两点，设计了以下两个部分.

2.3.1　行为评价

为了进行基于流量行为的用户评价，首先需要确定要收集哪些流量特征以及如何收集. 正常流量和攻击流量两个基本区别在于新流的频率和每个流数据包的个数[4]. 为了降低攻击的代价，攻击者倾向于用短流(单个流里面数据包较少) 来攻击，因为这样可以用较低的代价来完成更强的攻击. 正常情况下，每个新流都会触发一个 packet-in 消息到控制器，因此控制器可以获知新流的频率信息. 然而，控制器无法获知每个流的数据包数量，统计查询消息必须由控制器发送到交换机上(查询间隔为 T s) 来获取相应流表的快照，读取各流表项的 counter 字段. 数据统计服务在多数控制器平台上都集成在网络操作系统里作为基础服务，因此可以调用该服务并根据源地址对流表项数据进行分类统计.

在得到这些流量特征数据后，需要设计评价标准来量化不同用户的行为. 如之前所述，新流的频率和每个流的数据包个数是正常流量和攻击流量的两个显著区分点. 考虑过去 T s，将用户 i 的 packet-in 消息的个数记为 pi_i，用户 i 所有流表项中正常流表项所占的比率记为 cr_i，即

$$cr_i = \frac{\text{flow entries}(\text{counter} > t \,\&\, \text{源地址} = i)}{\text{flow entries}(\text{源地址} = i)}$$

式中，t 为常数参数，用于区分流表项是否正常(在实验中通常设为 1 或 2) . 由 pi_i 和 cr_i 两项数据结合得出分数 w_{in}，w_{in} 与 pi_i 负相关而与 cr_i 正相关，为保证两项数据对评价得分具有相同的影响，实验中将两项数据线性映射为相同定义域的离散化分数，然后相加得到 w_{in}. 考虑到评价一个用户还需要考量历史行为，采用了指数加权移动平均方法(EWMA) 来得出最终对用户 i 的评价分数 w_i，$w_i = (1-\alpha)\, w_i + \alpha w_{in}$，这里 α 是一个常数，在 0 ~ 1 之间取值.

2.3.2　网络服务区分

基于评价打分，系统可以对不同的用户采取不同的策略，来实现有差别的服务. 本文用两个阈值 (th_l，th_h) 根据打分将用户分为三个等级(恶意、中立和正常) ，阈值可以由网络管理员根据网络策略进行配置. 得分 w_i 低于 th_l 的用户视为恶意，得分 w_i 超过 th_h 的用户视为正常，而得分在两个阈值之间的用户看作中立，即不能确认该用户是正常用户

还是恶意用户.

控制器接收到来自恶意用户的新流时,控制器会直接向控制器下发丢弃或限速的流表规则到入口交换机,在接下来一段时间内丢弃或限速来自该用户的数据包. 本文之所以采取这样比较直接的措施,是因为 th_i 阈值设置较低,正常用户的流量行为基本不会被判为恶意.

正常用户遵循相似的规律,因为 th_h 阈值很难由恶意流量达到,因此系统会正常处理来自这些用户的流量. 考虑攻击者可能会进行流表溢出攻击[7],文中对不同的 flow-mod 消息赋予不同的 importance 值,用来保护数据平面中正常用户的流表. 来自中立用户的新流设置较低的 importance 值,而来自正常用户的新流设置较高的 importance 值,这样正常用户的已安装表项就不会被恶意表项所替换.

对于中立用户,基于控制器的资源使用情况,采取概率选择的方法来处理 packet-in 消息. 当控制器较忙时,概率值设置较低,只有少部分来自中立用户的 packet-in 消息会被处理;相反,在控制器相对空闲时,大部分 packet-in 消息都会被处理. 假定正常条件下 SDN 控制器每秒可以处理 N 个 packet-in 消息,当前控制器每秒收到 N_R 个 packet-in 消息,那么使用 N_R/N 来代表网络资源使用量. 概率值的选择应和该值成负相关关系,实验采用了简单的线性关系来实现. 上述策略汇总如表 2 所示.

表 2 基于行为的网络服务区分策略

评价分数	对应措施	Importance 值
恶意	在一段时间内丢弃或限速所有数据包	无/最低
中立	一定概率选取数据包处理	低
正常	处理所有数据包	高

3 系统实现与实验评价

3.1 系统实现与实验环境配置

为验证 FloodShield 的效果,本文在 Floodlight 控制器基础上实现了系统的原型,将源地址验证模块和有状态数据包监控模块集成在控制平台上. 所有实现都遵循 OpenFlow 协议,也不需要引入额外的设备,因此也很容易迁移到其他的 SDN 控制器平台上. 实验使用 Mininet 和 Open vSwitch 来模拟 SDN 网络.

基于 Floodlight 和 Mininet,本文建立了扇出为 2,深度为 3 的树状拓扑(见图 3),控制平面和数据平面分别运行在两台配置相同服务器上,这两台主机由传统的二层硬件交换机相连. 服务器硬件参数为双核 Xeon-2620 CPU,64 GB RAM 和 1Gigabit NIC. 为了更好地模拟真实场景,将拓扑的带宽设为 100 Mbps,同时将每个交换机的每张流表大小都设为 2 000,使用 OpenFlow 1.4 版本协议来进行南向通信.

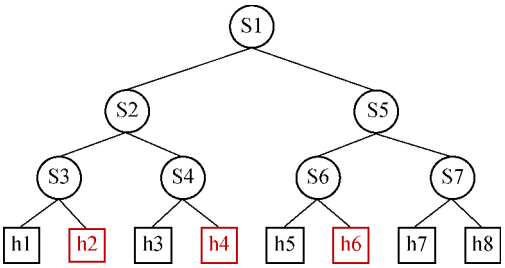


图 3 实验拓扑

为了与之前工作进行对比,本文分别使用以下四个系统进行了实验:(1) 没有保护系统的原始 OpenFlow 网络(of);(2) 部署 FloodGuard[3] 的 OpenFlow 网络(fg);(3) 部署 FloodDefender[4] 的 OpenFlow 网络(fd);(4) 部署 FloodShield 的 OpenFlow 网络(fs). 本节中,文中使用缩写 of,fg,fd,fs 来代表以上四个系统.

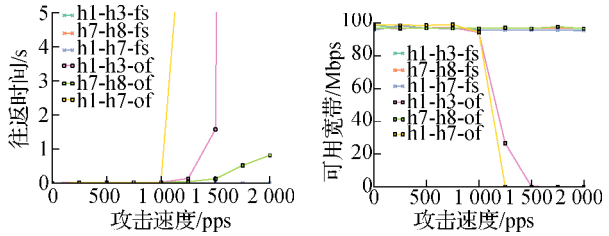
3.2 端到端防护效果

为评价 FloodShield 系统对 SDN 框架的总体防护效果,文中使用端到端的主机性能作为指标,测量了不同防护系统下其往返时间(round-trip time,RTT)和可用带宽.

本文使用 h2,h4 和 h6 三台主机作为攻击者,使用 pktgen 工具发送 UDP 数据包来发起洪泛攻击,其中真实源地址和伪造源地址的流量比例为 3/1. 同时使用 h7 访问 h8,h1 访问 h3 和 h1 访问 h7,用 ping 工具来测量 RTT 值,用 iperf 工具来测量可用带宽值. 其中,h7—h8 代表只经过边缘层交换机的通信过程,h1—h3 代表经过边缘层交换机和聚合层交换机的通信过程,而 h1—h7 代表三层交换机都经过的通信过程. 由于 FloodGuard,FloodDefender 和 FloodShield 在 RTT 和可用带宽上具有基本相同的表现,这里只展示 FloodShield 系统与原始 OpenFlow 系统的对比测试结果,说明 FloodShield 系统对端到端整体通信的保护效果.

如图 4(a) 所示,在原始 OpenFlow 系统中,随着攻击速率的增加,当攻击速率大于 1 000 数据包每秒(packets per second,pps) 时,三对主机的 RTT 开始急剧增大,而在 FloodShield 系统中,RTT 值基本保持不变. 如图 4(b) 所示,可用带宽的测量结果

和 RTT 相似，攻击速率大于 1 000 pps 后，原始系统下三对主机间的可用带宽都有一定减少，h1—h7，h1—h3 之间的可用带宽更是接近被全部占用，而 FloodShield 下可用带宽基本不变. 这是因为一般情况下 Open vSwitch 控制通道处理能力大概为 1 200 pps. 此外，攻击速率为 0 时的数据说明，在没有攻击时，FloodShield 系统下 RTT 和可用带宽值和原始系统基本相同，没有带来额外通信负担. 实验结果说明了 FloodShield 框架在受到饱和攻击时，对 SDN 架构下端到端通信有较好的保护效果.



(a) 往返时间　　(b) 可用带宽

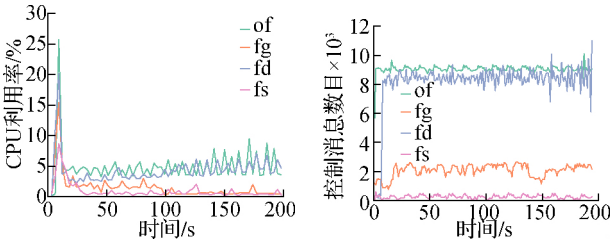
图 4　端到端防护效果

3.3　资源使用开销

饱和攻击会消耗 SDN 架构的资源，包括控制器的计算资源、控制通道带宽和数据平面的流表资源. 为验证 FloodShield 对分层资源的保护能力，同样与其他三个系统进行了实验比对.

本部分采取与之前同样的拓扑，同样使用 h2、h4 和 h6 三个主机，用 100—1 000 pps 的速率来进行 UDP 洪泛攻击，真实源地址和伪造源地址流量比仍为 3/1. 同时为了模拟网络中的正常流量，从清华大学校园网络中采集了部分流量，然后在其他主机上进行重放. 在进行攻击时，本文记录了控制通道里控制消息的数量和控制器的 CPU 使用率，同时记录了正常用户和攻击者所占的流表项个数，用于衡量流表的利用率.

控制器 CPU 使用率: 不同系统在攻击下的控制器 CPU 利用率如图 5(a) 所示，相比原始 OpenFlow 系统和 FloodDefender 系统，FloodShield 占用了更少的 CPU 资源. 因为 FloodGuard 采用了数据平面缓存的技巧，对控制平面能起到一定的保护作用，所以其和 FloodShield 达到了相似的保护效果. 而 FloodDefender 系统因为将更多任务交由控制器处理，使得控制器负荷较大，实验结果也支持了这个论点. FloodShield 在数据平面过滤掉了源地址伪造的流量以及部分恶意流量，因此控制器的负荷最小.



(a) 控制器 CPU 使用率　　(b) 控制带宽使用情况

图 5　资源使用率

控制通道使用情况: 本文将控制消息的范围限制在 packet-in，packet-out，flow-mod 和 flow-removed. 图 5(b) 显示了四个系统的控制消息数量. 由于 FloodGuard 的主动(proactive) 流表下发策略及数据平面缓存的速率控制策略，因此控制通道使用率较低; FloodDefender 减少了恶意流量导致的 flow-mod 消息个数，但同时刷新流表又需要下发新的 flow-mod 消息，因此控制通道利用率和原始系统相似; 由于 FloodShield 在数据平面丢弃了部分恶意流量，因此控制消息个数最少，对控制通道的保护效果也最好.

流表利用率: 流表利用率的结果如表 3 所示，与 OpenFlow(98%)、FloodGuard(45%) 和 FloodDefender(66%) 相比，FloodShield 占据了更少的流表表项(39%)，对正常用户提供了更好的服务，同时有效抑制了攻击用户所占的表项数量. FloodGuard 使用速率控制来保护控制器和交换机，因此正常用户和攻击用户所占总体表项相比原始 OpenFlow 系统都有下降(45%/98%); FloodDefender 使用两阶段过滤，并且定期刷新交换机的流表，因此，正常用户和恶意用户所占表项也有下降(38%)，然而 FloodDefender 系统会在交换机引入大量额外的监控流表表项(28%); FloodShield 更全面的保护了流表，一方面，FloodShield 在数据平面都过滤了一部分恶意流量，另一方面，FloodShield 系统中正常用户的流表表项也不会被恶意表项代替，因此 FloodShield 对流表表项的保护效果最好.

表 3　平均占用流表项个数

系统	正常用户	攻击者	总计
OpenFlow	535.30(27%)	1 423.14(71%)	1 959.21(98%)
FloodGuard	389.77(19%)	508.38(25%)	900.77(45%)
FloodDefender	492.09(25%)	250.37(13%)	1 313.11(66%)
FloodShield	679.17(34%)	97.70(5%)	799.87(39%)

3.4　攻击检测分析

本文对有状态数据包监控做进一步的分析，测试是否能正确监测攻击者，实验环境与 3.3 节类

似.在表4中,正常用户指h1和h3,攻击者指h2和h4.正常用户的packet-in消息有83%被判定为正常并且正确处理,17%的消息被判定为中立,没有被判定为恶意的情况.而攻击者流量导致的packet-in消息有58%被判定为恶意并且被直接丢弃,42%被判定为中立,因为攻击者被检测出后,在一段时期内所有的数据包都会被丢弃,这段时期内该用户的得分会慢慢上升,因此用户评价会变为中立,在继续发送一部分流量后又会被系统判定为恶意.

表4 攻击检测分析 %

用户	恶意	中立	正常
正常用户	0	17	83
攻击者	58	42	0

以上实验结果表明,FloodShield系统可以较好地区分正常用户和恶意用户,基于行为的评价打分标准,没有严重的误判出现.

4 结语

本文提出了FloodShield,一种针对数据-控制平面饱和攻击的易部署、全面、轻量级的SDN防御框架.基于对SDN在饱和攻击下脆弱点的分析,设计了FloodShield的两个组件:源地址验证和有状态数据包监控.实验表明FloodShield可以提供对SDN数据平面、控制通道和控制平面的全面保护,同时只带来极低的额外代价,不会影响网络系统的正常效率.

在未来工作中,作者将尝试解决如何将源地址验证机制部署在SDN与传统网络相连的网关处,这个场景比纯SDN网络要更加复杂,也具有更实际的意义.此外,还将尝试在实际环境中部署的方案,这将使作者对问题的认识更加深入.

参考文献(References)

[1] Shin S, Gu G. Attacking software-defined networks: a first feasibility study [C]//HotSDN '13 *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. Hong Kong, China, 2013: 165-166.

[2] Shin S, Yegneswaran V, Porras P, et al. Avant-guard: scalable and vigilant switch flow management in software-defined networks [C]//*Proceedings of the* 2013 *ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013: 413-424.

[3] Wang H, Xu L, Gu G. Floodguard: a dos attack prevention extension in software-defined networks [C]//45*th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (*DSN*). Rio de Janeiro, Brazil, 2015: 239-250.

[4] Gao S, Peng Z, Xiao B, et al. FloodDefender: protecting data and control plane resources under SDN-aimed DoS attacks [C]//*INFOCOM* 2017-*IEEE Conference on Computer Communications*. IEEE. 2017: 1-9.

[5] Qazi Z A, Tu C C, Chiang L, et al. SIMPLE-fying middlebox policy enforcement using SDN [J]. *ACM SIGCOMM Computer Communication Review*, 2013, 43(4): 27-38. DOI: 10.1145/2534169.2486022.

[6] Katta N, Alipourfard O, Rexford J, et al. Cacheflow: dependency-aware rule-caching for software-defined networks [C]//*Proceedings of the Symposium on SDN Research*. ACM, 2016: 6.

[7] Zhang M, Bi J, Bai J, et al. FTGuard: a priority-aware strategy against the flow table overflow attack in SDN [C]//*Proceedings of the SIGCOMM Posters and Demos*. ACM, 2017: 141-143.

[8] Jain S, Kumar A, Mandal S, et al. B4: experience with a globally-deployed software defined WAN [J]. *ACM SIGCOMM Computer Communication Review*, 2013, **43**(4): 3-14. DOI: 10.1145/2534169.2486019.

[9] Hong C Y, Kandula S, Mahajan R, et al. Achieving high utilization with software-driven WAN [C]//*ACM SIGCOMM Computer Communication Review*. ACM, 2013, 43(4): 15-26.

[10] ONF. Openflow switch specification version 1.5.0 [EB/OL]. (2014-03-26) [2017-08-24]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[11] Liu B, Bi J, Zhou Y. Source address validation in software defined networks [C]//*Proceedings of the* 2016 *ACM SIGCOMM Conference*. Florianopolis, Brazil, 2016: 595-596.

[12] Voellmy A, Wang J, Yang Y R, et al. Maple: Simplifying SDN programming using algorithmic policies [J]. *ACM SIGCOMM Computer Communication Review*, 2013, **43**(4): 87-98. DOI: 10.1145/2534169.2486030.