轻量级的软件定义网络数据包转发验证

王首一 李 琦 张 云
(清华大学深圳研究生院 广东 深圳 518055)
(清华大学计算机科学与技术系 北京 100084)

摘 要 软件定义网络(Software Defined Network,SDN)引入控制层与转发层分离简化了网络管理和功能部署,近年来得到了广泛的关注.然而,SDN 无法检测由于网络攻击或者转发规则的错误实施导致的数据包被错误转发.例如,SDN 中转发的数据包会被异常的规则或攻击者丢弃、篡改或注入虚假数据包.此外,由于处于数据层的 SDN 交换机仅提供了简化的数据转发功能,因此作者无法简单地部署传统 IP 网络中的数据转发验证方案.因此,作者需要提出一个适用于 SDN 的有效数据转发验证方案以确保数据包的正确转发.已有 SDN 转发验证的方案通常通过逐跳验证或者对比全部流的统计信息,这会带来巨大的计算和通信开销.文中基于 OpenFlow 协议提出了一个轻量级的 SDN 数据包转发验证方案 LPV(Lightweight Packet Forwarding Verification).由于 LPV 利用 SDN 本身提供的 Packet-in 消息机制以及组表读取转发结点的流转发统计值,在检测转发异常行为以及定位异常行为结点的同时,避免了大量读取转发结点状态而引入的计算和通信开销.LPV 利用流表规则对入口和出口交换机进行采样,将采样信息的消息验证码 MAC(Message Authentication Code)值和相应的流统计信息上报给控制器.由此,控制器可以通过对比包的 MAC 值和统计信息来检测网络中的异常转发行为.与此同时,LPV 可以通过分析收集的信息找出篡改或丢弃包的结点以定位异常行为的结点.通过基于随机化采样的转发验证机制,LPV 有效降低控制器和交换机中引入的处理和通信开销.同时,随机化采样实现了交换机转发状态的一致性检测,任何攻击者都无法通过推断采样来绕过 LPV 的检测.作者在开源 Floodlight 控制器和 ofsoftware13 软件交换机中实现了 LPV 并在 Mininet 中进行了仿真实验,实验结果表明 LPV 能够检测及定位数据包篡改、流量劫持等转发异常行为,同时仅引入了大约 10%的平均转发延迟和小于 10%的通信开销.

关键词 软件定义网络;轻量级转发检测;一致性检测;异常定位
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2019.00176

LPV: Lightweight Packet Forwarding Verification in SDN

WANG Shou-Yi LI Qi ZHANG Yun
(*Graduate School at Shenzhen, Tsinghua University, Shenzhen, Guangdong* 518055)
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Software-Defined Networking (SDN) simplifies network management by separating control and data planes, and has been received much attention recently. However, SDN cannot ensure correctness of packet forwarding in the networks, e.g., the packets in SDN can be dropped, tampered with, or faked, which may be incurred by false forwarding rule enforcement or attacks. SDN has a simple packet forwarding mechanism in its data plane so that the forwarding verification techniques in the traditional IP networks cannot be applied in SDN. Therefore, it is challenging to verify packet forwarding and ensure correctness of packet forwarding in SDN. The existing studies verify packet forwarding in SDN by verifying packets hop-by-hop or periodically comparing flow statistics of all flows, which incurs significant computation and communication

收稿日期:2016-06-28;在线出版日期:2017-07-07.本课题得到国家重点研发计划专项(2016YFB0800102)和国家自然科学基金(61572278)资助.**王首一**,男,1990 年生,硕士研究生,主要研究方向为软件定义网络、网络安全.E-mail:wang-sy14@mails.tsinghua.edu.cn.
李 琦(通信作者),男,1979 年生,博士,副研究员,中国计算机学会(CCF)高级会员,主要研究方向为网络安全、隐私保护、大数据安全等.E-mail:qi.li@sz.tsinghua.edu.cn.**张 云**,女,1993 年生,硕士研究生,主要研究方向为软件定义网络.

overhead. In this paper, we present LPV (Lightweight Packet Forwarding Verification), a system provides the ability of verifying SDN data plane forwarding. The goal of LPV is to provide a reliable and practical mechanism to detect and defend against wrong packet forwarding. To this end, we develop a lightweight forwarding verification approach to detecting forwarding anomalies and locating malicious switches by leveraging the Packet-in mechanism and the flow statistics maintained in switches. LPV samples packets delivered by ingress and egress switches according to dedicated flow rules, and reports the message authentication code (MAC) values of packets and the statistics of the corresponding flows by Packet-in messages. Thereby, the controllers can detect malicious forwarding behaviors by comparing the MAC values of the packets and the statistics of the flows. Moreover, LPV can locate the switches that perform the malicious packet forwarding behaviors, e. g. , malicious packet modification and packet dropping, by analyzing the correlations of the information, i. e. , the Packet-in messages and flow statistics. By enforcing the sample mechanism, LPV significantly reduces the computation cost, and communication and storage overheads, which incurred by packet processing in switches and controllers. In particular, by randomly sampling packets according to the flow rules, LPV ensures the consistency of packet processing performed by different switches. Adversaries cannot easily infer which packets are sampled so that they cannot interfere with the verification. We implement a prototype of LPV with open source OpenFlow controllers, i. e. , Floodlight, and open source OpenFlow switches, i. e. , ofsoftware13, and evaluate the performance by Mininet experiments. The experimental results show that LPV detects various forwarding anomalies, while introducing negligible overhead, i. e. , around 10% average delays in packet forwarding and less than 10% communication overhead.

Keywords　software-defined networking; packet forwarding anomalies; consistency verification; localization

1　引　言

SDN 技术从提出至今，由于其分离了控制层面和转发层面，对网络提供了良好的抽象，得到了非常广泛的关注. 随着 SDN 功能的逐渐完善，支持的场景逐渐丰富，因此安全问题也逐渐凸显，成为了影响 SDN 发挥强大功能的障碍，特别是由于攻击和配置错误等导致的网络转发异常. 攻击者仅需要通过某一个恶意中间结点篡改或者丢弃数据包来实施攻击. 此外，SDN 中实现灵活的流量工程，或者设置的复杂过滤策略都将失效，可能使得攻击更加隐秘. 特别是由于数据平面与控制平面的分离，导致数据平面的笨拙，因此对数据平面转发的攻击变得更加简单，例如，实施恶意丢包、流量劫持、数据包内容的恶意篡改和向正常网络流中插入虚假数据包等.

由于 SDN 的灵活机制导致了对数据层攻击的防御变得非常复杂[1-7]，目前 SDN 还没有提供比较有效的数据转发验证机制. 虽然 Sphinx[8] 提供了 SDN 数据流转发验证，通过获取数据层转发的统计信息检测数据丢包以及流量劫持这类攻击，但由于其缺乏有效的消息验证功能并不能确保收到的统计信息一定是正确的，也无法保证到达目的的数据包与源端发送的数据包的一致性，因而它无法检测智能的数据层攻击，例如恶意丢弃和重发数据包导致的重放攻击. 同时，目前很多研究提出 IP 网络数据转发验证机制，例如通过提供完整的消息验证功能来实现转发异常检测机制. 然而，这些机制需要逐跳计算校验码并存储密钥，因此需要较大的存储和计算开销. 由于 SDN 交换机仅提供了简单的数据转发功能，因此无法直接应用和部署这些方案.

本文提出了 LPV，一种轻量级的 SDN 数据包转发验证机制，在提供有效的数据转发验证的同时引入了很小的开销. LPV 通过 SDN 的组表实现对网络中转发的数据包进行实施采样并添加标签，由网络中的入口交换机随机将这些采样的数据流的统

计值以 Packet-in 的方式汇报给控制器. 控制器通过比较入口和出口结点的数据包转发统计值发现数据转发中的异常. 一旦发现入口和出口结点的数据转发统计值不一致,控制器则从转发这些数据包的所有结点中的统计值来定位出现转发异常的结点. LPV 在提供有效数据转发验证的基础上,有效避免了读取结点状态引入的额外计算、通信和存储开销. 我们使用开源控制器 Floodlight 和软件交换机 ofsoftware13 实现了 LPV. 在 Mininet 模拟环境中进行的实验表明 LPV 能够有效实现数据转发验证,并仅引入了不超过 10%的通信开销和转发延迟.

本文的贡献主要有以下 3 点:

(1)提出了一个轻量级的数据转发验证框架,实现数据包和数据包转发路径的实时验证.

(2)提出了有效的数据转发的一致性检测方案,引入随机检测模型来动态地验证数据包,以较小的开销代价实现高效的转发验证.

(3)基于开源的 Floodlight 控制和开源的软件交换机 ofsoftware13 实现了 LPV 的原型系统,并在 Mininet 模拟网络环境中进行了性能评价.

2 问题描述

2.1 攻击模型

在本文中,我们假设攻击者可以在网络中控制少量恶意结点,并且可以通过修改流表项或者发布恶意流表丢弃数据包或错误转发数据包. 同时,如果有恶意主机连接恶意结点的情况下,攻击者可以对经过恶意结点的包中数据进行任意的替换、删除、增加或伪造数据包注入网络并转发至想要攻击的目的地标. 但是,攻击者无法获知正常转发结点的流表项或数据信息.

此外,我们假设攻击者无法操纵控制器来为其服务,现有的系统保护方案可以有效保护控制器的安全性,防止控制器被恶意控制. 此外,与现有的数据转发工作类似,我们假设网络中任一数据转发路径上恶意结点的数量小于路径中结点数量的一半. 如果某条路径上恶意结点过多,它们可以共谋陷害正常结点,使得网络转发完全失效.

2.2 SDN 介绍

在当今社会中,网络已经成为了人们现代生活不可或缺的基础设施,它已经深入到人们的生产、生活和学习等方方面面. 然而,随着人们对网络的需求越来越多,而传统的基于 IP 的网络架构在每次提出特定需求后都需要对网络架构进行重构,网络维护和配置的成本十分巨大,面临的问题和挑战也在增多,已经越来越不能满足用户对网络的需求. 面对日渐复杂的网络需求,为了适应互联网的发展,业内有了改变网络基本体系结构、采用新的设计理念的主流意见,来对网络体系结构的性质和功能提出要求[9]. 近年来软件定义网络(Software-Defined Network,SDN)就在改变当前的传统的网络架构,并且成为了近几年工业界和学术界的一大热点. SDN 诞生于美国 GENI[10-11]项目资助的斯坦福大学 Clean Slate① 课题,它是一种新型网络体系结构,是网络虚拟化的一种实现方式. 通过软件编程的形式定义和控制网络,是网络领域的重大突破. 由于其控制平面和数据平面分离以及可编程的本质特性,SDN 使得网络管理变得更加简单、智能.

SDN 的核心是使网络软件化并充分开放,从而使得网络能够像软件一样灵活、便捷,同时提高网络的创新能力. SDN 技术通过把原有封闭的体系解耦为数据平面、控制平面,并为网络提供可编程的接口,从而革命性地改变了现有的网络架构[12-15]. 利用分层的思想, SDN 将数据与控制相分离. 在控制层,包括具有逻辑中心化和可编程的控制器,可掌握全局网络信息,方便运营商和科研人员管理配置网络和部署新协议等[16]. 在数据层,交换机仅提供简单的数据转发功能,可以快速处理匹配的数据包,适应流量日益增长的需求. 两层之间采用开放的统一接口进行交互. 控制器下发遵循统一标准规则的流表,交换机仅需按照这些规则执行相应的动作即可. 因此,SDN 技术能够有效降低设备负载,协助网络运营商更好地控制基础设施,可以有效降低整体运营成本.

SDN 架构如图 1,应用层包括各种不同的业务和应用,可以管理和控制网络对应的转发、处理策略;控制层主要负责处理数据层资源的抽象,支持网络拓扑、状态信息的汇总和维护,并基于应用来调用不同的数据层资源,从而对数据层进行决策;基础设施层,即数据层,负责基于流表的数据处理、转发和状态收集. 应用层和数据层之间的接口称为北向接口,控制层和数据层之间的接口称为南向接口. OpenFlow[17]是第一个定义在 SDN 控制层和数据层之间的标准通信接口,即南向接口标准,OpenFlow

① Clean Slate. http://cleanslate.stanford.edu/2016,6,25

协议[①]也渐渐发展成为主流协议，其定义了 SDN 控制层和数据层之间的通信标准. SDN 控制器通过 OpenFlow 协议控制交换机对数据包的转发. 下面对本文用到的 4 个关键点进行介绍：

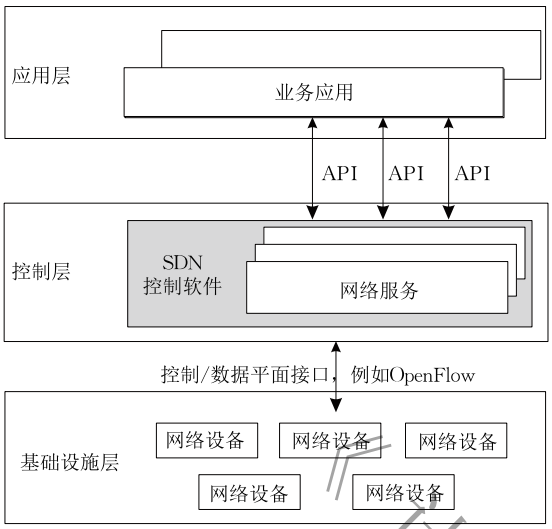

图 1　ONF 提供的 SDN 架构

(1) 流表. SDN 中交换机中的流表负责数据的转发处理，每个流表包含多个流表项，每个流表项的主要组件如表 1，Match Fields 是匹配域，根据这个去匹配数据包；Priority 是优先级，数值越大，优先级越高，优先级高的先匹配先执行；Counters 是计数器，该流表规定每处理一个数据包，计数器的数量值都加 1.

表 1　流表项的主要组成

Match Fields	Priority	Counters	Timeout	Cookie	Flags

通过查看转发路径上同一条流的流表项的计数器的数量值，可以分析出该流的数据包是否都正常转发.

(2) 转发机制. 当新流的数据包到达交换机后，交换机流表中没有处理该流的流表项，交换机遍历所有流表项后，会有找不到匹配项的情况，即 table-miss，这时，交换机会向控制器发送 Packet-in 消息，控制器收到后，查询该数据包的转发路径，下发 Packet-out 消息将当前数据包转发出去，并发送 flow_mod 消息给路径上所有交换机下发流表项. 该流的后续数据包到达交换机后，会根据流表里的规则进行转发，交换机不会再发送 Packet-in 消息.

(3) Packet-in 消息. Packet-in 消息机制是 SDN 中一项重要交互方式，承担了大部分控制器关于交换机的消息来源，大部分交换机向控制器发送的交互信息都是通过 Packet-in 消息发送的. 除了因为 table-miss 触发产生，还可以通过流表项的 output: controller 动作触发. 我们可以利用 Packet-in 消息来对数据包完整性以及数量值进行验证.

(4) 组表. 组表的存在使得 OpenFlow 中的转发有多种方法. 如表 2 所示，组表里面包括多条组表规则，每条组表规则由组标识进行区分，并且每条组表规则还包含：组类型，用来决定组表的执行类型；计数器，记录被每个组处理的数据包数量；动作桶集，包含一系列的动作桶，每个动作桶包含一些动作集及相关参数. 组类型有 4 种：indirect，all，select，fast failover. 当组类型为 indirect 时，执行指定的动作桶，这种类型只支持单个桶；当组类型为 all 时，执行组里的所有桶，用于多播或者广播；当组类型为 select时，有选择性地执行组的一个动作桶，选择基于交换机计算的选择算法，例如，基于用户指定元组进行哈希或者简单的轮循；当组类型为 fast failover 时，执行第一个活动的动作桶. 利用组表的 select 类型特性，可以实现随机地给数据包添加标签.

表 2　组表条目的主要组成

Group Identifier	Group Type	Counters	Action Buckets

2.3　相关工作

在对 SDN 缺陷的研究中 Benton 等人[18]根据最近对 SDN 部署的一些情况总结出了一个简短的 Openflow 协议存在漏洞的概览，并且提出了一些解决的建议，Bifulco 和 Cui 等人[19]提出了一种攻击，允许攻击者采集网络包转发逻辑的图谱. 远端攻击者获取特定流规则，并且利用这一知识使得攻击者了解转发逻辑，然后利用 RTT，包交换信息的分布等对网络进行多种攻击. Dhamecha 等人[20]总结了在单控制器情况下 SDN 网络存在的问题和安全挑战，Wang 等人[21]提出了一种针对 SDN 的 DDOS 攻击，攻击者利用 table-miss 触发 Packet-in 这种机制发送大量 Packet-in 消息，消耗网络资源. 同时也提出了他们的解决机制 FloodGuard，利用主动流规则分析和包迁移对网络进行防护. Prasad 等人[22]讨论了至今还尚未很好解决并且对 SDN 有很大影响的几类攻击，同时对最近的一些解决方案进行了评

① OpenFlow. OpenFlow1. 5. 0Specification. https://www. opennetworking. org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1. 5. 0. noipr. pdf /2016，6，25

估，Alharbi 等人[23]讨论了 SDN 中链路发现机制的缺陷，并且提出了一种基于 HMAC 的源认证的解决方案.

交换机是网络核心设备，控制着数据包转发，当这些设备出现异常或者被恶意入侵后，会出现数据包转发异常甚至丢包. 数据包转发验证在 IP 网络中被广泛研究[24-36]，大致可以将这些研究分为三类：

第一类是通过添加标签或签名来进行验证的方法[28-30]，这类方法很容易检测出数据包篡改，转发路径上的交换机通过验证嵌入在数据包里的标签来判断数据包是否正常转发. 但是，这些方法不能直接应用到 SDN 中，因为 OpenFlow 交换机只具备简单的数据包处理能力，不能处理那些复杂的标签操作，并且，这些方法带来了巨大的通信和计算开销.

第二类是通过流数量分析来检测转发异常[31-33]. 这些方法通过分析路径上每个交换机的入端口和出端口上的流统计信息来判断所有的流量是否被正常转发. 但是，这些方法需要很严格的时钟同步，所以，这些方法在实际网络中不易于部署，特别是大型网络. 而且，这些方法不能检测数据包篡改攻击.

第三类方法是数据包探测和消息验证机制[24-27,34-36]. 数据包探测方法[34-36]需要生成探测包去验证转发路径，这给网络带来巨大的通信开销. 消息验证方法[24-27]可以通过相邻交换机之间频繁的交互检测恶意交换机，每个交换机需要保存经过的所有流的路径，然后收集对应的确认包. 这些方法需要巨大的计算和存储开销，但是 SDN 交换机只具备简单的数据包处理能力，并不具备跟其他交换机通信来确认转达数据包的能力，所以，这些方法在 SDN 中不适用.

SDN 虽然是近年来涌现的新兴网络技术，但其网络转发相关的很多方案以及关注重点都有来自于传统网络的一些经验和启发[8,36-37]. 例如，SDNsec[36]采用的是第一类方法，即添加流标签验证数据转发. SDNsec 让每个交换机在它转发的数据包包头中都嵌入一个密码标签，通过检查数据包头部的密码标签集验证数据包是否正常转发. 但是，SDNsec 需要修改交换机内部原有实现机制来向数据包包头添加证据. 而 LPV 添加的标签不需要修改数据包原有结构. 此外，SDN Traceroute[37]采用的是第三类方法，即发送探测包，让探测包逐跳产生 Packet-in 来进行验证，但这种方式引入了较大的通信开销，仅适用于网络调试，无法对大规模的流量进行长期监视. 此外，它也没有考虑数据包可能被篡改的情况.

与目前已有的数据包转发验证方案相比，本方案优势是利用 SDN 的可信中央控制器，减小设备间确保正确信息传输所需要付出的代价，减小分布式方案无仲裁情况下数据比对和各个设备间通信的开销. 另外，多路径情况下，由于本方案检测粒度在流级别，因此只需要添加对同一流不同路径的流聚合，即可适用于多路径情况.

与我们工作最相似的是 Sphinx[8]，它采用第二类方法，在对软件定义 SDN 不做任何修改的前提下，提出了数据包转发异常检测方法，它收集所有交换机上的流数量信息，通过流数量的相关分析来识别转发异常. Sphinx 针对一些可能会影响整个网络的网络拓扑、数据层转发，甚至一些网络实体的对控制器的攻击进行分析，进一步提出了 Sphinx 的原型，旨在检测已知或未知的一些攻击. Sphinx 创新性地对流图进行抽象，从而更加精确地获知网络更新以及一些策略的实时信息. 首先，提供了确认有效性对所有网络的更新和限制条件从而实时确认网络性能. 其次，检测已知和未知的对网络拓扑和数据层转发的威胁. 同时 Sphinx 可以进行少量的修改从而保证用于不同的控制器实现. Sphinx 通过不断动态地学习全网信息，从而对异常产生警报. Sphinx 分析特定的网络控制信息从而获得新的网络行为或元数据为拓扑或转发状态，它为每个流量都建立流图，并且持续地更新和监视这些流图使其在允许范围内，当有标记的异常时报告. 它利用了自定义的算法可以实时地处理更新确定当时是否产生异常. 但是，这种设计不能检测出复杂的转发异常，例如，攻击者丢掉一定数量数据包并注入相同数量的数据包. 并且，Sphinx 需要控制器记录所有流量的数量并在每纳秒对这些数量进行比较，这会带来巨大的通信和计算开销. 我们的工作对这些问题都进行了很好的处理，我们通过数据包鉴定和随机地转发验证方法，在保证检测粒度的同时大大减小了通信开销.

目前学术界有很多做 SDN 安全的研究[38-46]，例如，AVANT-GUARD[38]和 FloodGuard[21]做控制层安全. 其他工作[40-44]做策略检测，这些工作主要集中在网络配置的一致性检测. 为了保证安全规则的正确实施，Flowguard[45]检测并解决数据包转发冲突并实时过滤规则. FortNox[46]和 SE-Floodlight[47]防止由策略冲突引发的各种攻击. Rosemary[48]增强了 SDN 控制层的鲁棒性. 这些工作和 LPV 都是正交的.

近几年在 SDN 安全应用方面也涌现了大量出色的工作[48-51]，FRESCO[49]实现了一个安全应用部署框架，使得多种安全服务同时部署成为可能；Yu 等人[50]提出了一种高效的 SDN 测量框架；Fayaz 等人[51]通过把流量重定向到数据中心检测不同的 DDoS 攻击；Xu 等人[52]通过不同路径上的流量监控来检测 DDoS 攻击. 不同于以上工作，我们提出的 LPV 是一种保证实时数据包转发正确性的轻量级数据包转发验证机制.

2.4　目　标

为了能够有效地解决这些威胁，我们的目标主要包括以下几个方面：

(1) 安全目标 1：确保转发数据可验证

确保目的端收到的数据包是正确的，未经篡改的原始数据，此外，确保目的端不会接收到由中间恶意结点伪造的并声称是某个源结点发送的数据包.

(2) 安全目标 2：验证数据转发的正确性并定位异常结点

确保数据包转发路径的正确性，防止恶意中间结点将数据包丢弃、篡改数据包内容或者直接进行流量劫持.

(3) 安全目标 3：异常结点定位和移除

隔离检测到的异常结点及路径，构造出新的可用拓扑，利用新的拓扑生成新的转发路径，实现正常的网络功能.

(4) 性能目标：实现轻量级的消息和转发路径认证

实现轻量级的消息和转发路径认证方法，既能保证控制器收集到的信息并未受到来自中间结点攻击，同时只引入较小的通信和计算开销. 现有的控制器如 Floodlight 实现了控制器与交换机之间的 SSL 引入了大量的计算开销以及密钥的存储开销. 此外，可以大大降低控制器维护结点状态的计算和通信开销成本.

3　概　述

在本节我们将介绍 LPV 设计的概要并且分析设计 LPV 的挑战.

LPV 通过以一定概率采样和验证数据包，有效地降低 SDN 数据包转发验证中的通信和计算开销. LPV 为不同的数据包分配不同标签并分析不同标签对应的数据包统计值，检测数据包转发路径的异常. 具体来说，通过数据包标签检测出数据包的篡改，同时基于标签的转发统计检测数据包是否能够验证正确数据转发. 为了实现这个目标，LPV 通过控制器下发一系列的特殊组表，在第一跳交换机处对某一特定流的数据包打标签，中间某一交换机对打标签的包中一些特殊标签进行流量统计，最后一跳交换机根据不同的标签进行不同的转发动作. 主要的工作步骤如图 2 所示.

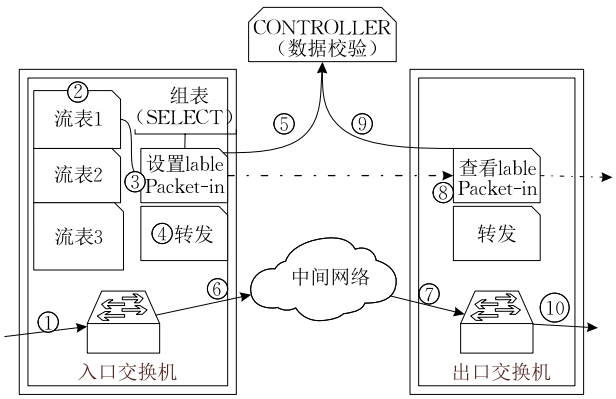

图 2　LPV 概述

3.1　数据包透明标记

第一跳交换机随机选择需要检测验证的数据包，并为这些数据包添加上对中间交换机透明的特殊标记. 我们把这一类特殊标统称为 Label-S，这一类数据包需要进行转发一致性检测. 当第一跳交换机要对某个包打上特殊标签 Label-S 同时，会将数据包中数据部分的摘要发送给控制器. 当这个包转发到达出口交换机时，出口交换机会根据标签对数据进行不同处理，对有 Label-S 特殊标签的数据包执行特殊的校验和验证动作.

3.2　数据一致性校验

当交换机根据流表规则向控制器发送 Packet-in 消息时，交换机首先计算出要发的数据包的摘要值，将这个摘要值发送给控制器，控制器收到后对不同交换机的 Packet-in 数据进行比对，验证数据在发送过程中是否被篡改. 同时，控制器通过比较数据包的数量检测数据包是否被恶意丢弃. 当控制器比对发现异常时，例如，Packet-in 的数据内容匹配失败或者转发的数据包数量不一致，系统可以判定数据包在发送的过程中被恶意篡改或丢弃控制器读取中间交换机状态.

3.3　异常结点定位

当一致性检测未通过时，控制器可以通过比较收集到信息定位出现故障的链路，并从相应链路所有交换机中获取相关数据包的转发信息以分析故障或者恶意结点. 一旦确认异常结点以后，LPV 可以

在网络拓扑中删除这些结点，从而生成新的可用网络拓扑以产生新的数据转发规则.

3.4 设计 LPV 的挑战

实现 LPV 还有以下 3 个挑战：

(1) 如何实现轻量级的消息正确性验证？简单校验所有数据包的完整性和转发路径的正确性会导致数据包转发过程中引入非常大的转发延迟并导致系统吞吐的降低.

(2) 为保证转发数据的正确性，每个数据包进入网络都要将数据包中的一部分发送至控制器用于数据正确性的验证，这样就会引入大量的通信开销和存储开销，因此如何有效地在验证数据正确性的同时降低数据转发正确性验证开销是另外一个挑战.

(3) 如何能够在检测到数据包转发异常后实时定位故障或者恶意结点？

3.5 总 结

LPV 系统主要工作流程如图 3 所示：当数据包进入网络后，与数据源结点直接相连的第一跳交换机将包的数据的一部分处理后作为验证签名发送至控制器保存，然后包在网络中进行正常转发，当包到达最后一跳交换机即将交付至目的结点时，最后一跳交换机将包的同一部分数据进行同样的操作然后发送至控制器，控制器根据存储的内容进行交叉对比验证，通过后告知最后一跳交换机交付目的结点，未通过则认为包被恶意篡改或数据包为恶意结点伪造后恶意注入，同时对路径标记为可疑路径. 最后，控制器收集流量信息，对比可疑路径上每跳交换机对应路径流的转发流量，当统计数据显示某跳交换机有异常时，将异常交换机其加入黑名单.

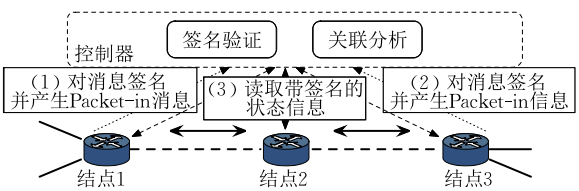

图 3 LPV 核心工作流程

4 方案设计

本节我们将展开介绍 LPV 中数据包透明标记、数据一致性校验和异常结点定位方案并解决 LPV 设计中的挑战.

4.1 数据流标签

本方案采用添加标签的方式来对数据包进行采样. 我们将数据包添加的标签分成两类，一类是特殊标签 Label-S 标签，另外一类是混淆标签 Label-C (即与 Label-S 标签不同的任意标签)，对于任意中间结点来说，这些标签之间没有任何差异. 即使有恶意结点可以对每个流进行计算和监控，也无法区分被验证的数据包. 因此，LPV 能有效防止恶意结点通过监听转发的数据包以干扰 LPV 方案的运行.

LPV 中，我们把每个转发路径上的交换机分为三类：出口、入口交换机以及中间交换机. 当网络中发生新的路径请求，根据 OpenFlow 协议的路由机制，入口交换机会向控制器发送 Packet-in 消息从而获得转发信息. 控制器会根据维护的全局网络拓扑图，计算该流的转发路径，并下发给路径上的各跳交换机. 在本系统中，控制器会在第一跳交换机中设置组表，并在组表中写入两类动作桶，一类动作桶用来向数据包添加标签 Label-S，用来标识检测流；另一类动作桶用来向数据包添加不是 Label-S 的任意标签，即利用混淆标签来标识非检测流. 同时，在第一跳交换机和最后一跳交换机的流表中写入一条新的流表规则，该规则使得标签为 Label-S 的数据包都以 Packet-in 消息的形式上报给控制器. 当该流的后续数据包到达入口交换机后，首先会执行交换机指定的随机选择算法选择的动作桶中的动作集，即加上特殊标签 Label-S，或不是 Label-S 的其它任意标签 Label-C. 当数据包到达最后一跳交换机后，如果数据包上写入的是特殊标签 Label-S，根据新添加的流表规则，它会通过 Packet-in 消息的形式上报给控制器，控制器收到后，会进行后续校验工作；如果标签不是 Label-S，那么交换机会按照控制器下发的转发规则正常转发该数据包.

流表下发的具体过程如算法 1 所示. 首先控制器判定发送 Packet-in 消息的是否是第一跳交换机发送的并且 Packet-in 触发原因为没有匹配流表项 (Packet-in 的 reason 字段为 NoMatch)，如果为否，则进入算法 2 的数据一致性校验 (如第 14 行所示). 如果为是，则首先获取路径上的所有交换机(如第 2 行所示). 控制器向除第一跳以及最后交换机下发正常转发流表表项 (3 行～6 行所示). 对于第一跳交换机下发组表，组表中的动作为：(1) 给数据包打特殊标签 Label-S，并发送 Packet-in；(2) 给数据包打任意标签 Label-C，并正常转发. 算法引入检测因子 θ 来控制流的采样和检测比例，θ 值表示添加混淆标签 Label-C 的组表项数量与添加特殊标签 Label-S 的组表项数量的比值. 当 $\theta=1$ 时，组表中添

加 Label-C 的表项和添加 Label-S 表项的数量比值为 1∶1，即当前流的采样比例约等于 50%. 当设定检测因子 θ 后，系统按照 1∶θ 的比例分别写入用于添加 Label-S 的表项和用于添加 Label-C 的表项，从而控制触发对 Packet-in 消息的采样概率（7 行～10 行所示）. 另外本方案采用了比值 θ 而非具体数量来表达两种流表项数量，由此在保证采样比例已知且不变的情况下，通过多次复写两类组表项来添加更多的混淆标签，有效防止了由于混淆标签种类数量太少而导致攻击者通过统计标签量的方式绕过本文提出的防御机制. 最后，对于最后一跳交换机，写入两条流表项：(i) 当标签为 Label-C 时则数据包正常转发；(ii) 否则标签为 Label-S，则交换机触发 Packet-in 消息并对数据包进行相应处理以后发给控制器（11 行～13 行所示）.

算法 1.　组表下发 (Flow Rule Generation).

输入：*Packet-in* 消息；检测因子 θ

输出：*GroupTable* 表项

```
1.  IF getHop (Packet-in)==ingress && reason==
        noMatch THEN
2.    Path=GetPath (Packet-in);
3.    FOR(int i=0; i<Path.length; i++)
4.     SwitchID =getSwtichID(Path(i));
5.      IF(i!=0 && i!=Path. length-1) THEN
6.        pushFlowEntry(SwitchID);
7.      ELSE IF (i==0) THEN
8.        FOR (int j=0; j<θ; j++)
9.          setGroupTable(setAction(forward,Label-C));
10.         setGroupTable (setAction (Packet-In,Label-S));
11.       ENDFOR
12.     ELSE
13.       setFlowEntry(setAction(Packet-in,Label-S));
14.       setFlowEntry(setAction(forward,Label-C));
15.     ENDIF
16. ELSE
17.    ConsistencyVerification(Packet-in);
18. ENDIF
```

4.2　数据一致性校验

LPV 根据数据包的标签随机对数据包进行一致性校验. 在添加标签动作完成后，标签为 Label-S 的待检测数据包会通过 Packet-in 消息发送到控制器（交换机根据 1.1 节中控制器下发的流表规则来产生 Packet-in 消息）. 这里我们对 Packet-in 消息格式进行了修改，交换机会把数据包内容部分利用 SHA-1 算法计算消息摘要并提取前 64 位作为消息摘要. 之后再将摘要值以及本流的首部信息和密钥 K 作为输入使用散列函数 SHA-1 用密钥 K，对消息 M 进行 HMAC 操作，将摘要写入 Packet-in 消息的 data 字段，将得出的 HAMC 值写入 Packet-in 的 cookie 字段然后再向控制器发送. 本文假设秘钥通过静态配置存储在控制器和各交换机中，我们可以利用已有的秘钥分发技术实现动态秘钥分发. 控制器收到加载有摘要信息和 HMAC 值的 Packet-in 消息后，首先将该消息的头部信息以及 data 字段取出来，把这两部分信息联结成消息 M，并选取相应密钥 K，使用散列函数 SHA-1 用密钥 K 对消息 M 进行 HMAC 操作，得到 HMAC 值，当计算出结果与 cookie 字段值相同时，控制器才认为收到的数据是有效的信息，才继续进行后续操作，否则直接丢弃这个包，本步骤主要是对信息源进行认证，确保控制器收到数据是真实数据.

当信息源认证通过后，控制器会进行一致性认证，将 cookie 字段的 HMAC 值数据提取出来存储至交换机的校验表 LPV-table 中，数据包按照控制器下发的正常转发流表项进行转发，当到达出口交换机时，同样地，根据流表规则，出口交换机也将标签为 Label-S 的数据包内容部分的摘要写入 Packet-in 消息的 data 字段后，将其发送给控制器，控制器收到 Packet-in 消息后，用密钥 K 做与入口交换机相同的 HMAC 计算. 将计算出来的 HMAC 值与 cookie 中的值进行对比，进行数据源认证. 通过后，将 cookie 字段的值与 LPV-table 中数据进行比对，如果存在相同值，则将表中对应的值删除，同时判定本次数据校验通过，如果未能找到匹配的相同值，则认为数据包在转发过程中被恶意篡改、丢弃或这个包是被恶意注入的虚假数据包. LPV-table 的数据结构如表 3 所示.

表 3　LPV-table 表字段的描述

字段	说明
MacAddress srcMac	源 MAC 地址
MacAddress dstMac	目的 MAC 地址
IPv4Address srcIp；	源 IP 地址
IPv4Address dstIp；	目的 IP 地址
IpProtocol ip_Proto；	IP 协议号
TransportPort SourcePort；	源端口号
TransportPort DestinationPort；	目的端口号
DatapathId dstSwDpid；	交换机 id
U64 cookie；	Cookie 字段（存数据 HASH 值）
int compareTag	比较标记（用于删除表项的标记）
TimeoutAlert alert	控制器存储包相关 MAC 的超时时钟

具体校验过程如算法 2 所示. 控制器先将 Packet-in 消息的 data 字段作为输入运行 HMAC

算法，将结果与 cookie 字段值对比，如果结果相同则数据源正确，继续后续操作，否则丢弃这个 Packet-in 包(1 行～3 行所示). 数据源认证通过后，判定 Packet-in 消息来自哪一跳，如果是路径上的第一跳交换机发送的，则将 Packet-in 消息中的 cookie 字段值存储至 LPV-table 中(4 行和 5 行所示). 如果是最后一跳交换机，则将 cookie 字段值从 LPV-table 查找对应项，如果有相同项，则从 LPV-table 中删除这一项，并且认为校验通过，如果未找到，则将本路径加入黑名单中(6 行～9 行所示). 另外，LPV-table 中表项都设置一个超时时钟. 若时钟超时，也将相应路径加入到黑名单中. 在实验中发现超过 95% 的数据包转发延迟都小 50ms，因此我们将超时时钟设定为 50ms，这样在确保大部分情况下不产生误报的同时也加快检测异常的速度.

算法 2. 数据包校验(Consistency Verification).

输入：*Packet-in* 消息

输出：*pathBlackList*

1. *switchID*=*getSwitchID*(*Packet-in*);
2. IF *getCookie*(*Packet-in*)!=*Hmac*(*getData*(*Packet-in*),*get Key*(*switchID*)) THEN
3. *Drop*(*Packet-in*);
4. ELSE IF *getHop*(*Packet-in*)==*ingress* THEN
5. *LPV-table*.*save*(*getCookie*(*Packet-in*))
6. ELSE IF *getHop*(*Packet-in*)==*egress* THEN
7. *com*=*LPV-table*.*equal*(*getCookie*(*Packet-in*));
8. IF(*com*) THEN
9. RETURN;
10. ELSE
11. *pathBlacklist*.*add*(*getPath*(*Packet-in*));
12. ENDIF
13. ENIF

4.3 异常结点定位

LPV 控制器验证转发异常并生成转发路径黑名单，并根据黑名单重新计算数据转发路径. 控制器周期性(3 s/次～5 s/次)的获取路径黑名单信息，数据包校验未通过时会直接触发控制器读流状态信息，并将本流通过路径加入到黑名单中. 在数据包校验未通过情况下会先触发控制器获取第一跳及最后一跳流状态信息，如果流统计数量小于某个差值，则只将路径加入黑名单中，如果大于某个差值，则直接触发控制器获取整个路径上的所有结点的流状态信息，并进行比对. 当一致性检测失败后，我们需要分析和比较在这条路径上每一跳交换机的转发行为. 当一个交换机对应转发流的流量统计值或转发包的数量值大于其上一跳交换机或者小于上一跳交换机，且差值比例大于预设的阈值 λ 时，则认定这个交换机有异常行为. 在这里 λ 值为一个比例值，为两跳交换机中对应转发流的流量统计值差值的比例. 通过不同 λ 值的实验，我们发现从 $\lambda=1\%$ 开始，随着 λ 的增加系统对注入攻击和篡改攻击的检测准确率会逐渐上升. 同时，系统对恶意丢包的检测准确率会从 $\lambda=3\%$ 开始逐渐降低，当 λ 约为 5% 时系统不同类型攻击的检测准确率曲线相交. 也就是说，当 $\lambda=5\%$ 时系统能够有效检测各种攻击. 因此，在本文中我们异常结点定位的阈值 λ 设置为 5%. 当认定交换机有异常行为后系统会将这个交换机加入到黑名单中，将本路径从黑名单中删除，之后控制器会将本结点从其拓扑中删除并自动更新控制器的拓扑，并且向所有与黑名单结点相连交换机下发指令关闭相连端口，在这之后，所有流经本结点的流都会重新触发寻路请求，最终所有流量都会根据控制器生成的新拓扑获得新的转发路径，从而对涉及黑名单结点的流的转发路径进行更新.

具体异常结点定位过程如算法 3 所示. 控制器定期查询 pathBlackList，如果黑名单不为空，则对任一黑名单路径流进行以下操作：(1) 获取本路径所有交换机；(2) 获取所有对应流的流量统计记录 count 值(1 行～6 行所示). 对于路径上任意两条，如果本跳 count 大于前一跳 count 值或本跳 count 小于前一跳 count，且差值比例大于阈值 λ，则认为这个交换机有异常行为，将本交换机加入 switchBlackList 中 (7 行～11 行所示).

算法 3. 异常定位(Localization).

输入：*pathBlackList*, *byteCount*, *threshold* λ

输出：*switchBlackList*

1. FOR (*i*=*pathBlackList*.*length*;*i*>0;*i*++)
2. *switchList*=*getPath* (*pathBlackList*(*i*));
3. FOR (*j*=0;*j*<*switchList*.*length*;*j*++)
4. *count*[*j*]=*getByteCount* (*switchList*(*j*));
5. *count*[j+1]=*getByteCount* (*switchList*(*j*+1));
6. *switchID*=*getSwitchID* (*switchList*(*j*+1));
7. IF (*count*[*j*]<*count*[*j*+1]) THEN
8. *switchBlackList*.*add* (*switchID*);
9. ELSE IF ((*count*[*j*]−*count*[*j*+1])/*count*[j]>λ)
10. *switchBlackList*.*add* (*switchID*);
11. ENDIF
12. ENDFOR
13. *pathBlackList*.*delete* (*i*);
14. ENDFOR

5　实验及评估

本部分我们对 LPV 的性能进行了展示，分别从发包的延迟，吞吐率的变化及准确率等方面对 LPV 进行评估.

5.1　实验设置

我们使用华为 RH2288 作为实验运行服务器，其配置为 Intel Xeon E5-2609 CPU，128GB 内存，运行 64 位 Ubuntu 操作系统. 使用 Mininet 模拟网络环境，开源 ofsoftware13 交换机，开源 FloodLight 控制器. 同时使用 Cbench 对控制器进行压力测试，检测运行 LPV 情况下控制器处理 Packet-in 消息数量. 模拟网络使用 3 层树形拓扑结构，连接约 1000 台模拟主机（depth＝3，fanout＝10，参数分别为树形结构的深度和每个结点的子结点数量）. 我们把阈值 λ 设定为 5%. 阈值的设定根据我们的统计得出，当阈值 λ 为 5%时，LPV 对多种恶意攻击的综合误报率与漏报率都相对较低，综合准确率最高，所以为获得实验的最优综合性能，我们将 λ 设定为 5%.

5.2　实验结果

在实验 1 中，我们使用自定义发包脚本，分别在使用及未使用 LPV 的系统中发送 10 万个包，使用 Wireshark 抓包程序分别从网络入口及触发抓取数据包，计算数据包的转发延迟，描绘转发延迟的 CDF 图像. 如图 4 所示，可以看出在不使用 LPV 的情况下，15 ms以下延迟占 59%，使用 LPV 情况下 15 ms 以下延迟占 56%，即在使用 LPV 的系统中，有额外约 3%的数据包延迟大于15 ms. 未使用 LPV 的系统中，有 87%的数据包延迟在40 ms以下，使用 LPV 的系统中，这一比例为 83%. 即在使用 LPV 的系统中，有额外约 4%的数据包延迟大于40 ms. 未使用 LPV 系统的所有数据包的平均转发延迟为 30.32 ms，使用 LPV 系统的所有数据包的平均转发延迟为33.17 ms.

在实验 2 中，我们分别在中间交换机上对数据包进行恶意丢弃、注入虚假数据包、篡改数据包等操作，检测因子 θ 为 7，即特殊标签触发率约为 12%，各个攻击分别持续10 s，各做 100 次，如果 LPV 未做出反应则记录为漏报.

同时，在正常情况下运行系统 100 次，每次持续 60 s，如果 LPV 报出异常则记录为误报. 分别以 10%、20%和 30%的概率分别进行随机恶意丢弃、随机注入虚假数据包和随机篡改数据包操作，并最终取平均值.

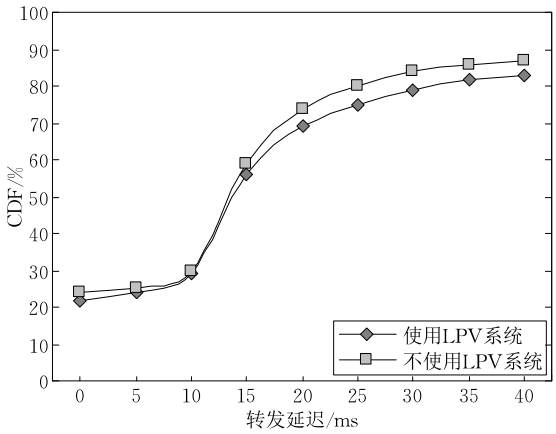

图 4　转发延迟 CDF

实验结果如图 5 所示，其中，误报率约为 3%，主要原因是 LPV-table 表项超时时钟为 50 ms，当部分数据包转发延迟超过 50 ms 时，系统会产生误报，但是调高 LPV-table 表项的超时时钟会直接影响对丢弃数据包攻击的检测时间，因此 50 ms 依然为最优值. 其他数据中，插入数据包和恶意丢包的漏报率都较低，分别为 2%和 4%，主要原因是由于对于两种攻击数据包校验及 count 值分析都能够检测部分情况，而篡改内容攻击主要依靠数据包校验，因此对内容篡改攻击检测漏报约为 10%. 同时，由于阈值 λ 的影响，部分数量较少的丢包未触发异常检测，所以恶意丢包漏报率要略高于注入包攻击.

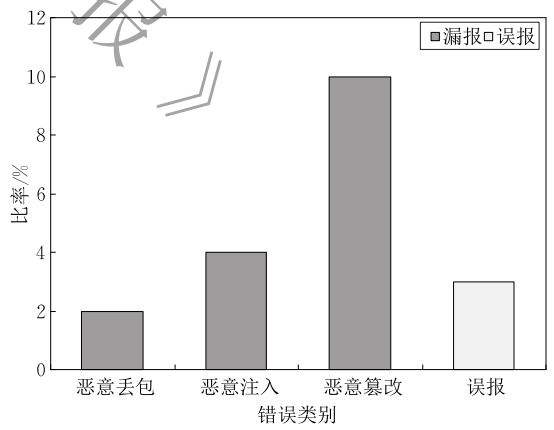

图 5　检测准确率

在实验 3 中，我们在分别设定不同检测因子 θ 的情况下，对网络吞吐率进行了测量. 实验结果如图 6 所示，θ 分别为 7，4，3，1 对应的触发几率约为 12%，20%，25%，50%. 以未使用 LPV 的系统吞吐率作为基准，在 θ＝7 时，吞吐率下降了 7%左右，而 θ＝1 时，吞吐率下降最多，降低了约 34%，但是从实验 2 可知，θ＝7 时已经能够得到较好的准确率，而

$\theta=7$ 时，吞吐率降低不到 10%.

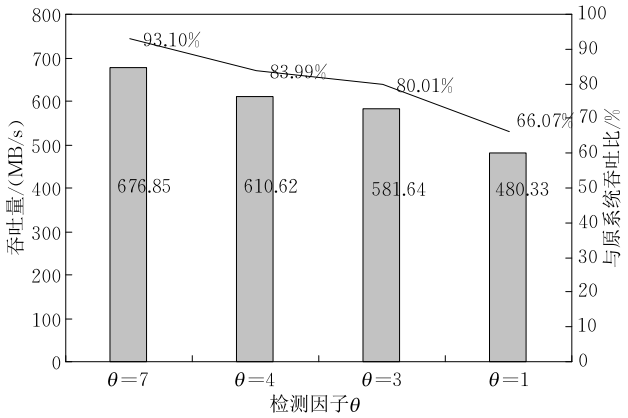

图 6 吞吐量开销

在实验 4，我们通过测量在不同 FlowMods 下发速率的情况下，LPV 对控制器处理 Packet-in 消息时间的影响. 实验结果如图 7 所示. 我们可以看出在 FlowMods 在 2000/s 以下时，Packet-in 消息处理时间增加了约 100 μs. 延迟的主要原因是部分 Packet-in 消息需要进行 HMAC 以及查表运算，当 FlowMods 速率大于 2000/s 时，不论是否使用 LPV，Packet-in 消息的处理时间都会突然增长，主要原因是由于处理速度无法保证导致队列增加，在这中情况下，使用 LPV 系统的 Packet-in 消息处理时间要比不使用 LPV 的系统处理时间长约 150 μs 也是由于队列的原因.

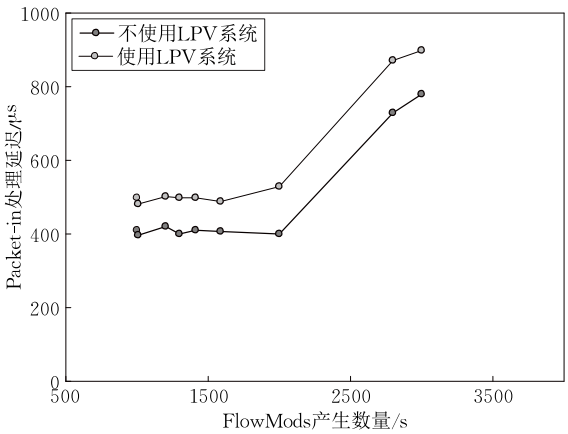

图 7 不同 FlowMods 速率下 Packet-in 处理延迟

通过以上实验结果可以看出，检测的准确率受到检测因子 θ 以及阈值 λ 的共同影响. 阈值 λ 我们已取到综合最优值 5%. 检测因子 θ 越大则漏报率越高，同时误报率越低，这也就意味着较高的采样比例可以获得较高的安全性能. 同时，吞吐量影响与检测因子 θ 成反比关系，检测因子 θ 越大，吞吐量收到的影响越小，而根据实验 2 中的结果可以显示，在检测因子 $\theta=7$ 时，已经可以获得较好的检测准确率，同时根据实验 1 和实验 3 的结果可知，获得这种准确率并不需要付出较高的延迟以及吞吐开销.

根据实验 4 的结果可知，由于 LPV 系统需要控制器进行部分额外的计算操作，因此增加了部分 Packet-in 处理时间，特别是在 FlowMods 发放速率较高的情况下，这一时间继续增加，而 FlowMods 速率与网络中交换机数量成正比，因此，在网络启动初始或突然新增大量设备的情况下，LPV 系统可能会面临一些挑战，由于额外的控制器处理时间以及 LPV-table timeout 的时间设置，会导致部分误报增加，当网络趋于稳定时，这部分额外误报会消失，幸运的是网络不会频繁的初始化.

最后，我们利用 Cbench 对控制器进行了压力测试，我们发现在持续了大约 600 s 的时间里，CPU 的平均使用率约为 17%左右，造成较高 CPU 使用率的原因是模拟网络环境以及大量的虚拟主机的存在，在不启动控制器的情况下，静态拓扑发送同样规模的数据包，CPU 使用率约为 11%，因此 LPV 控制器只消耗了约 5%的 CPU 资源，而内存资源使用率约为 27%，同样的消除模拟网络消耗的资源，LPV 系统消耗的内存资源约为 8%，这部分内存资源的消耗主要是由于保存 LPV-table 以及流状态信息及 count 值引入的.

6 结 论

提出了一种基于透明标签的轻量级 SDN 数据转发验证系统方案 LPV. 对数据包进行随机抽样检测和验证，在确保检测准确率的同时有效降低了数据包转发验证的开销. 与现有单纯基于流状态检测的方案相比，LPV 可以较好检测和防御恶意的数据包篡改攻击. 在 Mininet 中进行的实验表明 LPV 系统在仅增加了较少转发延迟以及小于 10%的吞吐开销的情况下，可以有效检测和防御的数据包篡改和流量截止等攻击. 下一步工作，我们将基于硬件交换机实现 LPV，并通过规模部署实现评价在大规模网络中的验证性能和开销.

参 考 文 献

[1] Ferguson A D, Guha A, Liang C, et al. Hierarchical policies for software defined networks//Proceedings of the Workshop on Hot Topics in Software Defined NETWORKING. Helsinki, Finland, 2012: 37-42

[2] Tavakoli A, Casado M, Koponen T, Shenker S. Applying NOX to the datacenter//Proceedings of the ACM SIGCOMM Hot Topics in Networks. New York, USA, 2009: 1-6

[3] Cui Y, Xiao S, Liao C, et al. Data centers as software defined networks: Traffic redundancy elimination with wireless cards at routers. IEEE Journal on Selected Areas in Communications, 2013, 31(12): 2658-2672

[4] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic flow scheduling for data center networks//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation. San Jose, USA, 2010: 281-296

[5] Liu H H, Wu X, Zhang M, et al. zUpdate: Updating data center networks with zero loss. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 411-422

[6] Heller B, Seetharaman S, Mahadevan P, et al. ElasticTree: Saving energy in data center networks//Proceedings of the USENIX Conference on Networked Systems Design and Implementation. USENIX Association, Washington, USA, 2010: 17-17

[7] Li D, Shang Y, Chen C. Software defined green data center network with exclusive routing//Proceedings of the IEEE INFOCOM. Toronto, Canada, 2014: 1743-1751

[8] Dhawan M, Poddar R, Mahajan K, et al. Sphinx: Detecting security attacks in software-defined networks//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2015: 1-15

[9] Jain R. Internet 3.0: Ten problems with current Internet architecture and solutions for the next generation//Proceedings of the IEEE Symposium on Military Communications Conference. Washington, USA, 2006: 153-161

[10] Greenberg A, Hjalmtysson G, Maltz D A, et al. A clean slate 4D approach to network control and management. ACM SIGCOMM Computer Communication Review, 2005, 35(5): 41-54

[11] Yan H, Maltz D A, Ng T S E, et al. Tesseract: A 4D network control plane//Proceedings of the USENIX Conference on Networked Systems Design and Implementation. Santa Clara, USA, 2007: 27-27

[12] Nunes B A A, Mendonca M, Nguyen X N, et al. A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys and Tutorials, 2014, 16(3): 1617-1634

[13] Tennenhouse D L, Wetherall D J. Towards an active network architecture//Proceedings of the DARPA Active Networks Conference and Exposition. San Francisco, USA, 2002: 2-15

[14] Tennenhouse D L, Smith J M, Sincoskie W D, et al. A survey of active network research. IEEE Communications Magazine, 1997, 35(1): 80-86

[15] Gude N, Koponen T, Pettit J, et al. NOX: Towards an operating system for networks. ACM SIGCOMM Computer Communication Review, 2008, 38(3): 105-110

[16] Shenker S. The future of networking, and the past of protocols. Open Networking Summit, 2011, 20(1): 1-30

[17] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74

[18] Benton K, Camp L J, Small C. OpenFlow vulnerability assessment//Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING. Hong Kong, China, 2013: 151-152

[19] Bifulco R, Cui H, Karame G O, et al. Fingerprinting software-defined networks//Proceedings of the International Conference on Network Protocols. San Francisco, USA, 2015: 453-459

[20] Dhamecha K, Trivedi B. SDN issues a survey. International Journal of Computer Applications, 2014, 73(18): 30-35

[21] Wang H, Xu L, Gu G. FloodGuard: A DoS attack prevention extension in software-defined networks//Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks. Rio de Janeiro, Brazil, 2015: 239-250

[22] Prasad A S, Koll D, Fu X. On the security of software-defined networks//Proceedings of the 4th European Workshop on Software Defined Networks. Bilbao, Spain, 2015: 105-106

[23] Alharbi T, Portmann M, Pakzad F. The (in) security of topology discovery in software defined networks//Proceedings of the Conference on Local Computer Networks. Florida, USA, 2015: 502-505

[24] Kim H J, Basescu C, Jia L, et al. Lightweight source authentication and path validation. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 271-282

[25] Mizrak A T, Cheng Y C, Marzullo K, et al. Fatih: Detecting and isolating malicious routers//Proceedings of the International Conference on Dependable Systems and Networks. Yokohama, Japan, 2005: 538-547

[26] Naous J, Walfish M, Nicolosi A, et al. Verifying and enforcing network paths with ICING//Proceedings of the Conference on emerging Networking Experiments and Technologies. Tokyo, Japan, 2011: 1-12

[27] Avramopoulos I, Kobayashi H, Wang R, et al. Highly secure and efficient routing//Proceedings of the Conference of the IEEE Computer and Communications Societies. Hong Kong, China, 2004: 208

[28] Mahajan R, Rodrig M, Wetherall D, et al. Sustaining cooperation in multi-hop wireless networks//Proceedings of the Symposium on Networked Systems Design and Implementation. Boston, USA, 2005: 231-244

[29] Zhang X, Zhou Z, Hsiao H C, et al. ShortMAC: Efficient data-plane fault localization//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2012: 2-12

[30] Argyraki K, Maniatis P, Irzak O, et al. Loss and delay accountability for the Internet//Proceedings of the IEEE International Conference on Network Protocols. Beijing, China, 2007: 194-205

[31] Awerbuch B, Curtmola R, Holmer D, et al. ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks. ACM Transactions on Information & System Security, 2008, 10(4): 6

[32] Padmanabhan V N, Simon D R. Secure traceroute to detect faulty or malicious routing. ACM SIGCOMM Computer Communication Review, 2003, 33(1): 77-82

[33] Zhang X, Jain A, Perrig A. Packet-dropping adversary identification for data plane security//Proceedings of the ACM Conference on Emerging Network Experiment and Technology, (CONEXT 2008). DBLP. Madrid, Spain, 2008: 24

[34] Marti S. Mitigating routing misbehavior in mobile ad hoc networks//Proceedings of the International Conference on Mobile Computing and NETWORKING. Boston, USA, 2000: 255-265

[35] Liu K, Deng J, Varshney P K, et al. An acknowledgment-based approach for the detection of routing misbehavior in MANETs. IEEE Transactions on Mobile Computing, 2007, 6(5): 488-502

[36] Sasaki T, Pappas C, Lee T, et al. SDNsec: Forwarding accountability for the SDN data plane//Proceedings of the International Conference on Computer Communication and Networks. Hawaii, USA, 2016: 1-10

[37] Agarwal K, Rozner E, Dixon C, et al. SDN Traceroute: Tracing SDN forwarding without changing network behavior//Proceedings of the the Workshop on Hot Topics in Software Defined Networking. Chicago, USA, 2014: 145-150

[38] Shin S, Yegneswaran V, Porras P, et al. AVANT-GUARD: ScaLabel and vigilant switch flow management in software-defined networks//Proceedings of the ACM Sigsac Conference on Computer & Communications Security. Berlin, Germany, 2013: 413-424

[39] Skowyra R, Xu Lei, Gu Guo-Fei, et al. Effective topology tampering attacks and defenses in software-defined networks//Proceedings of the 48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18). Luxembourg, 2018: 374-385

[40] Son S, Shin S, Yegneswaran V, et al. Model checking invariant security properties in OpenFlow//Proceedings of the IEEE International Conference on Communications. Budapest, Hungary, 2013: 1974-1979

[41] Khurshid A, Zhou W, Caesar M, et al. VeriFlow: Verifying network-wide invariants in real time//Proceedings of the Workshop on Hot Topics in Software Defined Networks. San Jose, USA, 2012: 49-54

[42] Kazemian P, Chang M, Zeng H, et al. Real time network policy checking using header space analysis//Proceedings of the USENIX Conference on Networked Systems Design and Implementation. Lombard, USA, 2013: 99-112

[43] Kazemian P, Varghese G, Mckeown N. Header space analysis: Static checking for networks//Proceedings of the USENIX Conference on Networked Systems Design and Implementation. San Jose, USA, 2012: 9-9

[44] Zhou W, Jin D, Croft J, et al. Enforcing generalized consistency properties in software-defined networks//Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation. Oakland, USA, 2015: 73-85

[45] Hu H, Han W, Ahn G J, et al. Flowguard: Building robust firewalls for software-defined networks//Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. Chicago, USA, 2014: 97-102

[46] Porras P, Shin S, Yegneswaran V, et al. A security enforcement kernel for OpenFlow networks//Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks. New York, USA, 2012: 121-126

[47] Porras P, Cheung S, Fong M, et al. Securing the software defined network control layer//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2015: 44-57

[48] Shin S, Song Y, Lee T, et al. Rosemary: A robust, secure, and high-performance network operating system//Proceedings of the ACM Sigsac Conference on Computer & Communications Security. Scottsdale, Arizona, USA, 2014: 73-83

[49] Shin S, Porras P, Yegneswaran V, et al. FRESCO: Modular composable security services for software defined networks//Proceedings of the Network & Distributed Security Symposium. San Diego, USA, 2013: 1-16

[50] Yu M, Jose L, Miao R. Software defined traffic measurement with opensketch//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation. Lombard, USA, 2013: 29-42

[51] Fayaz S K, Tobioka Y, Sekar V, Bailey M. Bohatei: Flexible and elastic DDoS defense//Proceedings of the 24th USENIX Security Symposium. Austin, USA, 2015: 817-832

[52] Xu Y, Liu Y. DDoS attack detection under SDN context//Proceedings of the IEEE Conference on Computer and Communications. San Francisco, USA, 2016: 1-9

WANG Shou-Yi, born in 1990, M. S. candidate. His research interests include network security, and software-define networking.

LI Qi, born in 1979, Ph. D., associate professor. His research interests include network and system security, particularly in Internet and cloud security, mobile security, and big data security.

ZHANG Yun, born in 1993, M. S. candidate. Her research interests focus on software-define networking.

Background

The Internet is the important global information infrastructure in the world, which plays a key role in economic development, social development, and science and technology innovation. Since it was designed more than 30 years ago and the design did not have any security considerations, it suffers massive attacks, in particular traffic hijacking attacks. In particular, Snowden pointed out several recent attacks to different nations in the Internet by the US government in 2013, which evinces that network attacks already endangered public and state security. Unfortunately, the current Internet infrastructure cannot effectively detect and defend against such attacks. Software-Defined Networking (SDN) is a promising technology to address the security problems in the current Internet. However, it is still unable to effectively detect and verify the packet forwarding anomaly problem, e.g., packet dropping or hijacking. Specially, the recent efforts in packet forwarding in SDN incur significant computation and communication overhead. Therefore, it is necessary to revisit the packet forwarding anomaly problem, and develop an efficient and effective approach to detecting the forwarding anomalies in realtime.

The goal of this project is to investigate and address the key issues of an attack-tolerant network. In particular, it investigates the requirements of packet forwarding verification in the Internet. We leverage the emerging SDN technology to implement efficient packet forwarding verification, which can efficiently defend against various types of attacks that incur packet forwarding anomalies, e.g., traffic dropping attacks, packet injection attacks, and traffic hijacking attacks. We develop a software-defined packet forwarding architecture that efficiently utilizes the benefits of SDN and detects forwarding anomalies in realtime. Moreover, we build countermeasures to the state-of-the-art attacks on traffic hijacking attacks and verify packet forwarding. This research is supported in part by the National Key Research and Development Program of China under Grant No. 2016YFB0800102 and the National Natural Science Foundation of China under Grant No. 61572278.