

東南大學

硕士学位论文

基于软件定义网络的 DDoS 防护系统

专业名称: 信息与通信工程（信息安全）

研究生姓名: 杨慧文

导师姓名: 宋宇波

DDOS DEFENSE SYSTEM BASED ON SOFTWARE-DEFINED NETWORKING

A Thesis Submitted to

Southeast University

For the Academic Degree of Master of Engineering

BY

YANG Hui-wen

Supervised by

A.Prof SONG Yu-bo

School of Information Science and Engineering

Southeast University

March 2018

东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____日期：_____

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名：_____导师签名：_____日期：_____

摘要

目前, 分布式拒绝服务攻击 (DDoS, Distributed Denial-of-Service Attack) 已成为网络安全的最大威胁之一, 且影响态势日益严重。传统的对抗方式基于静态网络架构, 采用入侵检测、流量过滤和多重验证等方法。这些方法存在明显的缺陷: 难以实现全网统一调度、消耗大量资源并且易导致网络设备愈加繁重。因此, 近年来基于动态网络架构进行 DDoS 防护成为研究的热点。其中, 软件定义网络 (SDN, Software-Defined Networking) 作为一种新型动态网络体系, 其数控分离、集中控制与动态可编程等特性颠覆了现有的网络架构, 为对抗 DDoS 攻击提供了新的思路。现有基于 SDN 的 DDoS 防护方案存在以下问题: 系统开销与检测周期难以权衡、交换机资源浪费以及网络可用性不强等。针对上述问题, 本文提出触发检测和深度检测相结合的 DDoS 检测方案, 以及路由动态调整和溯源主动阻断相结合的 DDoS 防御方案, 并在 Mininet 平台上实现了基于 SDN 的 DDoS 攻击防护系统。主要工作和创新点如下:

1. 针对现有检测方案中检测周期过小将导致系统开销大的问题, 本文提出了一种触发检测和深度检测相结合的联合检测方案。该方案首先通过触发检测, 对全网流量特征的时间序列进行统计特性跟踪, 实现异常流量的粗粒度检测; 接着引入时间窗特征, 采用机器学习中的 AdaBoost 集成学习算法对异常流量进行细粒度的深度检测。经实验表明, AdaBoost 算法的检测精确度优于其他分类算法, 可达 97.7%, 同时该联合检测方案的具有开销小、低漏警率的优点。

2. 针对现有异常检测算法中检测实时性差、鲁棒性不强的问题, 本文提出了一种改进的累积和算法 (SMNA-CUSUM, Sliding-window based Multi-dimension Non-parametric CUSUM algorithm with Adaptive threshold), 利用 CUSUM 算法对样本偏差灵敏度高、计算开销小的优势, 通过实时跟踪多维度序列并基于其统计特性的累积偏差进行异常检测。该改进算法引入滑动窗口来构造特征序列, 每当新样本到达时均结合历史信息实时做出判决; 引入动态阈值针对不同漏警率和虚警率需求进行自适应调整, 提升算法的灵活性和对不同序列的鲁棒性; 引入多维度的特征来降低检测的漏警率。

3. 针对现有防御方案中阻断响应延时、网络可用性不强的问题, 本文提出了一种基于路由动态调整和溯源主动阻断相结合的防御方案。该方案首先采用改进的 K 最短路径算法, 实现基于带宽的最优路由, 使得整个网络可以动态地根据流量变化来进行路由, 得以在面临攻击时充分利用网络资源来分散和吸收攻击流量; 同时基于主机行为进行攻击溯源, 并采用基于动态流表的报文实时过滤方案实施主动阻断。

4. 针对现有报文过滤方案中阻断后残留大量无效流表占用交换机资源的问题, 本文提出了一种基于流表保护机制的报文过滤方案。该方案制定针对攻击源的动态流表来实时过滤攻击报文, 并为流表项定义较短的闲置超时时间, 自动删除无效流表以释放资源。同时, 提出全局白名单机制来保护合法主机被误判和阻断, 增强系统的灵活性与网络的可用性。实验表明, 该防御方案对攻击报文进行了成功的过滤, 且过滤后删除无效流表约 22.7%, 节省了交换机的资源。

5. 为了验证本文所提方案的有效性, 本文在 Mininet 平台上实现了 DDoS 防护系统, 并搭建 Fat-tree 结构的网络拓扑, 基于 Iperf 和 Scapy 分别实现了真实流量和 DDoS 攻击流量的模拟, 对防护系统进行测试和评估。结果表明, 该防护系统具有系统开销小、检测准确率高的特性, 实用价值较强。

关键词: 分布式拒绝服务攻击, 软件定义网络, 异常检测, 攻击溯源, 攻击阻断

Abstract

At present, Distributed Denial-of-Service (DDoS) Attack, with increasingly serious impact, has become one of the biggest threats to network security. Traditional defense mechanisms are based on static network and adopt methods such as instruction detection, traffic filtering and multiple authentication. These methods have obvious drawbacks as follows: It is difficult to achieve unified scheduling across the entire network, consumes lots of resources and it easily leads to increasingly heavier network equipment. Therefore, DDoS protection based on dynamic network has become a research hotspot in recent years. Among them, Software-Defined Networking (SDN) is a new typical dynamic network and provides inspirations to the defense of DDoS. The existing SDN-based DDoS protection solution has the following problems: it is difficult to weigh between the system overhead and detection period, wastes resource of switches in abundance and makes the network with low availability. Aiming at these problems, a DDoS detection scheme combined with trigger detection and in-depth detection, as well as a DDoS mitigation scheme combined with dynamic routing adjustment and active blocking are proposed. Finally, a DDoS defense system based on SDN has been implemented on the Mininet platform. The main work and innovations are as follows:

1. A DDoS detection scheme combined with trigger detection and in-depth detection is proposed to solve the problem on the difficulty in tradeoff between detection period and system overhead. Firstly the trigger detection are implemented to track the time series of characteristics of the network traffic to achieve coarse-grained detection of abnormal traffic. Then taking advantage of time-based features, the in-depth detection are implemented to adapt the AdaBoost algorithm to preform fine-grained detection of abnormal traffic. Experiments revealed that the detection accuracy of AdaBoost algorithm is better than other classification algorithms, which reaches 97.7%. What's more, the DDoS detection scheme has the advantage of low overhead and low alarm rate.

2. An improved CUSUM algorithm named SMNA-CUSUM is proposed to solve the problem of poor real-time performance and poor robustness in existing anomaly detection algorithms. Taking advantages of high sensitivity and low overhead of CUSUM, SMNA-CUSUM algorithm performs anomaly detection by tracking multi-dimensional sequences in real time and based on the cumulative deviation of statistical characteristics. A sliding-window mechanism is introduced to construct feature sequences and an adaptive threshold is introduced for adjustment between missed alarm rates and false alarm rates.

3. A DDoS mitigation scheme combined with dynamic routing adjustment and active blocking is proposed to solve the problem of low network availability and delay in blocking. An improved K-shortest path algorithm is introduced to perform bandwidth-based routing, which enables the entire network to be dynamically routed based on traffic changes and to absorb malicious traffic in the face of attacks. In addition, an attack tracing method based on host behavior is introduced, followed by a real-time filtering solution based on dynamic flow table, to perform the blocking process.

4. A packet filtering scheme based on flow protection mechanism is proposed in the mitigation scheme

to solve the problem that the switches are occupied by a large number of invalid flows. The short idle timeout is set to enable the automatic deletion of invalid flows. In addition, a global whitelist mechanism is introduced to protect legitimate hosts from misjudgment. Experiments revealed that the proposed mitigation scheme has successfully filtered the malicious packets. What's more, 22.7% of the flows have been deleted, which saved resources of switches.

5. To verify the effectiveness of the proposed schemes, a DDoS defense system based on SDN has been implemented on the Mininet platform. A Fat-tree topology network is built and the simulation of normal traffic and DDoS attack traffic is performed to evaluate the defense system. Experiments revealed that the defense system has the characteristics of low system overhead, high detection accuracy, and strong practical value.

Keywords: DDoS, SDN, anomaly detection, attack traceback, attack blocking

目录

摘要.....	I
Abstract	III
目录.....	V
插图目录.....	IX
表格目录.....	XI
第一章 绪论.....	1
1.1 研究背景及意义.....	1
1.2 国内外研究现状.....	2
1.2.1 基于 SDN 的 DDoS 检测技术.....	2
1.2.2 基于 SDN 的 DDoS 防御技术.....	3
1.2.3 小结.....	4
1.3 本文主要内容与章节安排.....	5
1.3.1 本文的主要工作.....	5
1.3.2 本文的章节安排.....	5
第二章 SDN 架构和 DDoS 攻防理论基础.....	7
2.1 引言.....	7
2.2 SDN 技术.....	7
2.2.1 SDN 架构.....	7
2.2.2 OpenFlow 协议.....	8
2.3 DDoS 攻击.....	11
2.3.1 DDoS 攻击原理.....	11
2.3.2 DDoS 防护体系.....	11
2.3.3 SDN 中的 DDoS 攻击.....	12
2.4 攻击防护的相关算法.....	13
2.4.1 基于 CUSUM 控制图的变点检测.....	13
2.4.2 基于机器学习的异常检测.....	14
2.5 本章小结.....	16
第三章 基于 SDN 的 DDoS 检测方案.....	17
3.1 引言.....	17
3.2 检测系统架构.....	18
3.3 触发检测模块.....	18
3.3.1 流表信息采集.....	18
3.3.2 特征提取和分析.....	19
3.3.3 多维特征序列的构造.....	22

3.3.4 基于 SMNA-CUSUM 算法的异常检测	24
3.4 深度检测模块	29
3.4.1 信息采集和特征提取	29
3.4.2 基于滑动时间窗的特征构造	30
3.4.3 基于 AdaBoost 算法的攻击检测	31
3.5 本章小结	36
第四章 基于 SDN 的 DDoS 防御方案	37
4.1 引言	37
4.2 防御系统架构	38
4.3 动态路由模块	38
4.3.1 拓扑发现	39
4.3.2 流量监控和带宽获取	40
4.3.3 基于优化的 K 最短路径算法的动态路由	41
4.4 攻击阻断模块	44
4.4.1 基于主机行为的攻击溯源	44
4.4.2 基于动态流表的报文实时过滤	46
4.5 本章小结	50
第五章 基于 SDN 的 DDoS 防护系统实现与测试	51
5.1 引言	51
5.2 系统设计和实现	51
5.2.1 SDN 网络搭建	51
5.2.2 系统架构	53
5.2.3 模块设计	53
5.2.4 模块间通信	54
5.3 实验环境搭建	55
5.3.1 网络拓扑的搭建	55
5.3.2 真实网络流量的模拟	56
5.3.3 DDoS 攻击的模拟	59
5.4 实验结果和分析	60
5.4.1 系统评估标准	60
5.4.2 检测方案结果和分析	61
5.4.3 防御方案结果和分析	65
5.5 本章小结	67
第六章 总结与展望	69
6.1 本文工作总结	69
6.2 未来研究展望	70
参考文献	71

致谢.....	75
作者简介.....	76

插图目录

图 2.1	SDN 网络架构.....	8
图 2.2	OpenFlow 交换机模型.....	9
图 2.3	流表项结构.....	9
图 2.4	流表匹配过程.....	10
图 2.5	OpenFlow 通信流程.....	10
图 2.6	DDoS 攻击示意图.....	11
图 2.7	针对 SDN 的 DDoS 攻击.....	13
图 2.8	机器学习分类.....	14
图 2.9	集成学习的一般结构.....	15
图 2.10	boosting 框架.....	16
图 2.11	bagging 框架.....	16
图 3.1	攻击检测系统架构.....	18
图 3.2	DDoS 攻击时五个特征量的变化情况.....	22
图 3.3	特征序列的来源.....	22
图 3.4	基于 packet-in 消息的特征提取流程.....	23
图 3.5	构造多维特征序列的滑动窗口机制.....	24
图 3.6	SMNA-CUSUM 算法流程图.....	26
图 3.7	不同 h 值对检测结果的影响.....	28
图 3.8	t=3 时的时间窗.....	31
图 3.9	t=4 时的时间窗.....	31
图 3.10	基于 AdaBoost 的攻击检测.....	34
图 4.1	防御系统架构.....	38
图 4.2	链路发现和主机发现示例拓扑.....	39
图 4.3	流量监控.....	40
图 4.4	K 最短路径示例拓扑.....	42
图 4.5	SDN 中的动态路由过程.....	43
图 4.6	攻击阻断模块.....	44
图 4.7	报文过滤流程.....	47
图 4.8	针对攻击者的动态流表项.....	47
图 4.9	结合流表保护机制的报文过滤方案.....	48
图 4.10	结合全局动态白名单机制的攻击阻断流程.....	49
图 5.1	OVS 内部结构.....	52
图 5.2	部署了 DDoS 防护系统的 SDN 网络整体架构.....	53
图 5.3	防护系统流程图.....	54
图 5.4	胖树网络拓扑 (k=2).....	56
图 5.5	真实网络流量的模拟.....	58
图 5.6	服务器内的半连接.....	60
图 5.7	合法主机发起请求连接失败.....	60
图 5.8	攻击检测算法的归一化混淆矩阵.....	62
图 5.9	攻击检测算法的 ROC 曲线.....	62
图 5.10	AdaBoost 算法与 SMNA-CUSUM 算法的 CPU 占用率.....	63
图 5.11	基于滑动时间窗的特征构造对判决时间的影响.....	64
图 5.12	不同分类算法的精确率对比.....	64

图 5.13	初始状态下的主机之间的路径	65
图 5.14	主机 h1 到 h4 的路径	65
图 5.15	主机 h2 到 h4 的路径	65
图 5.16	攻击发生前交换机 e7 中的流表项	66
图 5.17	攻击发生后交换机 e7 中的流表项	66
图 5.18	服务恢复后合法主机成功连接	66
图 5.19	网络流量特征的变化	67

表格目录

表 3.1	流表项计数器信息	19
表 3.2	表征网络流量变化的典型特征	21
表 3.3	触发检测模块参数	27
表 3.4	不同的 h 值对虚警率和漏警率的影响	28
表 3.5	表征网络流量变化的特征	30
表 3.6	AdaBoost 算法	32
表 3.7	深度检测模块的参数	35
表 3.8	TP 、 FN 、 FP 、 TN 的定义	35
表 3.9	不同分类算法的对比	36
表 4.1	交换机端口统计信息	41
表 4.2	流表项统计信息	41
表 4.3	基于源地址的 4 元组	45
表 4.4	基于目的地址的 4 元组	45
表 5.1	Ryu 重要组件与功能	52
表 5.2	四种实验类型	61
表 5.3	四种实验的检测精确度	61
表 5.4	检测方案准确率对比	62
表 5.5	交换机内的流表项数量变化	66

第一章 绪论

1.1 研究背景及意义

当前，DDoS 攻击（DDoS, Distributed Denial-of-Service Attack）已成为网络安全的最大威胁之一，其破坏力强、实施门槛低且缺乏简单可行的对抗措施。纵观 2016 全年以及 2017 年上半年 DDoS 攻击的威胁态势，不难发现以下趋势。

(1) 攻击强度增大：2016 年我国境内发生 DDoS 攻击约 22 万次，与去年同期相比增长 18.6%；攻击总流量约 36 万 TBytes，同比增长 25%。大流量攻击事件较 2015 年有明显增多，攻击峰值在 50-100Gbps 的攻击总数在 2016 第四季度同比增长 172.6%。按攻击流量统计，SYN Flood 和 UDP Flood 是流量最大的两种攻击类型，分别占比达 49% 和 30%。按攻击次数统计，全球 Top 3 的 DDoS 攻击均为反射类型攻击，依次为 NTP Reflection Flood、CHARGEN Reflection Flood 和 SSDP Reflection Flood，合计占比 80% 以上。

(2) 攻击成本降低：随着黑产利益链的不断成熟，DDoS-for-Hire 的产业化，任何恶意者都可以通过购买 DDoS 攻击服务发起恶意攻击。同时，由于物联网设备大幅增长，物联网僵尸网络急剧扩张，众多的物联网设备成为被感染和被控制的 Bot 端。基于物联网的僵尸网络恶意程序不仅有耳熟能详、感染范围极大的 Mirai，还有 AES.DoS、Luabot、Hajime、台风 DDoS 等等。

(3) 攻击目标广泛：各行各业都面临 DDoS 攻击的威胁。长期以来，政府、金融和游戏是 DDoS 攻击的重灾区，而随着攻击难度的降低以及各行各业与互联网拥抱，其他领域如互联网电商、教育机构、媒体行业等等也面临着极大的威胁。

由上述威胁态势看出，如何在对抗 DDoS 攻击这一问题上取得突破性进展是当前网络亟待解决的问题。

对抗 DDoS 攻击的传统方式包括入侵检测、流量过滤和多重验证等等，上述方式存在如下缺陷：大多以在现有网络结构中打补丁并不断更新补丁的方式进行防护，不但使得网络设备承载的功能越来越繁重，而且难以做到统一的调度、配置和融合。目前大多数企业缓解 DDoS 攻击时都依赖基于云的技术来增加服务能力，从而使云吸收来自 DDoS 的负担。但是，这些解决方案成本高昂，同时也引入了隐私问题，因为它们要求公司将网络流量重定向到云的提供商。此外，即使拥有大量资源，大多数情况下依旧难以满足低成本和高灵活性的业务需求。从根本上讲，传统网络是對抗 DDoS 攻击的瓶颈，由于其网络架构是静态的，在面临新的挑战如访问量的陡增时，需要大量冗余资源或备用资源来及时调整。

近年来，软件定义网络（SDN, Software-Defined Networking）理念的出现，为设计、搭建以及管理网络提供了一种全新的思路。它是一种新型网络架构，给传统网络带来的最大改变是网络的可编程性和开放性。它也是一个涵盖多种网络技术的综合性术语，旨在使得网络像现代数据中心的虚拟化服务器和存储基础设施一样具有高度的灵活性和敏捷性。SDN 的目标是赋予网络快速响应不断变化的业务需求的能力。在 SDN 中，网络管理员可以通过控制器中的集中控制台实时检测及管理流量，无需触及每一个的交换机；可以将服务提供到网络所需的任何位置，而无需考虑服务器或其他

硬件组件连接到了什么设备。在传统网络需要使用专用设备（如防火墙或链路负载均衡器）时，SDN 只需要部署应用程序，命令控制器来管理数据平面的行为。SDN 的出现为检测和防御 DDoS 攻击带来了新的思路：其全局视角和集中控制的特性为实时监控全网和各节点的流量情况提供了条件，其数控分离和动态可编程的特性为网络中报文的深度分析、流量的监控管理以及实时更新网络转发规则提供了基础。因此，基于 SDN 的 DDoS 检测和防御技术成为研究热点。

1.2 国内外研究现状

目前，国内外学者对 SDN 中 DDoS 攻击的研究主要集中在攻击检测技术和攻击防御技术这两方面，接下来分别对攻击检测和攻击防御的研究现状进行分析，最后进行小结，分析现有研究存在的不足。

1.2.1 基于 SDN 的 DDoS 检测技术

在 SDN 环境中进行 DDoS 攻击检测正处于研究和探索阶段，大多数研究是在借鉴了传统网络中攻击检测方案的基础上，进行一系列改进以适应 SDN 的新特性。其核心思想是利用 SDN 具有全网视角、集中控制和数控分离的特性，通过采集网络信息、提取特征、分析特征以及选用合适的检测算法等一系列步骤，实现攻击的检测。不同检测方案，其采集信息的方式、提取特征的方法、选用的检测算法以及检测算法所处节点的位置各有不同，因此各有利弊，也都存在提升的可能。

首先，根据检测算法在网络中所处的位置，可将不同的检测方案分为部署在控制器中的检测和部署在交换机中的检测两类方案。文献^[1-9]研究的是部署在交换机的检测方案：AVANT-GUARD^[1]是一个通过给交换机赋能而实现对抗 DDoS 攻击和端口扫描的安全框架，它在交换机上引入了两个模块：连接迁移和启动触发器，连接迁移模块代理 TCP SYN 类型的请求并对它们进行分类，如果请求被认为是合法的，则被授权和迁移到真正的目标；启动触发模块感知变化并触发相应事件，实现自动化安装流表规则并缩短响应时间。这种机制最显著的副作用是性能损失较大，由于它采用了连接迁移，每个流程都需要进行分类，另外，该机制只能防御 TCP SYN 这种类型的 DDoS 攻击。

部署在交换机中的方案由于需要修改交换机的底层实现逻辑因而代价较大且可拓展性不强。因此，本文主要研究部署在控制器中的检测方案，根据方案中所使用检测算法原理的不同，将其分为基于传统统计学算法的检测方案以及基于机器学习算法的检测方案等两大类。

对基于传统统计学检测方案的研究^[10-18]有以下成果：Dao^[16]等人提出通过分析每个主机发出数据包的频率来判断该主机属于合法主机还是僵尸主机的方法，其依据是合法主机与僵尸主机的发包频率有显著的差异，但该特征过于单一，误判率较高。很多文献使用熵^[12-15]作为表征网络流量的关键特征。文献^[14]依据网络遭受 DDoS 攻击时其目的 IP 地址分布的随机性将减小的现象，提出使用目的 IP 地址的熵值来衡量目的地址分布的随机性，进一步地，通过实验的方式确定熵的合理阈值与观测的时间窗长度，在实际网络环境中可以短时间内发现熵值的异常。Giotis^[12]等人提出通过从全网数据包中提取四个特征（源 IP 地址、目的 IP 地址、源端口号和目的端口号）并计算其熵值的方式来表征网络的状态，通过模拟实验设定四个熵值的阈值以检测异常，同时该文献使用 sFlow^[19]工具来收集网络信息以减轻控制器与交换机之间的通信压力。对于特征量的异常检测，除了通过观测模拟攻击的实验来进行人工设定阈值这种方式，文献^[10, 11, 17, 18]使用基于累积和（CUSUM, Cumulative Sum）

控制图的方案对特征量的变化进行实时跟踪，建立模型并自动探测到异常值。Conti^[10]等人提出了一种改进的 CUSUM 算法，实现了随着网络的变化而自动调节阈值的功能，并用改进算法来检测网络中数据包的数目是否产生异常，最后基于 CAIDA 和 DARPA 两个数据集进行仿真实验以验证算法的可行性。舒远仲^[118]等人提出条件熵的概念并使用非参数的 CUSUM 算法来检测异常，与香农熵相比，其误报率从 26.7% 下降至 8.4%。

然而基于传统统计学的检测方案具有以下缺点：（1）可行性不强：检测方案依赖于前期对同一个网络进行模拟攻击并观测特征值来设定阈值，且设定合理阈值的难度系数较大；（2）准确性不够高：由于观测特征数量有限且其变化趋势与实验模拟的情况相关度较大，难以建立全方位的流量模型，误报率和漏报率均有待降低；（3）可拓展性不够强：完成了参数与阈值设定的方案难以直接移植到新的网络。

为了解决这些问题，文献^[20-31]将机器学习技术应用于检测算法，包括支持向量机（SVM, Support Vector Machine）、逻辑回归（LR, Logistic Regression）、K-近邻（KNN, k-Nearest Neighbor）、朴素贝叶斯（NB, Naive Bayes）、神经网络等算法。Braga^[21]分析了与 DDoS 攻击相关的统计信息，提出基于流的特征“六元组”，使用自组织映射（SOM, Self-Organizing Map）算法根据“六元组”识别出恶意流量和正常流量，实现了一种轻量级的检测方法，但 SOM 算法收敛较慢，训练时间较长。文献^[31]则提取了网络中数据包的几个特征：响应时间、源地址和目的地址，分别使用朴素贝叶斯算法、KNN 算法、K-means 算法等不同算法对特征进行分类以识别异常，但该文献对不同算法的不同性能分析不足，使用现有数据集进行仿真，缺少实际 SDN 网络的仿真环境。肖甫^[26]等人在全网流量中以交换机为单位提取了 5 个关键特征：数据包中位数、字节中位数、对流比、端口增速和源 IP 地址增速，采用 KNN 分类算法做异常检测，并与 SVM 算法进行对比，获得更高的检测率以及更低的误报率。Atlantic^[29]是一个部署在 SDN 网络中以进行流量的异常检测、分类以及攻击缓解的框架，其流量识别模块使用 K-means 算法进行聚类以及 SVM 算法进行分类。随着深度学习的发展和流行，不少文献^[32-35]开始尝试使用深度学习算法来进行攻击检测以获得更高的准确率。Tang^[33]等人利用五层神经网络来训练模型，但由于其特征选择不佳，导致准确率仅有 75%，另外，作者使用同样的特征将其输入传统机器学习算法，结果表明神经网络算法的准确率高于其他算法。Niyaz^[32]等人根据流量所属的不同协议提取不同的特征，并使用栈式自动编码器的算法，对流量进行分类，其准确率达到 95.65%。

1.2.2 基于 SDN 的 DDoS 防御技术

对 SDN 中 DDoS 防御技术的研究大多是建立在攻击检测的基础之上，其核心思想是利用 SDN 动态可编程的特性，在检测到攻击后立即做出响应即控制器向交换机下发相应的流表以缓解攻击。常见的响应措施^[34, 36-39]包括丢弃数据包、流速限制、端口阻断以及重定向等等。

DefenseFlow^[36]是早期对抗 DDoS 攻击的 SDN 应用，但它不是一个纯 SDN 解决方案，因为该系统的实现需要硬件的支持，因此实施难度较大且代价较高。更多的研究将重点放在纯 SDN 解决方案领域。在 Giotis^[12]等人提出的方案中，使用 IP 地址和端口号这两项去匹配交换机中接收到的数据包，成功匹配的数据包意味着其属于攻击流量，因此对其执行丢弃操作；另外，该文献还提出白名单的机制以降低误检率。文献^[40]通过对具有攻击特征的流量进行速率限制以达到阻断攻击的效果。在文

献^[25]中, 被识别为攻击流的数据包将经历转发模块、丢弃模块和更新模块等三个模块, 保障系统在完成丢弃攻击流量的同时, 记录下攻击相关的信息以便进一步分析。文献^[12, 28, 29, 38, 40]均证明, 在 SDN 中通过控制器向交换机下发流表以实现实时转发或阻断攻击流量的方案可行性较高且效果明显, 但在完成攻击阻断后, 大量无效流表被存留在交换机中, 这将消耗交换机的内存资源甚至导致流表溢出现象。因此, Cui^[34]等人提出, 在完成攻击阻断后, 通过控制器向交换机下发流表删除的命令来删除与阻断攻击相关的流表项, 实现交换机内存的释放, 但该文献并未说明具体的删除方式以及删除时间。

1.2.3 小结

综上所述, 虽然国内外对基于 SDN 的 DDoS 攻击检测和防御技术的研究已经取得了很多重要成果, 但是仍然存在以下不足:

- (1) 现有检测方案中, 由于系统开销与检测延时之间的矛盾难以权衡而导致检测周期难以确定, 多数检测方案采用轮询的方式来采集网络信息, 基于这些信息提取特征并进行分类。作为检测方案的基础, 采集到的网络信息十分重要, 而轮询的周期决定了信息的粒度, 因此如何选择轮询周期成为关键问题, 周期太长将导致准确率低且检测延时过长, 而过短的周期则导致系统开销太大, 造成控制器的负担甚至引发其他问题。
- (2) 现有检测方案中, 特征提取的方法有待改善。一方面, 多数研究提取特征时只关注当前的网络状态, 忽视了历史状态, 而网络遭受 DDoS 攻击时其流量的变化与时间相关性较大, 该时间特性应当被表征为合适的特征量; 另一方面, 由于 SDN 中的检测方案是在借鉴了传统网络中的基础上改进的, 多数研究在提取特征时没有结合 SDN 中基于流来转发数据包以及基于流来统计信息的特性。
- (3) 现有防御方案中, 存在大量无效流表占用交换机资源的问题。许多研究已经证明, 在 SDN 中通过控制器向交换机下发流表以实现实时阻断攻击流量的方案可行性较高且效果明显, 但其中存在一个严重的问题, 即交换机中会遗留大量因为攻击而产生的失效流表。而交换机作为 SDN 中数据平面内的一个轻量级的转发节点, 其内存有限, 无效的流表将造成内存的消耗甚至导致交换机流表溢出。因此, 如何处理无效流表是防御方案中的一个重要问题。
- (4) 现有防御方案中, 如何降低对合法流量的影响、提升网络的可用性这一问题并未得到重视和研究。多数方案采用直接过滤检测到的攻击源报文的方式进行攻击阻断, 但由于检测时可能发生误检, 导致合法请求受到影响。SDN 网络目前常用于数据中心, 作为向外提供服务和资源的基础设施, 其可用性尤为重要。
- (5) 现有研究缺乏真实的 SDN 应用环境对系统进行验证。很多文献^[22-24, 30]在测试其提出的算法时, 使用现有数据集, 如 KDD-99、CAIDA 和 DARPA^[41]等进行离线分析, 但由于数据集时间久远因而存在大量无效数据, 同时数据集并不是基于 SDN 网络产生的, 适用性不强。另外, 虽然文献^[10, 21, 26-29, 31, 33, 35]搭建了 SDN 的环境并进行了仿真, 但其设计的网络拓扑结构过于简单, 并不是 SDN 应用场景中的网络拓扑。因此, 设计合适的网络拓扑并搭建 SDN 网络环境来进行整个系统的验证和评估是现有研究中所欠缺的。

本文基于上述问题进行深入研究, 提出新的解决方案来检测和防御 DDoS 攻击, 并基于所提的

方案设计攻击防护系统，进一步地，搭建仿真环境对系统进行测试和评估。

1.3 本文主要内容与章节安排

1.3.1 本文的主要工作

本文研究了软件定义网络中 DDoS 攻击的防护技术。首先分析了现有的基于 SDN 的 DDoS 检测和防御方案的研究成果与不足之处；在此基础上，设计了一个完整的 DDoS 防护方案，包括 DDoS 检测方案以及 DDoS 防御方案；最后，在 Mininet 平台上实现了整个 DDoS 防护系统，并通过设计仿真实验，完成了对系统的测试和评估。

本文完成的主要工作包括：

- (1) 分析了基于 SDN 的 DDoS 检测和防御方案的研究背景，概述了国内外的研究进展以及存在的问题。
- (2) 针对现有检测方案中检测周期过小将导致系统开销大的问题，提出了一种触发检测和深度检测相结合的联合检测方案。该方案首先通过触发检测，对全网流量特征的时间序列进行统计特性跟踪，实现异常流量的粗粒度检测；接着引入时间窗特征，采用机器学习算法对异常流量进行细粒度的深度检测。整个方案在保障较高的精确率的同时降低了系统开销。
- (3) 针对现有异常检测算法中检测实时性差、鲁棒性不强的问题，提出了一种改进的累积和算法 (SMNA-CUSUM, Sliding-window based Multi-dimension Non-parametric CUSUM algorithm with Adaptive threshold)，通过实时跟踪多维度序列并基于其统计特性的累积偏差进行异常检测。在对触发检测进行需求分析的基础上，引入滑动窗口来构造特征序列，引入动态阈值针对不同漏警率和虚警率的需求进行自适应调整，引入多维度的特征来降低检测的漏警率。
- (4) 针对现有防御方案中阻断响应延时、网络可用性不强的问题，提出了一种基于路由动态调整和溯源主动阻断相结合的防御方案。该方案首先采用改进的 K 最短路径算法，实现基于带宽的最优路由，使得整个网络可以动态地根据流量变化来进行路由，得以在面临攻击时充分利用网络资源来分散和吸收攻击流量；同时基于主机行为进行攻击溯源，并采用报文实时过滤方案实施主动阻断。
- (5) 针对现有报文过滤方案中阻断后残留大量无效流表占用交换机资源的问题，提出了一种基于流表保护机制的报文过滤方案。该方案制定针对攻击源的动态流表来实时过滤攻击报文，并为流表项定义较短的闲置超时时间，自动删除无效流表以释放资源。同时，提出全局白名单机制来保护合法主机被误判和阻断，增强系统的灵活性与网络的可用性。
- (6) 完成了基于 SDN 的 DDoS 攻击防护系统的设计与实现，以及系统验证平台的设计与实现。最后在验证平台中进行实验对防护系统进行测试，并从可靠性、复杂度、可用性和可拓展性等几个角度对系统进行评估。

1.3.2 本文的章节安排

基于上述研究内容，本文共分为六章，具体的章节安排如下：

第一章为绪论，首先介绍了 DDoS 攻击的威胁态势以及防护瓶颈，概述了 SDN 的出现为 DDoS

防护带来的新思路,在此基础上分析了现有的基于 SDN 的 DDoS 检测和防御方案的研究成果与不足之处,最后简单介绍了本文的研究内容和章节安排。

第二章研究基于 SDN 的 DDoS 攻击防护技术的理论基础。首先对 SDN 技术进行研究,包括 SDN 的架构、特性以及 OpenFlow 协议和相关标准。接着深入分析了 DDoS 攻击的原理,并从攻击预防、攻击检测和攻击响应等三个方面研究 DDoS 攻击的防护技术。最后,详细介绍本文所涉及的包括变点检测、CUSUM 控制图、机器学习以及集成学习等相关算法的背景和理论知识。

第三章对基于 SDN 的 DDoS 检测技术进行研究,设计了由触发检测和深度检测两个模块构成的 DDoS 检测方案。首先提出了一种改进的累积和算法(SMNA-CUSUM 算法),通过实时跟踪多维度序列并其基于统计特性的累积偏差进行异常检测,该算法应用于触发检测模块,对全网流量实现粗粒度的异常检测;接着引入时间窗特征,采用 AdaBoost 集成学习算法对异常流量进行细粒度的深度检测,最终确定是否存在攻击。

第四章对基于 SDN 的 DDoS 防御技术进行研究,提出了一种基于路由动态调整和溯源主动阻断相结合的防御方案。路由动态调整的目的是实现基于带宽的最优路由,利用网络资源来分散和吸收攻击流量,本章从拓扑发现、流量监控和最优路由制定等过程来介绍该方案。溯源主动阻断包括基于主机行为的攻击溯源模块和基于动态流表的报文实时过滤模块;此外,还提出了基于黑名单的流表保护机制和全局动态白名单机制,与报文实时过滤模块结合,以解决无效流表占用资源的问题,同时增强系统的可用性和灵活性。

第五章是 DDoS 防护系统的实现与测试。基于第三章和第四章所设计的检测方案和防御方案,本章搭建了 SDN 的仿真环境并在该环境中实现了完整的 DDoS 防护系统,在此基础上,完成仿真的设计,包括网络拓扑的选择、数据中心流量的模拟、服务器的搭建以及 DDoS 攻击的模拟;最后在网络中进行不同的实验并分析防护系统各个模块的结果,对系统的可靠性、复杂度、可用性和可拓展性等性能进行评估。

第六章为总结和展望,归纳和总结了全文的内容,分析了 DDoS 防护系统的改进方向以及其他需要进一步完成的工作。

第二章 SDN 架构和 DDoS 攻防理论基础

2.1 引言

DDoS 攻击目前已成为网络安全的最大威胁之一,且其影响态势日益严重。文献^[42-49]对 DDoS 攻击的原理、危害以及防御方案等各方面进行了研究。研究表明,虽然针对 DDoS 攻击的对抗方案不断被提出,但由于 DDoS 攻击方式多样、实施易且防守难,对抗方案存在较多的缺陷,主要体现在:

- (1) 需要复杂的网络设施:多数方案需要在网络中部署复杂的第三方工具来执行攻击检测和防御措施;
- (2) 需要大量的冗余资源:部分方案通过给网络配置大量冗余资源以承载攻击和增强灵活性;
- (3) 检测延时较长且检测准确率较低。从根本上讲,上述缺陷是由于传统网络的静态架构导致的。因此,现有的传统网络是对抗 DDoS 攻击的重要瓶颈。

软件定义网络理念的出现,为设计、搭建以及管理网络提供了一种全新的思路,同时也为 DDoS 攻击的检测和防御带来了新的启示:其全局视角和集中控制的特性为实时监控全网和各节点的流量情况提供了条件,其数控分离和动态可编程的特性为网络中报文的深度分析、流量的监控管理以及实时更新网络转发规则提供了基础。

近年来,包括统计学、信息论和机器学习等多个领域的知识,被应用到 SDN 环境中的 DDoS 攻击检测和防御之中。文献^[10-18]基于传统的统计学理论提出了检测方案,其中文献^[12-15]使用熵作为表征网络流量的关键特征,文献^[10, 11, 17, 18]使用基于 CUSUM 控制图对特征量的变化进行实时跟踪,建立模型并自动探测异常值。文献^[20-31]将机器学习中的分类算法应用于攻击检测,并基于检测结果设计了防御方案,分类算法包括支持向量机、逻辑回归、K 近邻、朴素贝叶斯和神经网络等等。

本章主要介绍基于 SDN 的 DDoS 攻击防护技术的理论基础。SDN 是防护系统的网络环境,因此,本文首先对 SDN 技术进行研究,包括软件定义网络的架构、特性以及 OpenFlow 协议和相关标准。接着,深入分析了 DDoS 攻击的原理,并从攻击预防、攻击检测和攻击响应等三个方面研究 DDoS 攻击的防护技术,此外对 SDN 环境中不同类型的 DDoS 攻击进行简要概述。最后,介绍本文提出的攻击防护系统所涉及的相关算法的背景和理论知识,包括变点检测、CUSUM 控制图以及机器学习算法。

2.2 SDN 技术

2.2.1 SDN 架构

软件定义网络是一种新型网络架构,给传统网络带来的最大改变是网络的可编程性和开放性。一方面,SDN 重构了网络的系统功能,实现了数控分离;另一方面,SDN 对网络资源进行抽象,建立了新的网络抽象模型。

一般来说,SDN 网络架构主要由数据层、控制层、应用层、南向接口、北向接口这五部分组成,如图 2.1 所示。

数据层包括基于软件或硬件实现的通用数据平面设备,通过南向接口(SBI, Southbound Interface)

接收来自控制层的指令，并根据指令完成网络数据处理。控制层具有网络的全局视角，负责管控网络：对下层，通过南向接口与数据层通信，对上层，通过北向接口（NBI，Northbound Interface）向应用层提供基础服务和可编程能力。应用层由开发者通过控制层提供的编程接口对底层设备进行编程，以实现不同的网络功能应用，为开发者提供了丰富的定制开发能力与自主权。传统网络中如负载均衡^[50]、网络配置^[51]、流量工程^[52]、安全控制^[53]等服务均可在应用层上得到实现。

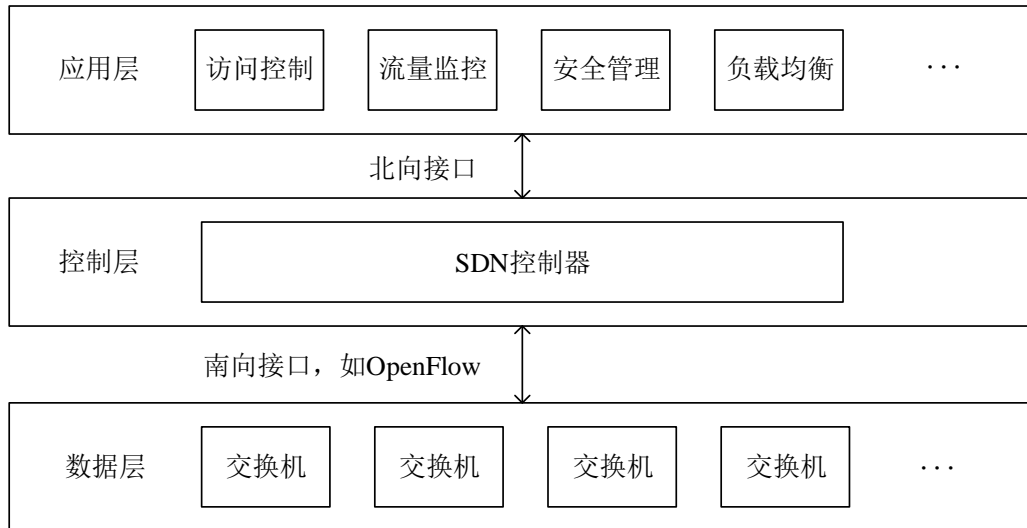


图 2.1 SDN 网络架构

基于上述架构，SDN 有以下几个典型特征。

(1) 数控分离

数控分离即网络控制平面和数据转发平面的解耦合，两个平面之间有一个开放的厂商无关的通用接口：南向接口。数控分离使得控制平面与数据转发平面之间不再互相依赖，两者独立进行体系结构的演进；双方只需要遵循通用接口进行通信即可。数控分离是 SDN 架构区别于传统网络体系的重要标志。

(2) 网络开放可编程

SDN 定义了通用的网络抽象模型，控制平面通过南向接口与数据转发平面进行通信，控制平面作为整个网络的大脑，定义数据转发平面的网络处理行为。用户在控制平面的控制器上直接编程，实现对网络的配置、监控和管理。

(3) 集中控制

集中控制指的是逻辑上对分布式网络状态的集中统一管理。SDN 架构中，控制器担负着收集和管理所有网络状态信息的重任，集中控制为软件编程定义网络功能提供了架构基础。

2.2.2 OpenFlow 协议

OpenFlow 协议于 2008 年在 Nick McKeown 教授等发布的论文^[54]中被提出，自 2009 年 OpenFlow1.0 版本正式发布后，不断演进和拓展，如今成为 SDN 最主流的南向接口协议。它不仅仅是一种南向接口协议，还建立了一种新的数据平面抽象模型，定义了交换机规范。

(1) OpenFlow 交换机

实现了 OpenFlow 规范的交换机称为 OpenFlow 交换机。OpenFlow 交换机是网络策略的参与者，

位于 SDN 架构中的数据层。交换机分为流表和安全通道两部分，其模型见图 2.2。安全通道是用于和控制器通信的安全连接，通道可以建立在 TCP 之上，也可以基于 TLS 加密之后的 Socket 建立；流表用于存放转发规则，数据进入交换机后，在流表中寻找匹配的流表项并执行相应的策略，若无匹配项，则上报控制器以获取进一步指令。与传统的交换机不同，OpenFlow 交换机对于报文的匹配层次可达网络协议的第 4 层，可以匹配到端口，而传统交换机只是两层设备。

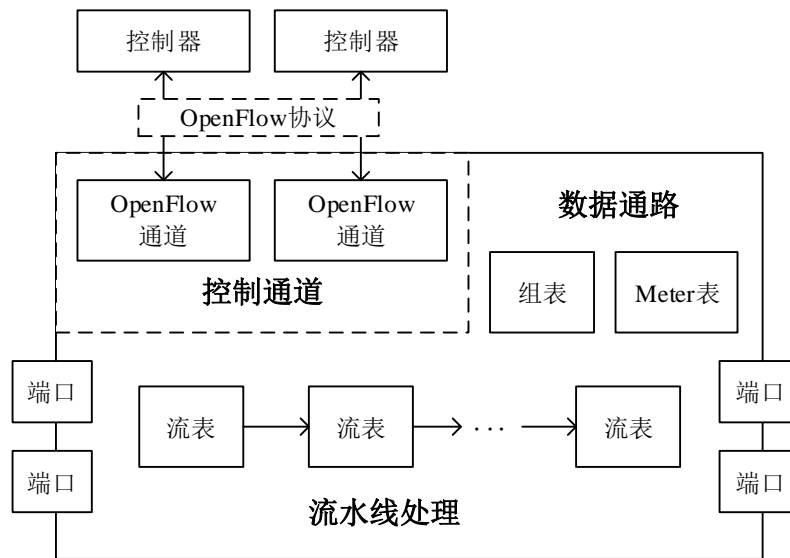


图 2.2 OpenFlow 交换机模型

(2) OpenFlow 表

在 OpenFlow 中，不同的协议，统一使用通用流表处理转发模型（跨网络层级、跨协议）进行处理。流表用于存储流表项，多级流表以流水线的方式处理。

每个流表项由匹配域、指令集和计数器等三个主要部分以及其他部分组成。如图 2.3 所示。

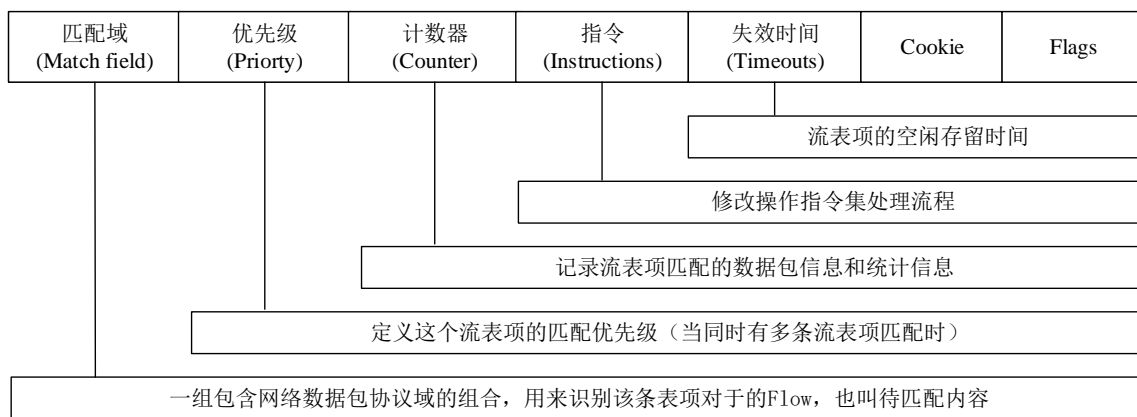


图 2.3 流表项结构

匹配域用以区分不同的数据流，数据包进入交换机后，根据匹配域寻找与之匹配的流表项，匹配成功则执行该流表项对应的指令集，完成数据处理。一个典型的流表匹配过程如图 2.4 所示。

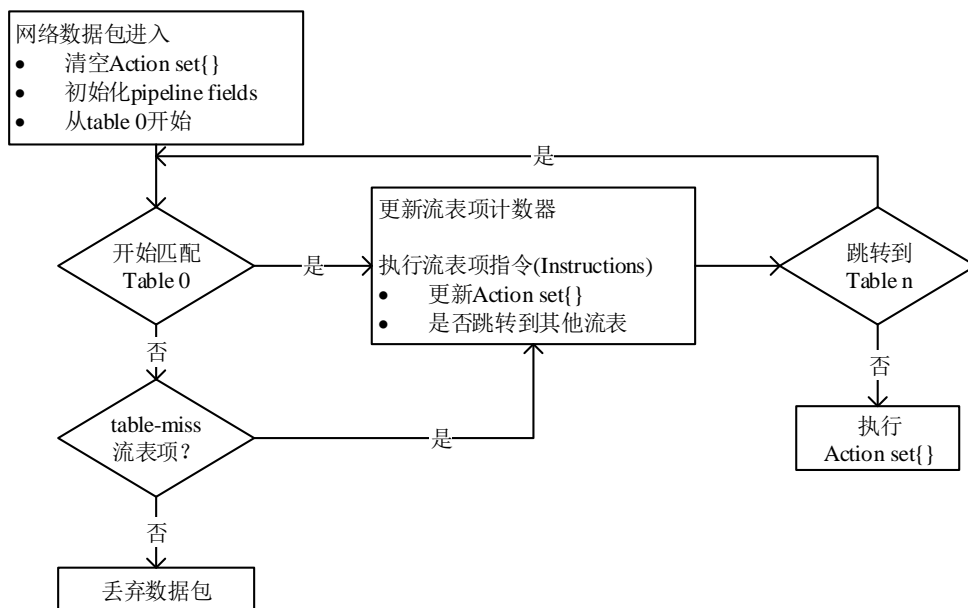


图 2.4 流表匹配过程

计数器则记录了匹配该流表项的数据包数和字节数。流表的计数器模块为控制器获取流表的统计信息以及全网的流量信息提供了基础。

除了流表外，OpenFlow 表还包含组表和计量表，由于不在本文研究范围内，此处不做赘述。

(3) OpenFlow 通道和消息

OpenFlow 通道是交换机和控制器间的通信通道，通道中转发的数据称为 OpenFlow 消息/报文。OpenFlow 消息共有三大类型：Controller-to-Switch (C-S 消息)、Asynchronous (异步消息)、Symmetric (对称消息)。报文细节可参考协议标准。值得注意的是，控制器既可以主动下发消息至交换机，亦可异步接收来自交换机的消息，这为整个网络的实时监控和动态制定策略提供了保障。

(4) OpenFlow 通信流程

图 2.5 描述了交换机与控制器的通信流程：

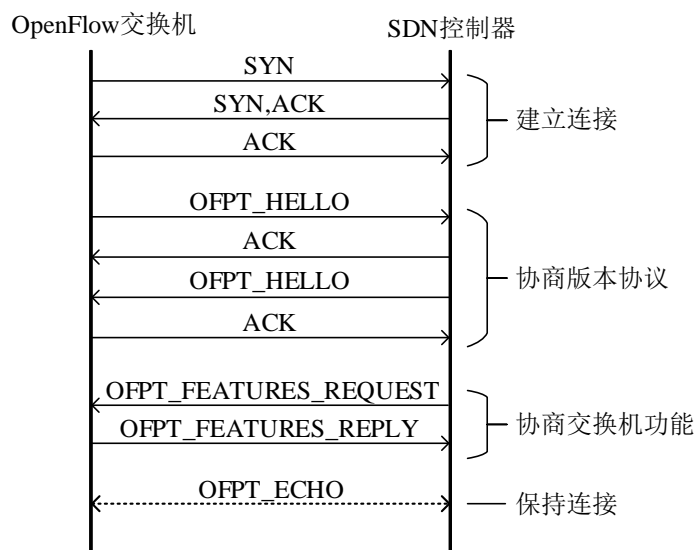


图 2.5 OpenFlow 通信流程

基于图 2.5，在 OpenFlow 协议中，交换机与控制器的通信流程包括如下四个步骤：

- a) 建立连接：通过三次握手建立安全通道；
- b) 协商版本协议：交换机与控制器相互发送 Hello 消息，协商协议版本；
- c) 协商交换机功能：控制器发出 Feature Request 报文，交换机将其特性信息（ID，缓冲区，端口等）封装在 Feature Reply 报文中回复至控制器；
- d) 保持连接，正常通信：完成上述交互后，进入正常通信状态。同时，控制器周期地向交换机发送 echo 报文以保持连接。

2.3 DDoS 攻击

2.3.1 DDoS 攻击原理

拒绝服务攻击（DoS，Denial of Service）指利用各种攻击手段耗尽攻击目标的资源，从而使得攻击目标无法为合法用户提供正常服务。其中，攻击目标可以是网络、计算机、应用等对象，资源则包括网络带宽、CPU、内存、开放的进程以及允许的连接等等。DoS 攻击分为带宽消耗型攻击和资源消耗型攻击两大类，常见的攻击手段包括：SYN Flood、UDP Flood、ICMP Flood 和 HTTP Flood 等。

当网络上两个或两个以上被黑客操控的电脑（也称僵尸主机）向攻击目标发起 DoS 攻击时，则被称为分布式拒绝服务攻击（以下简称 DDoS 攻击）。图 2.6 是典型的 DDoS 攻击示意图，其攻击流程如下：首先，攻击者感染大量的主机，被感染者称为僵尸主机；接着，攻击者控制僵尸主机向目标服务器发送大量看似合法的请求，消耗或长期占有大量资源；最终，服务器的资源被耗尽，无法处理合法用户的正常请求，攻击者从而达到了目的。

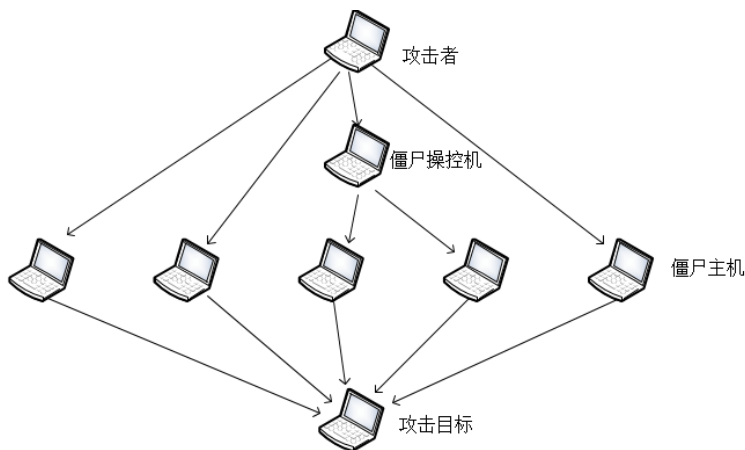


图 2.6 DDoS 攻击示意图

2.3.2 DDoS 防护体系

对抗 DDoS 攻击是一个不断完善的系统工程，一直以来均没有完美的、彻底的解决方案。随着攻击态势愈加凶猛，危害不断增强，人们对防御方案的研究也日益深入。下面根据系统所处的不同阶段，分别介绍防护 DDoS 攻击的几个方面。

(1) 攻击预防

顾名思义，即防止拒绝服务的发生，通过日常惯性的运维使得系统健壮稳固。在系统正常运行

时, 采取预防措施降低服务被攻陷的可能, 使得系统即使在被攻击时, 也能够为合法用户提供正常服务。攻击预防主要有两种方法: (a) 访问源管控, 即根据用户 (访问源) 的权限和行为严格监管其对于系统资源的访问权和使用权, 使非法用户无法触及资源, 从而保障系统的可用性, 文献^[55-59]中有关于该方法的详细阐述。(b) 资源冗余和备份, 提供大量的、足够的资源以应对 DDoS 攻击。攻防是双方资源的比拼, 提供足够的资源是承受住攻击的保障, 如扩充带宽、选用高性能服务器、部署服务器池以及采用负载均衡方案等等。

(2) 攻击检测

攻击检测的目标是尽可能早地探测到企图发起的 DDoS 攻击并及时触发响应机制。检测方法分为两类: 特征检测和异常检测。基于特征检测的系统将已知攻击的特征存储于数据库中, 并监测每个通信是否存在这些攻击模式, Snort 工具就是这类方法的典型代表。该方法可以实现高效率、低误差地检测已知攻击, 但需要频繁更新攻击的特征, 且只对已知类型的攻击有效。基于异常检测的系统能够检测出未知的攻击, 系统实时地将其当下的状态与正常状态中的网络流量或行为模型进行比对以检测是否存在异常, 无需预先定义攻击特征。文献^[60-64]分别提出了不同的异常检测应用实例。异常检测的关键在于 (a) 如何建立正常行为模型; (b) 如何权衡攻击覆盖率与误判率这对矛盾。对正常行为进行建模既可以基于标准化的规范, 例如根据 TCP 协议中实现三次握手以建立连接的要求, 检测半连接并标记为异常, 该方案误报率低但无法检测出伪装成正常流量的攻击。相比于标准化的行为建模, 应用更广泛的是基于训练所建立的模型, 即通过监测正常状态中网络的行为和流量特征, 训练出不同参数的阈值, 所有超出阈值的行为则被判定为异常。其中, 阈值的设定以及模型的更新是两个关键问题, 信息论、统计学等领域的深入研究为解决问题提供了理论基础, 而机器学习技术的出现和成熟又带来了新的思路。

(3) 攻击响应

攻击响应机制由检测系统触发, 采取一系列措施以减轻攻击对受害者产生的影响并尽快恢复受害者的服务能力。常见的响应方式包括流速限制、报文过滤、系统重构等。流速限制指限制疑似恶意流量的数据包速率, 这是一种较为宽容的应对方案, 常被部署于误报率较高的攻击检测系统中, 且无法避免部分恶意流量的进入, 文献^[61, 62, 64, 65]中详细介绍了流速限制的具体应用。报文过滤机制即根据检测系统识别出的恶意报文特征来过滤具有相同特征的报文, 例如动态防火墙。该机制的关键在于区分出合法报文与恶意报文的不同特征, 与上文基于训练的异常检测方案联合部署, 将达到很好的效果。系统重构指的是改变受害者的网络拓扑或临时增加系统资源以抵御攻击, 常见的例子包括可重构的 overlay 网络^[66]、资源副本服务^[67]等等。

2.3.3 SDN 中的 DDoS 攻击

SDN 中的 DDoS 攻击分为两大类: 针对主机的 DDoS 攻击以及针对 SDN 架构的 DDoS 攻击。本文主要针对的攻击类型为前者。

(1) 针对主机的 DDoS 攻击

该类型的 DDoS 攻击即传统网络中常见的攻击, 只是依附的网络架构变为 SDN 网络。攻击目标是位于 SDN 网络数据平面的主机, 攻击手段包括 SYN Flood、UDP Flood、ICMP Flood 和 HTTP Flood 等等。

(2) 针对 SDN 架构的 DDoS 攻击

图 2.7 所示为专门针对 SDN 架构的 DDoS 攻击，对网络的危害主要包括三个方面：

- 交换机内存超负荷：由于交换机收到无法匹配的数据包时会向控制器发出“packet-in”类型消息，同时缓存该数据包。攻击者通过发出大量的数据包将导致交换机内部缓存过多而超出负荷。
- 通信链路资源耗尽：攻击者发送大量的伪造数据包使得交换机不断地向控制器发出请求，而二者之间的安全通道带宽有限，最终将导致通信链路发生堵塞。
- 控制器资源消耗：控制器作为 SDN 网络中的大脑，在具有集中控制这个优点的同时存在单点故障的风险。控制器收到大量“packet-in”数据包后，其有限的 CPU 资源、内存资源以及 I/O 资源等将被大量消耗，导致系统延迟以及丢包等现象。

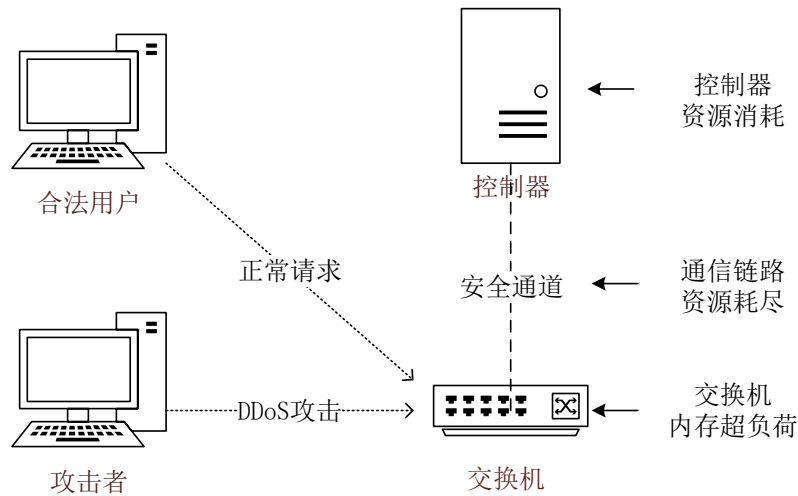


图 2.7 针对 SDN 的 DDoS 攻击

2.4 攻击防护的相关算法

基于训练建立攻击检测模型，即通过监测网络在正常状态和异常状态下的行为和流量特征来建立模型并基于模型识别攻击流量，是目前最受关注、应用最广的攻击检测方法。近年来，包括统计学、信息论和机器学习等多个领域的知识，被应用到攻击防护中。下面介绍本文所涉及的变点检测、CUSUM 控制图和机器学习等技术。

2.4.1 基于 CUSUM 控制图的变点检测

变点检测是统计学中的一个分支。在一个序列或过程中，当某个统计特性在某一时间点受系统性因素而非偶然性因素的影响发生了变化，即序列在该点之前符合某种分布，在该点之后符合另外一种分布，这个时间点则被称为变点。利用统计的方法，对时间序列的状态进行观测，估计变点位置的技术称为变点检测技术。

变点检测的关键在于根据序列统计特性的改变而判定异常点，常见的统计特征有：均值、方差和残差等。传统的变点检测大多基于统计原理，包括贝叶斯方法、最小二乘法、极大似然法等。随着统计控制过程（SPC, Statistical Process Control）的兴起，控制图技术迅速发展并广泛应用于变点

检测之中。控制图即运用数理统计方法，判断序列是否偏离典型分布，甄别是否存在异常。它用曲线表示出随着时间的推移所采集的样本，以显示样本是否在控制范围内。

目前 SPC 中最成熟的三大控制图是：休哈特控制图（Shewhart）、累积和控制图（CUSUM）以及指数加权滑动平均控制图（EWMA）。Shewhart 控制图需要满足样本容量大、样本服从或接近正态分布以及样本相互独立等条件，而实际过程中这些条件很难满足。CUSUM 控制图则拓宽了 Shewhart 控制图的应用范围，而且弥补了 Shewhart 控制图对中小漂移和异常点的检测灵敏性，并且相比于 EWMA 控制图，CUSUM 控制图对于监控过程的随机性有更强的容忍度，因此，本文主要研究 CUSUM 控制图并在其基础上进行改进。

CUSUM 算法^[68]最初由 Page 在 1954 年提出，其理论基础是序贯分析原理中的序贯概率比检验（SPRT, sequential probability ratio test）理论。CUSUM 的核心思想是对样本序列加以累积，将序列的小偏移累积起来，达到放大的效果，从而提高检测过程中对小偏移的灵敏度。较高的灵敏度能够更好地降低风险，因此非常适用于攻击检测的场景。

2.4.2 基于机器学习的异常检测

近年来机器学习被广泛应用于异常检测和攻击检测中。机器学习分为监督学习、非监督学习以及强化学习等三大类，如图 2.8 所示。

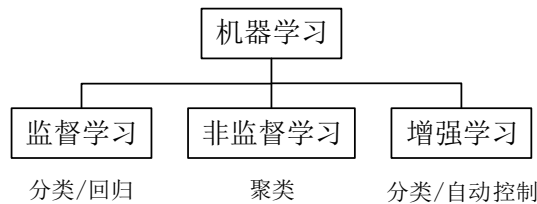


图 2.8 机器学习分类

基于监督学习对网络流量进行分类以识别异常流量是攻击检测的主要方法。分类算法使用训练数据（训练数据中包含特征和分类标签）来构建预测模型，这些预测模型可以使用它从训练数据中所挖掘出来的特征来对新的、未曾见过的数据集进行分类标签预测。常用的分类算法包括：逻辑回归（LR, Logistic Regression）、朴素贝叶斯（NB, Naive Bayes）、K-近邻（KNN, k-Nearest Neighbor）、支持向量机（SVM, Support Vector Machine）以及决策树（DT, Decision Tree）等等。

逻辑回归属于广义的线性回归模型，是一种判别模型，通过直接对条件概率 $P(y|x)$ 建模，建立代价函数例如最大化似然函数或最小损失函数，然后通过优化方法迭代求解出最优的模型参数。该算法分类速度较快，但容易发生欠拟合的现象，对于非线性可分的问题其分类精度较低。

朴素贝叶斯分类是一种生成模型，在满足特征条件独立性的前提下，先对数据的联合分布建模，基于贝叶斯定理，通过对象的先验概率，利用贝叶斯公式计算出其后验概率来进行分类。该算法理论基础严谨、需要调节的参数较少且分类速度较快，但要求属性之间相互独立，在很多场景中并不适用。

K-近邻算法根据计算出的不同特征值之间的距离来进行分类。它将测试集的数据特征与训练集的数据进行特征比较，然后提取样本集最近邻数据特征的分类标签。K-近邻算法的核心计算过程是计算测试数据与类别中心的距离，具有精度高、对异常值不敏感、无数据输入假定、简单有效的特

点，但计算复杂度太高，每分类一个数据需要计算所有数据。

支持向量机的核心思想是通过把数据映射到多维空间中以点的形式存在，找到能够区分不同类别的最优超平面，并根据这个平面进行分类。该算法拥有较高的分类正确率，对过拟合有很好的理论保证，通过选取合适的核函数，对特征线性不可分的问题也有较好的表现。但其对参数调节和核函数的选择过于敏感，且需要较大的内存。

决策树算法基于树的理论实现数据分类。作为一个非参数的预测模型，它代表对象属性与对象值之间的映射关系。其核心是通过计算信息熵来对树进行分裂，以实现数据集的划分。该算法无需担心离群点和数据是否线性可分的问题，且运行速度较快。但其容易过拟合，因此随机森林（RF，Random Forest）以及自适应提升算法（AdaBoost）等集成学习算法被提出，解决过拟合的问题。

集成学习本身不是一个单独的机器学习算法，而是通过构建多个机器学习器来完成学习任务。其核心思想是针对同一个训练集训练不同的个体学习器（弱分类器），通过整合弱分类器进行组合预测，构成最终的分类器（强分类器），图 2.9 所示为集成学习的一般结构。算法本身是通过改变数据分布来实现：根据每次训练集中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权值；将修改过权值的新数据集送给下层分类器进行训练，最后将每次训练得到的分类器最后通过一定的策略整合起来，作为最后的决策分类器。

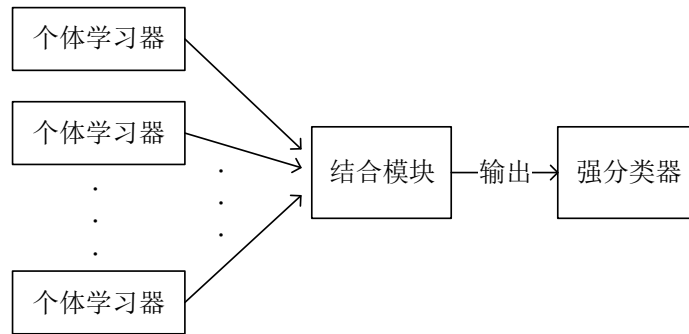


图 2.9 集成学习的一般结构

集成学习中的个体学习器可以是同质或异质的，同质个体学习器应用最为广泛，其常见的模型是决策树和神经网络。同质个体学习器按照学习器之间是否存在依赖关系，分为两类：（1）存在强依赖关系且学习器需要串行生成；（2）不存在强依赖关系。二者分别有两个主流框架：boosting 和 bagging。

图 2.10 表示了 boosting 框架的算法流程。训练过程为阶梯状，首先基于训练集用初始权重训练出弱学习器 1，根据其学习误差率的表现来更新训练样本的权重，提升在弱学习器 1 中误差率高的训练样本点的权重，使得这些误差率高的点在后续弱学习器中得到更多的重视；接着基于调整权重后的训练集来训练弱学习器 2，如此重复进行，直到弱学习器数达到事先指定的数目 T ，最终将这 T 个弱学习器通过集合策略进行整合，得到强学习器。

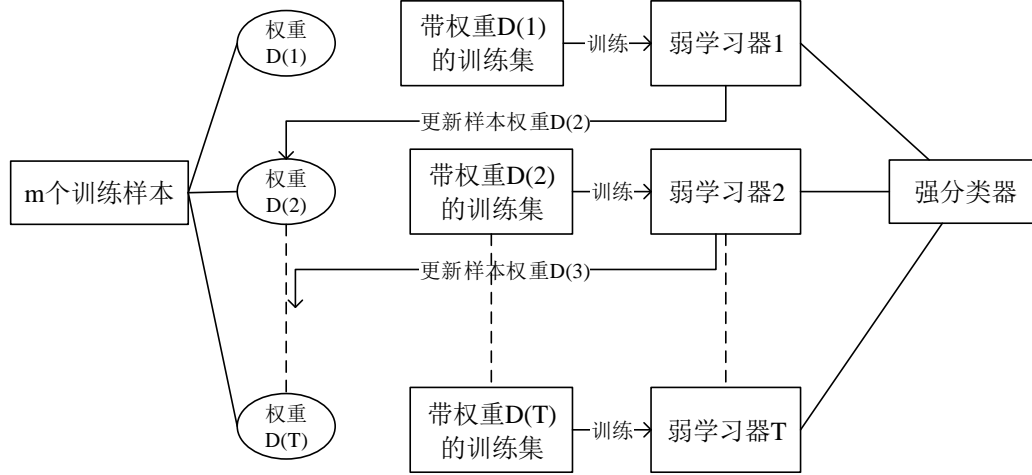


图 2.10 boosting 框架

图 2.11 表示了 bagging 框架的算法流程。与 boosting 框架不同，它的弱学习器之间没有依赖关系，可以并行生成。对训练集进行随机抽样得到 T 个采样集，再由这 T 个采样集分别独立地训练出 T 个弱学习器，最终将这 T 个弱学习器通过集合策略进行整合，得到强学习器。

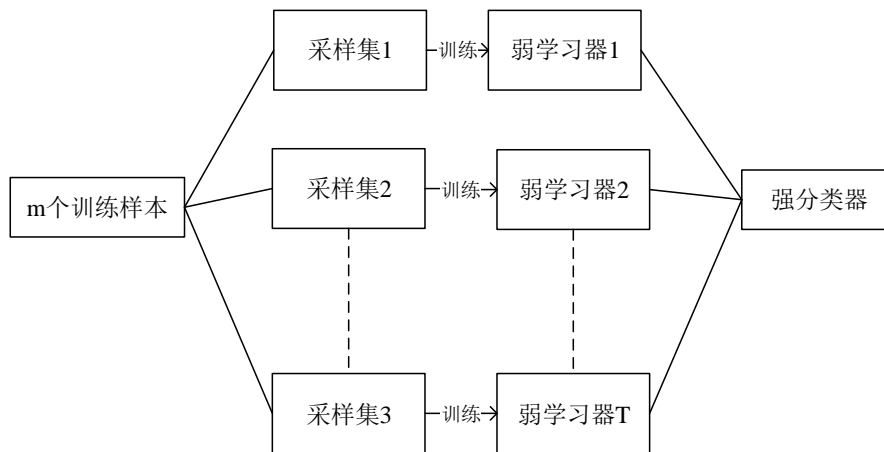


图 2.11 bagging 框架

2.5 本章小结

本章主要对基于SDN的DDoS攻击防护系统的相关技术进行阐述。首先研究了SDN技术，包括软件定义网络的架构、特征以及OpenFlow协议和相关标准；其次，深入分析了DDoS攻击的原理，并从攻击预防、攻击检测和攻击响应等三个方面研究DDoS攻击的防护技术，此外对SDN环境中不同类型的DDoS攻击进行简要分析。此基础上，详细介绍了本文提出的攻击防护系统所涉及的相关算法的背景和理论知识，包括变点检测、CUSUM控制图、机器学习常见算法以及集成学习框架等等。

第三章 基于 SDN 的 DDoS 检测方案

3.1 引言

近年来,在 SDN 环境中进行 DDoS 攻击的检测受到了广泛的关注。SDN 的出现为检测 DDoS 攻击带来了新的思路:其全局视角和集中控制的特性为实时监控全网的流量分布和各节点的流量数据提供了条件,其数控分离和动态可编程的特性为网络中报文的深度分析、流量的监控管理以及转发规则的更新提供了基础。

基于以上优点,SDN 中的 DDoS 检测方案不断涌现,其核心思路是利用 SDN 的特性,通过采集网络信息、提取特征、分析特征以及选用合适的检测算法等一系列步骤,识别异常主机或异常流量,实现攻击的检测。

本文主要研究部署在控制器中的检测方案。文献^[10-18]基于传统的统计学理论提出了检测方案,其中文献^[12-15]使用熵作为表征网络流量的关键特征,如文献^[14]依据网络遭受 DDoS 攻击时其目的 IP 地址分布的随机性将减小的现象,使用目的 IP 地址的熵值来衡量目的地址分布的随机性,进而实现攻击检测。对特征量的异常检测,除了通过多次模拟攻击的实验来设定阈值这种方式,文献^[10, 11, 17, 18]使用基于 CUSUM 控制图的方案对特征量的变化进行跟踪,建立模型并自动探测异常值。以上基于传统统计学的检测方案,由于算法依赖于前期模拟的攻击实验,导致算法的适应性与可移植性不强,同时难以建立全方位的流量模型,检测的准确率存在瓶颈。为了解决这些问题,许多文献^[20-31]将机器学习技术应用于检测算法,包括支持向量机、逻辑回归、K-近邻、朴素贝叶斯、神经网络等算法。Braga^[21]分析了与 DDoS 攻击相关的统计信息,提出基于流的特征“六元组”,并使用自组织映射算法根据“六元组”识别恶意流量,但 SOM 算法收敛慢,导致训练时间较长以及系统开销过大。

综上所述,现有的研究存在以下问题:

(1) 因系统开销与检测延时之间的矛盾导致检测周期难以确定。检测方案通过轮询的方式来采集网络信息,并基于这些信息提取特征并进行分类。作为检测方案的基础,采集到的网络信息十分重要,而轮询的周期决定了信息的粒度,因此如何选择轮询周期成为关键问题,周期太长将导致检测延时过长,而过短的周期则导致系统开销太大,造成控制器的负担。

(2) 特征提取的方法有待改善。一方面,现有研究提取的特征相似度较高,如提取源和目的地址的熵,然而流量的变化体现在各个维度,现有方案难以建立全面的流量模型;另一方面,多数研究提取特征时只关注当前网络状态,忽视了历史状态,而网络遭受 DDoS 攻击时其流量的变化与时间相关性较大,因此,若使用合适的特征来表征流量随时间变化的特性,将提升检测系统的准确率。

基于上述问题,本文提出了由触发检测和深度检测两个模块构成的检测方案,将低开销、粗粒度的触发检测算法与高精度、细粒度的深度检测算法相结合,以扬长避短。触发检测模块在网络中保持长期运行,通过提出的 SMNA-CUSUM 算法实时跟踪多维度序列进行粗粒度的异常检测,同时调节参数保障极低的漏检率,第一时间发现可能的异常,起到预检测的作用;深度检测模块则在触发检测模块发起警报后启动,提取更细粒度的特征并引入基于时间窗的特征构造方案,使用高精度的 AdaBoost 算法识别流量是否异常。此外,本文给出了检测方案中所涉及的参数的分析和选择。

3.2 检测系统架构

图 3.1 描述了攻击检测系统的整体架构，系统主要包括触发检测和深度检测两个检测模块，此外还包括 packet-in 消息处理与流表信息采集这两个与控制平面直接交互、为攻击检测提供网络信息的模块。

触发检测模块在网络中保持长期运行，提取全网流量特征并执行粗粒度的异常检测。首先，网络信息的采集来自两个方面，一方面流表信息采集模块通过主动轮询的方式获取所有交换机内的流表信息，另一方面 packet-in 处理模块被动接收来自交换机的 packet-in 类型的消息并进行处理；其次，特征提取模块基于采集到的信息，提取表征网络流量状态的四维特征向量并构造特征序列；最后，异常检测模块基于 SMNA-CUSUM 算法，对特征序列进行跟踪观测，若发现异常则立即发起攻击预警，通知深度检测模块。

深度检测模块在接收到攻击预警后立即运行，提取更细粒度的特征并使用基于决策树的 AdaBoost 算法识别攻击。首先，特征提取模块从流表信息采集模块获取网络信息，提取表征网络流量状态的八个特征并基于滑动时间窗进行特征向量的构造。接着，攻击识别模块根据基于决策树的 AdaBoost 算法以及训练好的模型，实时地对特征向量进行分类，做出最终的判决，若判决为异常，则表示网络中存在 DDoS 攻击。

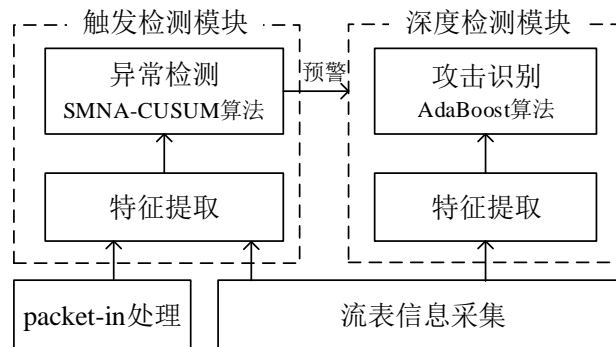


图 3.1 攻击检测系统架构

3.3 触发检测模块

触发检测是检测方案的第一个模块，也是长期在网络中保持运行的模块。本文提出了一种改进的 CUSUM 算法：基于滑动窗口的多维度自适应阈值非参数累积和算法（SMNA-CUSUM, Sliding-window based Multi-dimension Non-parametric CUSUM Algorithm with Adaptive threshold）。该算法作为触发检测系统的核心，负责对全网流量进行粗粒度的异常检测。本节将详细阐述触发检测的完整流程，包括流表信息采集、特征提取和分析以及多维特征序列的构造，最后介绍基于 SMNA-CUSUM 算法的异常检测过程以及参数的选择。

3.3.1 流表信息采集

触发检测模块的第一步是收集 SDN 网络中的流量信息，以便提取特征。在 SDN 环境里，各个交换机中的流表统计信息是全网流量信息的主要来源。流表信息采集模块负责收集和统计各个交换

机中的流表信息。该模块基于 OpenFlow 协议以轮询的方式定期地向所有交换机发送收集流表信息的请求，并通过安全通道接收交换机回复的流表信息，包括交换机内所有的流表项以及相应的计数器信息，最后对上述信息进行处理和统计。

首先介绍交换机中的流表信息的更新机制。每个交换机在完成与控制器的认证后加入网络并被分配一个 ID，此后开始正常的工作：一方面通过安全通道与控制器进行通信，另一方面根据控制器下发的流表中的规则对网络数据包进行转发。每当一个数据包到达交换机时，都会经历流表匹配的过程，即将报文信息与流表项的匹配域进行匹配，流表项的具体信息见图 2.3。

一旦数据包与流表项匹配成功，该流表项中计数器的对应统计字段则会实时更新，流表项的计数器信息见表 3.1。若数据包匹配失败，则被交换机转发至控制器，控制器根据业务规则生成新的流表并下发至交换机，告知交换机处理此类数据包的方法。

表 3.1 流表项计数器信息

统计量	Bits
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32

接下来阐述信息采集模块通过轮询的方式来收集流表信息的过程。记信息采集周期为 T_{poll} ，每隔 T_{poll} 该模块向网络中所有交换机发送收集信息的请求，即 OFP_FLOW_STATS_REQUEST 类型的报文；每个交换机在收到请求后，向控制器回复 OFP_FLOW_STATS_REPLY 报文，报文中包含了表 3.1 所示的流表项统计信息；流表信息采集模块解析报文，对数据进行处理。注意，信息采集周期 T_{poll} 的取值非常关键，过大将导致检测的延迟，过小将导致控制器资源的大量消耗。

由于每个交换机回复的原始报文是一整块流表信息，包含了不同的流表项以及相应的计数器信息，因此，流表信息采集模块在收到交换机回复的报文后，将执行如下操作：首先，将每条流表项与其对应的计数器信息取出；其次，由于流表项计数器中记录的信息是从该流表项被建立后至当前时间的统计信息，而此处需要采集当前周期内的统计信息，因而该模块需要记录上一个周期的统计信息，二者之差才是该流表项在每个时间周期内的统计信息：

$$X_i^k = S_i^k - S_{(i-1)}^k, \quad (3.1)$$

其中， k 表示第 k 个统计量， X_i^k 表示该字段在第 i 个周期的统计值， S_i^k 表示第 i 个周期，来自交换机的原始报文中计数器记录的统计量的值。至此，信息采集模块完成流表信息的处理。每个轮询周期中，当流表信息采集模块完成对原始报文的处理后，即可将处理后的数据传递至特征提取模块，执行特征提取。

综上所述，基于 OpenFlow 协议来采集流表统计信息以获取全网流量信息的方式，简单有效，相比于文献^[12]中使用 sFlow 来采集流表信息的方式，无需另外部署 sFlow 等三方工具，更加便捷，易于实施。

3.3.2 特征提取和分析

流表信息采集模块每隔时间周期 T_{poll} ，将获取到该周期内所有的流表信息。定义向量 \mathbf{C}_i ，表示

流表中第 i 个流表项的信息:

$$C_i = [dpID_i, srcIP_i, dstIP_i, pkts_i, bytes_i, speed_i], \quad (3.2)$$

其中, 下标 i 表示流表项的序号, $dpID$ 表示该流表项所在交换机的 ID, $srcIP$ 和 $dstIP$ 分别表示流表项中匹配域内的源 IP 地址和目的 IP 地址, $pkts$ 、 $bytes$ 和 $speed$ 表示与该流表项匹配的数据包的统计信息: 数据包数、总字节数、流速。本节将基于以上数据来提取网络遭受 DDoS 攻击时其流量的典型特征。

出现 DDoS 攻击的网络, 将呈现以下可能的特性:

- (1) 突发性: 网络中数据包的剧增;
- (2) 延迟性: 网络拥堵严重, 延迟巨大;
- (3) 非对称性: 网络中充斥大量半连接的数据包;
- (4) 攻击流数据包源 IP 分布离散, 发起大量数据包, 目的 IP 分布较为集中。

由于 DDoS 攻击类型不尽相同, 以上状态并非同时出现, 因此, 提取流量特征时需要考虑多个维度, 以尽量覆盖不同攻击所引起的网络状态变化。结合文献^[69, 70]对网络遭受 DDoS 攻击时其流量特征分布的研究, 本文从以下几个方面提取特征。

首先, 网络受到攻击时其流量大小的变化是最为明显也最易检测的, 流量的大小可以通过网络中所有数据包的和以及所有字节的和来表示, 同时基于 SDN 的特殊环境, “流” 的数量也可以表征流量的大小。但仅从流量大小这个角度来判断是否出现攻击, 缺乏严谨性: 一方面, 当网络出现热点事件或某个业务出现增长时, 将导致访问量增加, 甚至发生“闪拥”现象, 极有可能被误判为 DDoS 攻击; 另一方面, 慢速 DDoS 攻击无法通过流量大小这项指标来识别。因此, 需要考虑结合其他特征。DDoS 攻击发生时, 由于僵尸主机被利用, 发出大量的数据包, 流表统计信息中源 IP 地址是僵尸主机的概率增大, 源 IP 地址随机性减少, 而受害者收到的数据包增多, 导致网络中目的 IP 地址的分布更为集中, 随机性减少, 基于上述特征, 本文引入熵的概念, 来衡量网络中源 IP 和目的 IP 的随机性。

熵的概念源于热力学, 后来由 Shannon 将其引入信息论中, 称为信息熵, 用于度量信息的不确定性。随机性越大, 表示不确定性越多, 信息熵越大; 反之则信息熵越小。以目的 IP 地址为例, 其信息熵的计算方法是:

$$H(dstIP) = \sum_{i=1}^n p_i \log p_i, \quad (3.3)$$

其中, n 表示目的 IP 地址的总数量, p_i 表示第 i 个 IP 地址出现的概率, p_i 的计算方式如下:

$$W = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots\}, \quad (3.4)$$

$$p_i = \frac{y_i}{n}, \quad (3.5)$$

其中, W 代表了一个时间周期内采集的目的 IP 地址的集合, x_i 表示某一个目的 IP 地址, y_i 表示该地址出现的次数。

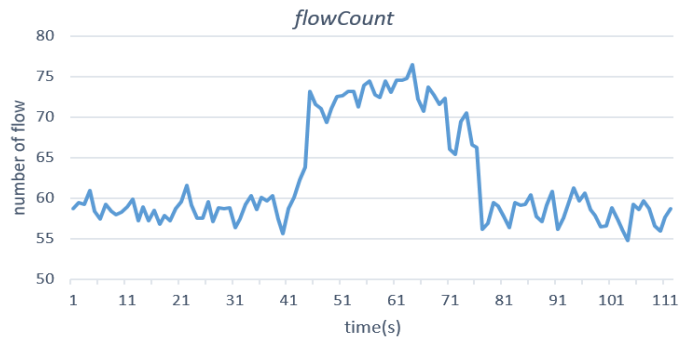
当每个 IP 地址均出现一次时, 目的 IP 地址的熵 $H(srcIP)$ 达到最大值, 当网络中出现 DDoS 攻击, 多个数据包的目的 IP 地址均集中到被攻击者 IP 上, 随机性变小, 熵值也将明显变小。

基于上述分析,并结合 SDN 网络基于流的数据转发特性,从流表信息中提取如表 3.2 所示的几个特征来表示网络流量的变化,作为识别是否遭受攻击的依据。

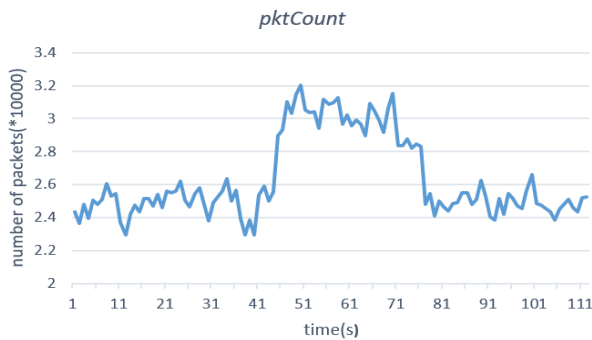
表 3.2 表征网络流量变化的典型特征

特征量	计算方法
总流表数	$flowCount = n$
总数据包数	$pktCount = \sum_{i=1}^n pkts_i$
总字节数	$byteCount = \sum_{i=1}^n bytes_i$
源 IP 地址的熵	$H(srcIP) = \sum_{i=1}^n p_i \log p_i$
目的 IP 地址的熵	$H(dstIP) = \sum_{i=1}^n p_i \log p_i$

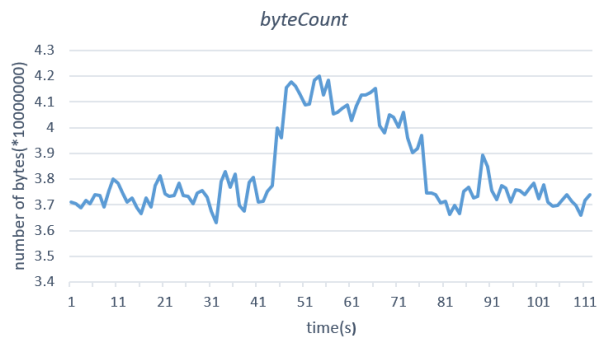
为了对上述特征做进一步分析,观测其在正常状态以及攻击状态下的实际变化情况,本文在 SDN 网络中模拟 DDoS 攻击,图 3.2 所示为网络流量特征的变化情况,图 (a) 至 (e) 分别表示表 3.2 中所示的 5 个特征量。实验中,在第 150s 发起持续时间约为 80s 的 TCP SYN Flood 攻击,每 5s 收集一次流表信息,共收集信息约 5 分钟。



(a)



(b)



(c)

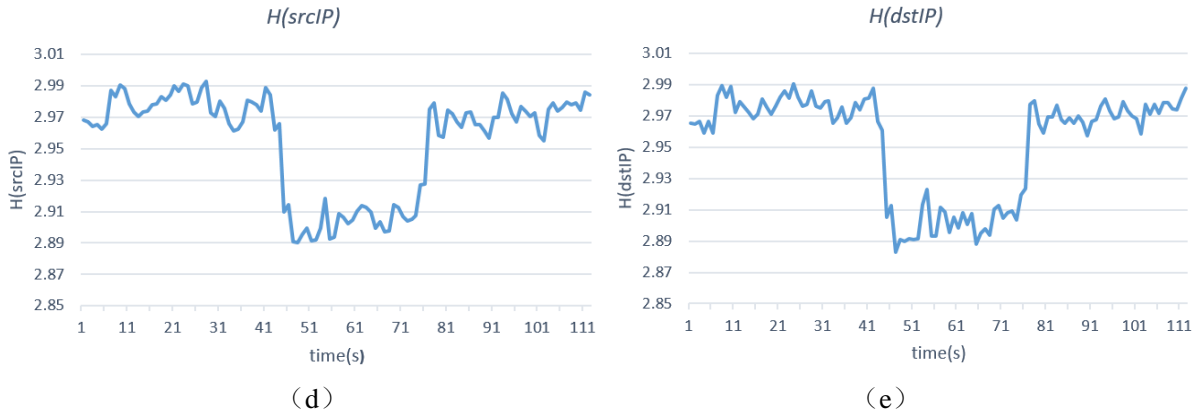


图 3.2 DDoS 攻击时五个特征量的变化情况

由上图可见，五个特征量均出现了明显的变化，由于流的数量、总数据包数和总字节数这三个特征相关性较大，因此从中选取一个特征以降低冗余，本文选择总数据包数。最终，经过特征提取，采用以下三个特征来粗粒度地表示网络流量的变化：总数据包数 $pktCount$ ，源 IP 地址的熵 $H(srcIP)$ ，目的 IP 地址的熵 $H(dstIP)$ 。

3.3.3 多维特征序列的构造

基于 SDN 网络环境，特征序列来自控制器中两个不同的数据采集模块：通过主动轮询方式采集信息的流表信息采集模块以及通过被动接收方式接收 $packet-in$ 数据包来采集信息的 $packet-in$ 处理和采集模块，如图 3.3 所示。流表信息采集模块和 $packet-in$ 处理和采集模块分别与特征提取模块相结合，构成控制器主动轮询和控制器被动接收大框架。

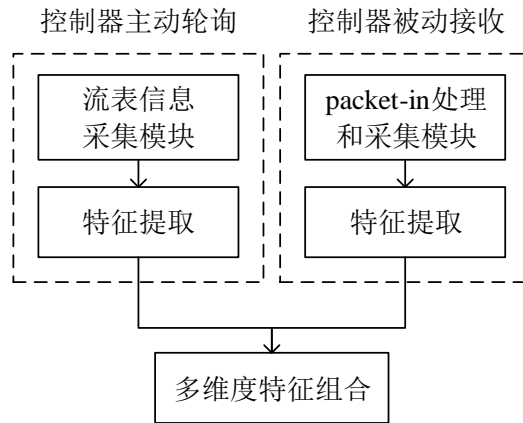


图 3.3 特征序列的来源

(1) 控制器主动轮询

控制器主动轮询的过程在 3.3.1 和 3.3.2 节中已经详细介绍：通过定期向交换机发送流表信息请求来收集流表信息并进行特征提取，最终输出三个特征。主动轮询的方式获取的是能够成功匹配交换机流表的数据包的分布情况，在此基础上进行特征提取和分析，可以检测网络中与已存在的流表项匹配的数据包是否发生异常。但值得注意的是，还有另一类数据包，即未成功匹配到任何流表项的数据包，这部分流量信息将被控制器被动接收，由 $packet-in$ 处理和采集模块进行分析。

(2) 控制器被动接收

在 SDN 网络中, 当交换机收到某个数据包时将尝试寻找匹配的流表项并根据规则转发, 若匹配失败, 则会主动向控制器发送一条 “packet-in” 类型的消息, 简称 packet-in 消息, 内容包括交换机 ID 以及数据包相关信息; 控制器端被动接收到该消息并根据业务逻辑进行决策, 制定转发规则。packet-in 消息的速率代表网络中新出现的 “流” 的数量。当 DDoS 攻击发生时, 网络中的新的数据流剧增, 大量的 packet-in 消息将被发送至控制器。因此, 控制器中 packet-in 消息的接收速率也是判断是否存在 DDoS 攻击的一个重要指标。

在控制器内设计基于 packet-in 消息的特征提取模块, 包括 packet-in 处理和采集模块、计数器以及计时器, 具体处理流程如图 3.4。每当 packet-in 消息到达控制器时, 对应的计数器加一, 接着由计时器模块判断是否达到一个周期的时间, 若达到则将计数器的值输出, 记为特征变量 $pkt_InCount$, 与此刻主动轮询获取的另外三个变量一起构成特征向量, 同时将计数器清零并重新开始计时, 则继续接收 packet-in 消息。

通过对 packet-in 数据包进行计数, 可以实时监控突发的新连接的流量情况, 与上文所述对已有连接的数据包监控结合起来, 全方位的进行网络流量监控。

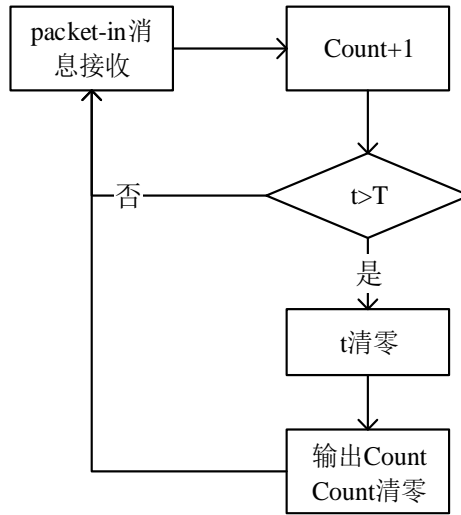


图 3.4 基于 packet-in 消息的特征提取流程

综上所述, 特征向量通过控制器主动轮询以及被动接收数据包这两个途径产生, 共提取四个维度的特征来表示网络流量, 分别是总数据包数 $pktCount$, 源 IP 地址的熵 $H(srcIP)$, 目的 IP 地址的熵 $H(dstIP)$ 和新出现流数 $pkt_InCount$ 。每隔时间周期 T_{poll} , 提取到的特征构成一个向量, 记为 C_t :

$$C_t = [pktCount, H(srcIP), H(dstIP), pkt_InCount]. \quad (3.6)$$

C_t 即为异常探测所需的特征向量。而异常检测算法需要通过观测特征序列来探测异常, 下面介绍多维特征序列的过程。本文采用如图 3.5 所示的滑动窗口机制来构造特征序列, 窗口长度为 8。每隔时间 T_{poll} , 最新的特征向量 C_t 被加入已构造的特征序列中, 同时, 滑动窗口向右滑动一个单位, 红色虚线框内新的特征序列被输入到检测算法中。

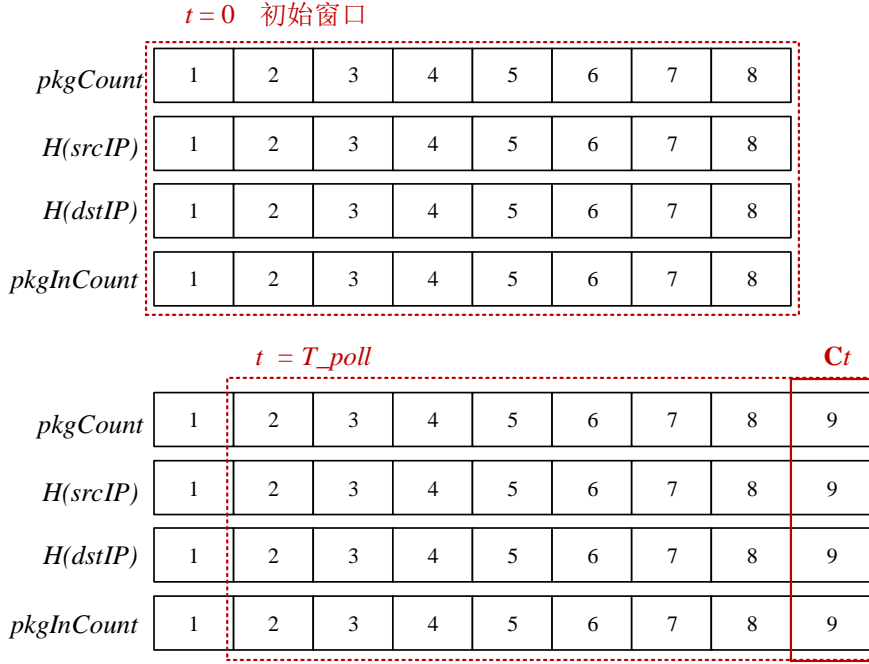


图 3.5 构造多维特征序列的滑动窗口机制

3.3.4 基于 SMNA-CUSUM 算法的异常检测

本节提出基于滑动窗口的多维度自适应阈值非参数累积和（SMNA-CUSUM）算法，该算法在 CUSUM 算法的基础上进行改进，实现基于特征序列的异常检测。

将网络中的流量看作一个随机模型，当发生 DDoS 攻击时，该模型的分布将发生变化，变点检测技术基于这一特征进行变点检测，也称异常检测。CUSUM 控制图是一种应用广泛的变点检测技术，通过度量当前值与目标值的累积偏差来判定是否出现异常。基于序贯分析原理，CUSUM 控制图在判断某个过程时，不是仅考虑前一次观测值，而是以所有的历史观测数据作为评估依据，由于充分结合了较长一段时间的数据信息，达到了偏差累积放大的效果，因此 CUSUM 控制图有着很高的灵敏度以及很短的检测时间。

CUSUM 控制图的核心是 CUSUM 算法。原始的 CUSUM 算法依赖于随机序列的参数模型，根据模型的概率密度函数来监控序列，计算每个样本值与目标值偏差的累积和。而网络流量模型是一个复杂的随机模型，其特征序列的整体分布难以预测，因此无法通过一个准确的参数模型来表示。此处引入非参数的方法来解决这个问题，文献^[71]证明了非参数的 CUSUM 算法在网络异常检测方面的适用性。

非参数的 CUSUM 算法与原始 CUSUM 算法的不同点在于：用正常运行状态下的平均值（即观测到的历史均值）替代原始算法中基于参数模型求出的目标值，通过当前值与平均值的累积偏差来判定是否出现异常。本文对非参数的 CUSUM 算法进行改进，以适应 SDN 环境中基于网络流量进行异常检测这一特殊场景，主要改进部分包括以下三点：

(1) 基于滑动窗口的实时检测

由于对网络进行异常检测是一个持续的过程，检测算法需要实时提取数据进行判决。过长的序列将导致计算复杂度的提升和存储资源的浪费，并且难以适应不断变化的网络流量模型，因此，本

文改进算法以适应基于滑动窗口的数据输入模型,在为序列添加新的数据的同时,及时移除旧数据,以保障序列的新鲜度。对于序列长度 L 的选择,遵循在满足为算法提供足够的历史数据信息的条件下尽量减少长度的原则。

(2) 自适应的阈值调节

CUSUM 算法通过判决函数 d_N 中的阈值 N 来判断是否发生异常,公式如下:

$$d_N = \begin{cases} 0, & y \leq N \\ 1, & y > N \end{cases}, \quad (3.7)$$

其中, y 表示累积和系数。

对于实时变化的网络流量模型来说,固定的阈值存在以下不合理性:首先,设定好的固定阈值将受限于本系统,缺乏拓展性,不同网络需要设定不同的阈值;其次,固定的阈值将无法适应网络流量的变化趋势,比如在高峰时期流量将明显大于低谷期,相同的阈值显然不合理。因此,本文提出了自适应的阈值调节算法,实时计算滑动窗口(长度为 L)中序列 X_i 的方差,阈值随着方差的变化而改变,与方差成线性关系,系数为 h (h 值的选择过程下文将详细阐述):

$$N = h * \sigma_{X_L}, \quad (3.8)$$

其中, σ_{X_L} 表示序列的方差:

$$\sigma_{X_L} = \sqrt{\frac{\sum_{i=1}^L (X_i - \mu_{X_L})^2}{(L-1)}}, \quad (3.9)$$

其中, μ_{X_L} 表示序列的均值:

$$\mu_{X_L} = \frac{\sum_{i=1}^L X_i}{L}. \quad (3.10)$$

(3) 多维特征结合

在攻击发生时,由于攻击种类不同,其引起网络流量的变化方式也不尽相同,单一维度的特征不足以全面地捕捉到网络流量的变化,换句话说,单一维度的特征将造成极高的漏检率。因此,本文改进算法以适应多维特征序列的检测。假设提取三个维度的特征,对于每个滑动窗口,均有序列 $X_{i,1}, X_{i,2}, X_{i,3}$, 分别计算累积和系数得到 y_1, y_2, y_3 , 第 k 维特征的判决函数为:

$$d_{N_k} = \begin{cases} 0, & y_k \leq N_k \\ 1, & y_k > N_k \end{cases}, \quad (3.11)$$

由此,得到一个三维的判决向量 $\mathbf{D} = (d_{N_1}, d_{N_2}, d_{N_3})$, 最终的判决函数为:

$$r(\mathbf{D}) = \begin{cases} 0, & \|\mathbf{D}\| < 1 \\ 1, & \|\mathbf{D}\| \geq 1 \end{cases}, \quad (3.12)$$

其含义是,当任一维度的特征判决结果为异常时,最终判决结果则为异常。作为在触发检测场景使用的算法,采用上述严格的判决方式可以获得极低的漏报率,一旦发现可能存在异常,则发起攻击预警,由深度检测模块来执行进一步检测。

综合上述改进点，SMNA-CUSUM 算法流程见图 3.6。

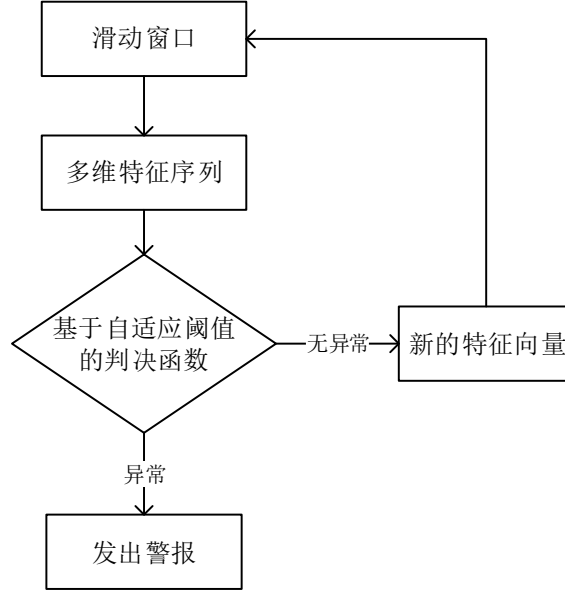


图 3.6 SMNA-CUSUM 算法流程图

根据图 3.6，本文提出的 SMNA-CUSUM 算法的完整步骤如下：

首先，确定长度为 L 的滑动窗口，提取该窗口中的 K 个特征序列并记为 $\{X_{i,k}\}$ ，表示第 k 维特征中的第 i 个样本点， μ_k 表示第 k 维特征序列的均值：

$$\mu_k = \frac{\sum_{i=1}^L X_{i,k}}{L}, \quad (3.13)$$

σ_k 表示第 k 维特征序列的方差：

$$\sigma_k = \sqrt{\frac{\sum_{i=1}^L (X_{i,k} - \mu_k)^2}{(L-1)}}, \quad (3.14)$$

对每个特征序列中的样本点，计算其累积和的上下界 $U_{i,k}$ 以及 $L_{i,k}$ ：

$$U_{i,k} = \begin{cases} 0, & i=1 \\ \max\left(0, U_{i-1,k} + X_{i,k} - \mu_k - \frac{1}{2}\sigma_k\right), & i>1 \end{cases} \quad (3.15)$$

$$L_{i,k} = \begin{cases} 0, & i=1 \\ \min\left(0, L_{i-1,k} + X_{i,k} - \mu_k + \frac{1}{2}\sigma_k\right), & i>1 \end{cases} \quad (3.16)$$

经过 L 次计算，即 $i = L$ 时得到该窗口中每个特征序列最终的上下界 $U_{L,k}$ 和 $L_{L,k}$ ，简记为 U_k 和 L_k ：定义第 k 个特征序列的判决函数：

$$d_k = \begin{cases} 0, & U_k \leq N_k \text{ and } L_k \geq -N_k \\ 1, & U_k > N_k \text{ or } L_k < -N_k \end{cases}, \quad (3.17)$$

其中， N_k 表示第 k 个特征序列的阈值：

$$N_k = h * \sigma_k, \quad (3.18)$$

其中, h 表示自适应阈值与方差的线性系数, 此处不同特征序列采用相同的系数。将计算得出的 k 个上下界: U_k 和 L_k , 分别带入 k 个判决函数中, 得到 k 维的判决向量 $\mathbf{D} = (d_1, d_2, \dots, d_k)$, 此时, 每个维度的特征序列均已判决完毕, 记 $r(\mathbf{D})$ 为最终的判决函数:

$$r(\mathbf{D}) = \begin{cases} 0, & \|\mathbf{D}\| < 1 \\ 1, & \|\mathbf{D}\| \geq 1 \end{cases}, \quad (3.19)$$

将特征向量 \mathbf{D} 带入判决函数中, 得到当前滑动窗口的最终结果: 1 表示异常, 0 表示正常。至此, 完成对一个滑动窗口 (序号为 1 至 L) 的检测。

将每个特征序列的滑动窗口同时后移一个值, 即序号为 2 至 $L+1$, 重复上述步骤, 完成对下一个滑动窗口的检测。在真实的网络环境中, 每隔周期 T_{poll} 产生一个特征向量, 该向量触发滑动窗口的后移行为。检测算法对当前滑动窗口内的序列进行异常检测实际上是对该窗口内最后一个样本点进行异常检测。由此, 每次实时采集的新样本均能结合历史数据信息被及时处理, 低延时地完成检测。

下面将阐述触发检测模块的完整流程:

- (1) 确定信息采集周期 T_{poll} , 以及 SMNA-CUSUM 算法中的滑动窗口长度 L ;
- (2) 每隔周期 T_{poll} , 控制器通过主动轮询以及被动接收数据包的方式收集网络流量信息并进行特征提取, 构造特征向量 \mathbf{C}_t ;
- (3) 重复步骤 (2) L 次, 构造多维特征序列, 完成初始滑动窗口的构建;
- (4) 每隔周期 T_{poll} , 控制器收集网络流量信息并进行特征提取, 构造特征向量并输入 SMNA-CUSUM 算法, 对当前周期进行异常检测, 若发现异常, 则触发深度检测模块;
- (5) 不断重复步骤(4), 完成对网络流量的实时跟踪与异常检测。

表 3.3 所示为触发检测模块中所涉及的参数:

表 3.3 触发检测模块参数

符号	参数说明
L	滑动窗口长度
T_{poll}	信息采集周期
h	自适应阈值与方差的系数

为了选取最优的参数组合, 首先分析触发检测模块的目标以及评估指标。在本文异常检测这个应用场景中, 主要考虑以下指标:

- (1) 错误率:

- a) 虚发警报: 将正常流量误判为异常流量触发警报, 发生这种错误的概率记为虚警率 α ;
- b) 漏发警报: 将异常流量误判为正常流量错过警报, 发生这种错误的概率记为漏警率 β 。

- (2) 检测延迟时间: 从攻击的出现 (异常点出现) 到算法检测到攻击所需要的时间记为 $t_{trigger}$

首先, 两类错误是不可避免的, 通过调节阈值可以在二者之间进行权衡: 增大阈值, α 减小, β 增大, 反之则 α 增大, β 减小。因此, 最优的阈值需要根据这两类错误造成的总损失来确定。在本文攻击检测场景中, 触发检测模块作为长期运行的模块, 目标是较严格的漏警率以及较宽容的虚警率。唯有这样才可以更好的与后续的深度检测模块结合起来, 实现高效低耗的效果。因此, 为了获得极

小的漏警率，需要将阈值设定到较小的范围。SMNA-CUSUM 算法中的阈值具有自适应性，其数值实时地随着滑动窗口内序列的方差而变化，线性系数为 h ，因此 h 越小意味着阈值越小，进一步的，漏警率越低。

图 3.7 表示了长度为 8 的时间窗中，对于某一个特征量， h 对实验结果的影响，时间窗中最后一个点为异常点。图 3.7 中 h 的值依次为 0.5、1、2 和 3，图中绿色阈值带外红色的星形点表明检测到异常，可以发现当 h 取值为 0.5 时，第三个点处发生了虚发警报，当 h 取值为 3 时，发生了漏发警报。

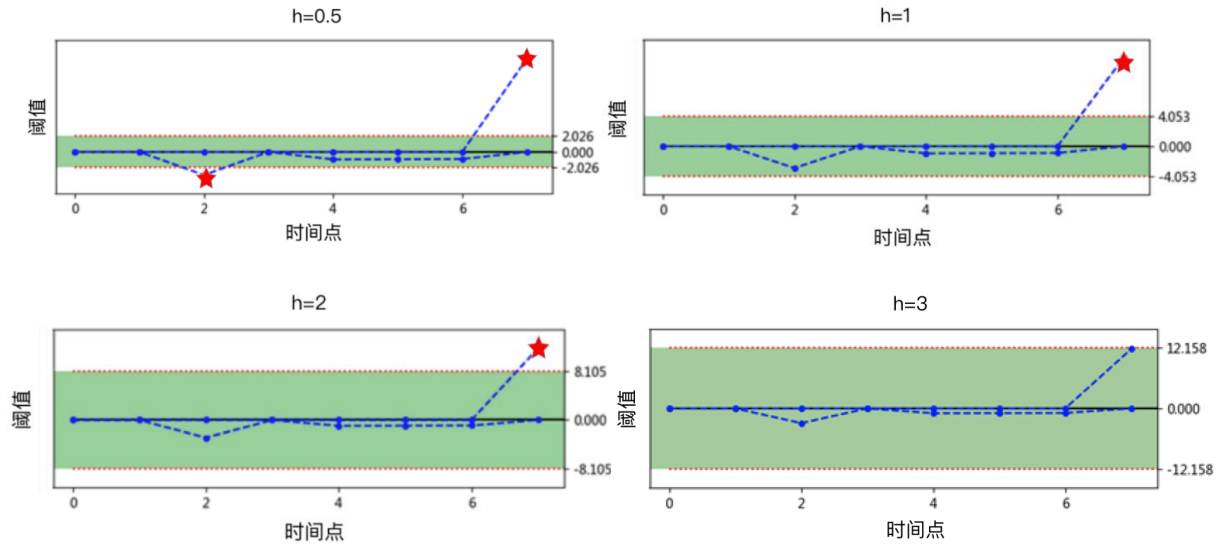


图 3.7 不同 h 值对检测结果的影响

表 3.4 所示为不同的 h 值对误检率和漏检率的影响。基于表 3.4 以及上述分析，本文选择 h 值为 1，以满足低漏警率的要求。

表 3.4 不同的 h 值对虚警率和漏警率的影响

h	虚警率(%)	漏警率(%)
0.5	36.5	0
1	12.8	0
2	4.4	7.6
3	0.4	22.9

接下来考虑检测延迟时间这项指标，由于 SMNA-CUSUM 算法的核心思想是统计累积和误差，对每个样本中出现的偏移灵敏度极高，其检测延迟时间优于其他变点检测算法如休哈特控制图（Shewhart）和指数加权滑动平均控制图（EWMA）等；而基于滑动窗口的特性使得每当新样本到达时，算法均能结合历史数据信息及时地处理新样本并做出判决，检测延迟时间与滑动窗口长度 L 无关，仅与信息采集周期 T_{poll} （新样本向量的间隔时间）相关： T_{poll} 越小即信息采集越频繁时，检测延迟的时间越短。但频繁的采集信息将造成控制器计算资源被消耗以及通信带宽被占用，与系统低消耗的目标不符，因此需要在性能与检测延迟时间两个指标间进行权衡，本文基于现有研究的经验以及大量实验，最终选取信息采集周期 T_{poll} 为 5s，滑动窗口长度 L 为 8。

3.4 深度检测模块

深度检测模块是检测方案的第二个模块，在触发检测模块发出攻击预警后立即运行。该模块采用基于决策树的 AdaBoost 分类算法对网络流量进行深度检测，识别攻击。本节将详细描述深度检测模块的流程，包括信息采集和提取、特征向量的构造以及基于决策树的 AdaBoost 算法的应用。最后将本文使用的算法与其他算法进行对比，验证了该算法的优越性。

3.4.1 信息采集和特征提取

在深度检测模块，网络流量信息的采集与触发检测模块相同，即控制器通过轮询的方式从交换机获取流表信息并实时更新，此处不再赘述。每隔时间 T_{poll} ，控制器中的流表信息采集模块获取该周期内的所有流表信息，记为矩阵 C ：

$$C = \begin{bmatrix} dpID_1 & srcIP_1 & dstIP_1 & pkgs_1 & bytes_1 & speed_1 \\ dpID_2 & srcIP_2 & dstIP_2 & pkgs_2 & bytes_2 & speed_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ dpID_i & srcIP_i & dstIP_i & pkgs_i & bytes_i & speed_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ dpID_n & srcIP_n & dstIP_n & pkgs_n & bytes_n & speed_n \end{bmatrix} \quad (3.20)$$

接下来，基于上述矩阵提取全网的流量信息。触发检测模块中提到五个特征：总流表数 $flowCount$ 、总数据包数 $pkgCount$ 、总字节数 $byteCount$ 、源 IP 地址的熵 $H(srcIP)$ 以及目的 IP 地址的熵 $H(dstIP)$ ，为了更细粒度、更全面地刻画流量特征，在此基础上，新增以下几个特征，作为 DDoS 攻击检测的依据。

不对称数据包数和不对称字节数：衡量流量的不对称性，指两个通信的 IP 地址间收发数据包的差的总数以及字节总数。若存在两个 IP 地址 a 和 b ，记 (a, b) 为一个 IP 地址对， a 发往 b 的数据包集合记为 A ， b 发往 a 的数据包集合记为 B ，则集合 A 与 B 的元素数量之差为收发数据包的差。DDoS 攻击发生时，网络中的流量将呈现明显的非对称性：攻击者将发出更多的数据包而受害者将收到更多的数据包，网络中的不对称数据包数和字节数将出现明显波动。

流表匹配率：衡量交换机中数据包与流表的匹配率，指匹配成功的流表项数与尝试匹配流表项次数的比率，该指标与网络中新出现的流的数量呈负相关。DDoS 攻击发生时，网络中出现较多尚未存在于流表项中的流，流表匹配率这一指标将明显减小。

基于上述分析，本文从流表信息中提取 8 个特征以尽可能细粒度地表示网络流量的变化，作为识别网络是否遭受攻击的依据，8 个特征为：总流表数，总数据包数，总字节数，源 IP 地址的熵，目的 IP 地址的熵，不对称数据包数，不对称字节数以及流表匹配率，其计算方法见表 3.5。

表 3.5 表征网络流量变化的特征

特征量	计算方法
总流表数	$flowCount = n$
总数据包数	$pktCount = \sum_{i=1}^n pkts_i$
总字节数	$byteCount = \sum_{i=1}^n bytes_i$
源 IP 地址的熵	$H(srcIP) = \sum_{i=1}^n p_i \log p_i$
目的 IP 地址的熵	$H(dstIP) = \sum_{i=1}^n p_i \log p_i$
不对称数据包数	$asypkgCount = \sum_{i=1}^k asypkgs_i$
不对称字节数	$asybyteCount = \sum_{i=1}^k asybytes_i$
流表匹配率	$match = \frac{1}{d} \sum_{i=1}^n \frac{matches_i}{lookups_i}$

表 3.5 中， n 表示该时间内收集到的各个交换机中流表项的总数， k 表示 IP 地址对的总数， d 表示交换机的总数， $asypkgs_i$ 表示第 i 个 IP 地址对的不对称数据包的数目， $asybytes_i$ 表示第 i 个 IP 地址对的不对称数据包的字节数目， $matches_i$ 表示第 i 个交换机匹配成功的次数， $lookups_i$ 表示第 i 个交换机尝试匹配流表的次数。

上述八个特征构成表征网络流量变化的特征向量，记为 \mathbf{D}_t ：

$$\mathbf{D}_t = \begin{bmatrix} flowCount \\ pkgCount \\ byteCount \\ H(srcIP) \\ H(dstIP) \\ asypkgCount \\ asybyteCount \\ match \end{bmatrix}. \quad (3.21)$$

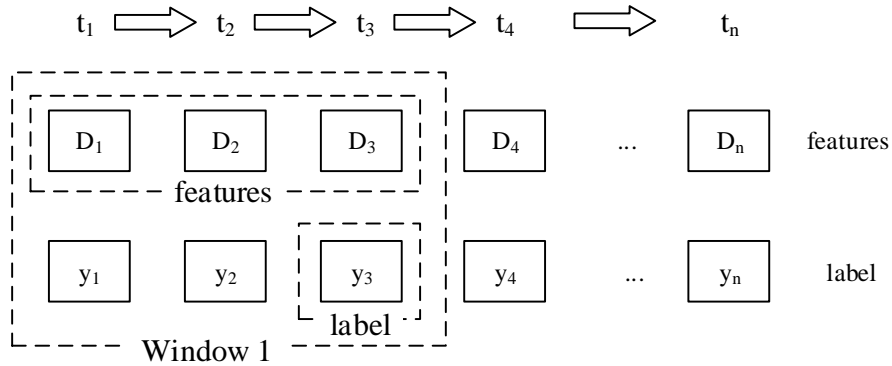
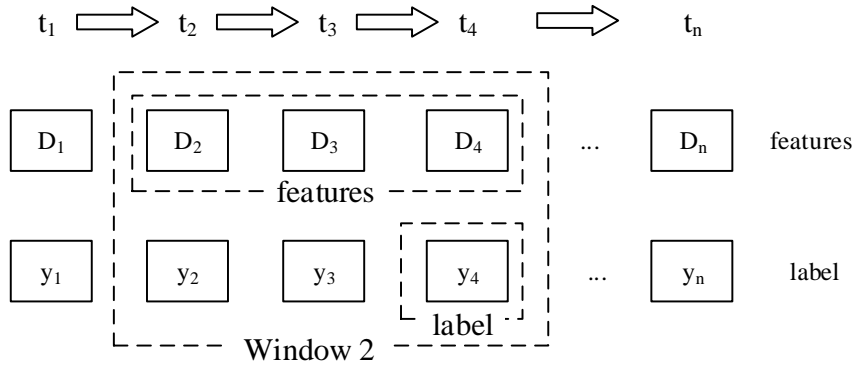
3.4.2 基于滑动时间窗的特征构造

上一节通过轮询的方式提取周期 T_{poll} 内的特征向量 \mathbf{D}_t 以刻画网络流量特征。然而，网络流量模型是一个随着时间的变化而不断改变参数的模型，仅提取某个时间段的特征而不联系其历史行为的方法，无法表现网络流量随时间变化的特性。也就是说，若直接将特征向量 \mathbf{D}_t 作为检测模块中分类算法的输入，各个特征随着时间的变化特性将被忽略，分类结果则达不到理想的精度。对于多维时间序列的分类问题，典型的解决方案包括长短期记忆神经网络（LSTM, Long Short-Term Memory）与时间递归神经网络（RNN, Recurrent Neural Network），但上述算法需要海量数据进行训练，并且更适用于原始序列而非人工提取的特征序列，因此，不适用于本文的环境。

根据 SDN 网络流量的变化特性以及时间序列相关的理论基础，本文提出基于滑动时间窗的特征

构造方法：通过记录一个时间窗内特征向量的变化情况来捕捉其随时间的变化特性，构造出包含时间变化特性的新的特征向量。“时间窗”指的是时间维度上的一个集合，定义时间窗的长度为 L ，对 t 时刻的网络流量进行攻击检测时，使用由 $t-1, t-2, \dots, t-L$ 这 L 个周期对应的 L 个特征向量组合而成的新的特征向量。“滑动”指的是时间窗将随着时间不断向前移动，攻击检测时，每隔时间 T_{poll} ，时间窗向后移动一个单位。

如图 3.8 所示，假设窗口长度 $L=3$ ，从时间 t_1 开始计时，在每个时间周期 T_{poll} 内采集并提取 8 维的特征向量 D_t ，经过三个周期提取到特征向量 D_1 、 D_2 和 D_3 ，基于滑动时间窗的规则，将这三个向量合并，构造出 24 维的特征向量来表征网络流量模型， y_3 表示这个 24 维特征向量对应的标签，即攻击是否存在。此后每隔时间 T_{poll} ，时间窗向右滑动一个单位，将新采集的 8 个特征与历史数据 16 个特征相结合，构成新的 24 维特征向量，输入至分类算法，判定当前最新时间间隔 T_{poll} 内网络中是否出现攻击。图 3.8 与图 3.9 表示时间窗滑动一个单位的过程。

图 3.8 $t=3$ 时的时间窗图 3.9 $t=4$ 时的时间窗

3.4.3 基于 AdaBoost 算法的攻击检测

选择合适的分类算法对构造好的特征向量进行分类判决，在线检测网络是否发生异常，是深度检测模块的核心。本文使用机器学习的方法，模拟网络的正常流量以及攻击流量作为训练样本，根据样本的特征向量及标签来训练分类算法，最终基于训练好的模型在网络中进行实时地攻击检测。

机器学习中常见的分类算法包括：朴素贝叶斯，逻辑回归，支持向量机，决策树等等，其核心思想均在第二节中做过简单介绍。一般而言，朴素贝叶斯算法最为简单易行，但需要满足前提条件即相互独立性，因而无法学习不同特征之间的相互作用，而本文构造的 24 维流量特征不满足相互独

立性这一条件；逻辑回归算法在特征满足近似线性且结果线性可分时表现优异，但本文网络流量特征无法保障这个前提条件；支持向量机算法则通过使用非线性的核函数来处理线性不可分问题，但其复杂度较高、效率较低；决策树算法既可以处理相关的特征，且无需担心是否线性可分这一问题，同时，相比于支持向量机算法，其复杂度更低且速度更快，但存在容易过拟合的缺点。为了解决决策树算法这个问题，同时获得更好的分类效果，本文研究了集成学习的理论，使用多个决策树组合来进行分类，即基于决策树的 AdaBoost 算法。

AdaBoost 全称 Adaptive Boosting，即自适应提升算法，属于集成学习中的一种典型算法。该算法的产生源于以下两个推论：（1）对于分类问题而言，给定一个训练样本集，求得精度较低的分类规则（弱分类器）要比求得高精度的分类规则（强分类器）要容易很多；（2）提升算法可以将若分类器提升为强分类器，从弱分类算法触发，反复学习得到一系列弱分类器，将这些分类器进行组合并构成一个强分类器。AdaBoost 算法有以下优势：（1）分类精度高：结合了多个弱分类器的优势；（2）可拓展性强：在 AdaBoost 的框架下，可以灵活使用各种回归分类模型来训练弱分类器；（3）构造简单：作为简单的二元分类器时，不需要调节过多复杂参数；（4）不容易发生过拟合现象。

AdaBoost 算法的核心思想是：按顺序拟合一系列弱分类器，后一个分类器建立在前一个弱分类器变换后的数据上，最终通过对所有弱分类器采取一个加权多数表决的方法来确定预测结果。最初，从初始训练集中训练出基分类器；根据该分类器的表现，对训练样本进行权值调整，使得该分类器预测错误的样本受到更多的关注；再根据调整后的样本集训练出下一个基分类器；循环上述步骤，直至基分类器的数据达到事先的设定值；最终将所有得到的基分类器进行加权结合，构造出强分类器。AdaBoost 算法的误差分析及其推导过程此处不做赘述，可参考文献^[72]。

本文所述的 SDN 环境中，通过基于滑动时间窗的特征构造方法，得到由所有样本点组成的二分类训练数据集 T ：

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}, \quad (3.22)$$

其中， N 表示样本点个数，每个样本点由特征向量 x_i 与标记 y_i 组成。特征向量 $x_i \in \chi \subseteq R^n$ ，对应上文提取的 24 维特征向量；标签 $y_i \in \psi = \{-1, +1\}$ ，对应网络中是否发生 DDoS 攻击。AdaBoost 分类算法的攻击检测流程见表 3.6。

表 3.6 AdaBoost 算法

输入：训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ，其中， $x_i \in \chi \subseteq R^n$ ， $y_i \in \psi = \{0, 1\}$
输出：最终分类器 $G(x)$

(1) 初始化训练数据的权值分布

$$D_1 = (w_{11}, \dots, w_{1i}, \dots, w_{1N}), w_{1i} = \frac{1}{N}, i = 1, 2, \dots, N, \quad (3.23)$$

(2) For $m=1, 2, 3, \dots, M$

a) 用弱分类器训练具有权值分布 D_m 的训练数据集，得到基分类器

$$G_m(x): \chi \rightarrow \{-1, +1\}, \quad (3.24)$$

- b) 计算基分类器 $G_m(x)$ 在训练数据集上的分类误差

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^N w_{mi} I(G_m(x_i) \neq y_i), \quad (3.25)$$

其中, w_{mi} 表示第 m 轮中第 i 个特征的权值, $\sum_{i=1}^N w_{mi} = 1$.

- c) 计算基分类器 $G_m(x)$ 的系数

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m}, \quad (3.26)$$

其中可见, α_m 随着 e_m 的减小而增大, 因而分类误差率越小的基分类器在最终分类器中的权重越大。

- d) 更新训练数据集的权重分布

$$D_{m+1} = (w_{m+1,1}, \dots, w_{m+1,i}, \dots, w_{m+1,N}), \quad (3.27)$$

$$w_{m+1,i} = \frac{w_{m,i}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), i = 1, 2, \dots, N, \quad (3.28)$$

该公式也可以写成:

$$w_{m+1,i} = \begin{cases} \frac{w_{m,i}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{m,i}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases}, \quad (3.29)$$

Z_m 是规范化因子, 使得 D_{m+1} 成为概率分布,

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i)), \quad (3.30)$$

由此可见, 被基分类器 $G_m(x)$ 误分类样本的权值得到了扩大, 而被正确分类样本的权值得到了缩小; 因此, 下一轮学习中, 误分类样本起了更大的作用。随着迭代的进行, 很难预测的样本将不断被调整。

- (3) 基分类器加权表决, 构建最终的分类器 $G(x)$

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x), \quad (3.31)$$

$$G(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right), \quad (3.32)$$

由此, 基于 AdaBoost 的攻击检测方案如图 3.10 所示, 包括训练和检测两个阶段。

- (1) 训练阶段: 使用训练集训练出分类器。在系统投入使用前, 采集 SDN 网络中的正常流量信息以及模拟攻击时的流量信息作为训练样本, 提取特征向量并结合对应的标签, 送入基于决策树的 AdaBoost 分类算法, 训练分类器;
- (2) 实时检测阶段: 使用训练好的分类器进行攻击检测。该检测模块由上一节触发检测模块发起的预警信号启动。一旦启动, 当前的网络流量特征向量被送入分类算法, 进行判决并输出结果。

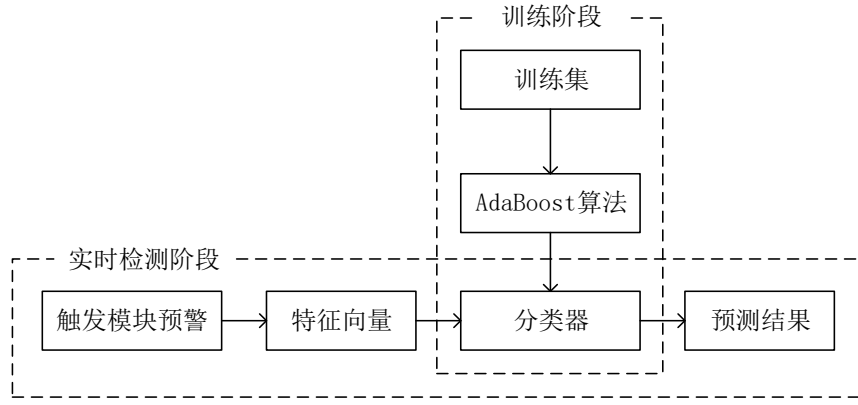


图 3.10 基于 AdaBoost 的攻击检测

下面分析深度检测模块所涉及的参数。基于决策树的 AdaBoost 算法的参数分为两个部分：AdaBoost 框架参数以及弱分类器参数。

AdaBoost 框架参数：

- (1) 弱分类器：本文使用基于决策树的 AdaBoost 算法，弱分类器为决策树。
- (2) 算法选择：AdaBoost 分类算法包括 SAMME 和 SAMME.R 两种方案。主要区别在于弱分类器权重的度量：SAMME 用对样本集分类效果作为弱分类器权重，而 SAMME.R 使用了对样本集分类的预测概率大小作为弱分类器权重。由于 SAMME.R 使用了概率度量的连续值，迭代一般比 SAMME 快，因此此处使用 SAMME.R 方案。
- (3) 弱分类器数量 M ：弱分类器的最大迭代次数，即最大的弱分类器的个数。若分类器数量太小，容易欠拟合；分类器数量太大，容易过拟合。在实际调参过程中，将该参数和下面介绍的学习速率一起考虑。
- (4) 学习速率 ν ：即每个弱分类器的权重缩减系数。为了防止过拟合，加入正则化项，此时强分类器的迭代公式变为

$$f_k(x) = f_{k-1}(x) + \nu \alpha_k G_k(x) \quad (3.33)$$

ν 的取值范围为 $0 < \nu \leq 1$ 。对于同样的训练集拟合效果，较小的 ν 意味着需要更多的弱分类器的迭代次数因此，学习速率和弱分类器数量这两个参数同时调节，最终决定算法的拟合效果。

弱分类器参数，即决策树算法的参数：

- (1) 划分时考虑的最大特征数
- (2) 决策树最大深度
- (3) 内部节点再划分所需最小样本
- (4) 叶子节点最少样本数
- (5) 叶子节点最小的样本权重和
- (6) 最大叶子节点数

决策树算法涉及的 6 个参数均是对决策分类过程进行条件限制的参数，在特征数量较多的情况下，一般通过交叉验证的方法不断测试以对这些参数进行设置。由于本文构造的向量维度较低，无需考虑上述 6 个参数。

基于以上分析，整个深度检测模块所涉及到的待调节参数见表 3.7：

表 3.7 深度检测模块的参数

符号	参数说明
L	时间窗长度
M	弱分类器数量
ν	学习速率

为了选取最优的参数组合，首先分析深度检测模块的目标以及评估指标。该模块的目标是在性能消耗可接受的范围内尽可能毫无遗漏地检测出攻击，评估的核心是分类算法的性能指标。

首先给出 TP 、 FN 、 FP 、 TN 等四个概念的定义，见表 3.8。

表 3.8 TP 、 FN 、 FP 、 TN 的定义

实际情况	预测结果	
	正例 (+1)	反例 (-1)
正例 (+1)	TP (真正例)	FN (假反例)
反例 (-1)	FP (假正例)	TN (真正例)

由表 3.8 可知，实际样本的反例总数 $N=FP+TN$ ，正例总数 $P=TP+FN$ ，总样本数目 $C=N+P$ 。

基于上述定义，下面给出深度检测模块完整的评估方式。

(1) 准确率：描述模型正确分类的样本数与总样本数之比，计算公式为：

$$Accuracy = \frac{TP + TN}{P + N}. \quad (3.34)$$

(2) 精确率：衡量模型预测正样本的准确性，计算公式为：

$$Precision = \frac{TP}{TP + FP}. \quad (3.35)$$

(3) 召回率/查全率：衡量模型预测正样本的可信性，计算公式为：

$$Recall = \frac{TP}{TP + FN}. \quad (3.36)$$

(4) F_1 和 F_β 度量： F_1 是精确率和召回率的调和均值，认为精确率和召回率的权重是一样的，但有些场景下，可能认为其中一个更加重要，则使用 F_β ，通过调整参数 β 获得更好的评估结果， β 大于 1 表示召回率的影响更大，小于 1 表示精确率的影响更大，计算公式为：

$$\frac{1}{F_1} = \frac{1}{Precision} + \frac{1}{Recall}, \quad (3.37)$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall}, \quad (3.38)$$

$$F_\beta = \frac{(1 + \beta^2) * Precision * Recall}{(\beta^2 * Precision) + Recall}. \quad (3.39)$$

在本文深度检测这个场景下，对分类器的召回率这项指标要求极高，即尽可能保证每次攻击产生都能检测出来。经过参数调试，本文最终选取时间窗长度为 3，弱分类器数量为 100，学习速率为

1。

将基于决策树的 AdaBoost 分类算法替换成其他分类算法，基于 1000 个训练集及 1000 个测试集进行实验，其性能指标如表 3.9 所示。由表格可见，本文使用的基于决策树的 AdaBoost 算法效果最佳，准确率达 97.7%，召回率达 98%。

表 3.9 不同分类算法的对比

分类器	Accuracy	Precision	Recall	F_1
AdaBoost	0.9769	0.98	0.98	0.98
KNN	0.9394	0.94	0.93	0.94
LR	0.9643	0.95	0.96	0.96
DT	0.9721	0.97	0.97	0.97
RandomForest	0.8807	0.85	0.83	0.84
MLP	0.3994	0.47	0.40	0.37

3.5 本章小结

本章针对现有检测方案中，检测延时与系统开销间的矛盾导致检测周期难以确定的问题，设计了由触发检测和深度检测相结合的联合检测方案。触发检测模块中，低开销的 SMNA-CUSUM 算法通过跟踪基于全网流量特征的多维序列来进行粗粒度的异常预警，调节其阈值系数 h 保障极低的漏警率。深度检测在收到预警信号后启动，通过基于时间窗的特征构造方案将流量随时间变化的特性加入特征向量中，提升了检测的准确率，对于分类算法的选择，基于实验和分析选择效果最佳的 AdaBoost 算法，仿真表明其性能优于其他算法，精确度达到 97.7%。整个联合检测方案具有开销小、检测准确率高的优点。

第四章 基于 SDN 的 DDoS 防御方案

4.1 引言

对 SDN 中 DDoS 防御技术的研究大多是建立在攻击检测的基础之上。在完成攻击检测的部署后,网络面临的主要挑战是如何重新配置以缓解 DDoS 攻击。而 SDN 集中控制和动态可编程的特性为制定新的网络规则和实时更新网络提供了极大的便利。

现有防御方案的核心思想是利用 SDN 数控分离的架构,在控制器检测到攻击后立即制定响应措施,并向交换机下发相应的流表以指挥其执行响应措施,达到缓解攻击的效果。

常见的响应措施^[34, 36-39]包括丢弃数据包、限制流速、阻断端口以及重定向流量等等。在 Giotis^[12]等人提出的方案中,使用 IP 地址和端口号这两项去匹配交换机中接收到的数据包,成功匹配的数据包意味着其属于攻击流量,因此对其执行丢弃操作;另外,该文献还提出白名单的机制以提升网络的可用性。文献^[40]通过对具有攻击特征的流量进行速率限制以达到阻断攻击的效果。在文献^[25]中,被识别为攻击流的数据包将经历转发模块、丢弃模块和更新模块等三个模块,保障了系统在完成丢弃攻击流量的同时,记录下攻击相关的信息以便进一步分析。文献^[12, 28, 29, 38, 40]均证明,在 SDN 中通过控制器向交换机下发流表以实现实时转发或阻断攻击流量的方案可行性较高且效果明显,但在完成攻击阻断后,大量无效流表被存留在交换机中,这将消耗交换机的内存资源甚至导致流表溢出现象。因此,Cui^[34]等人提出,在完成攻击阻断后,通过控制器向交换机下发流表删除的命令来删除与阻断攻击相关的流表项,实现交换机内存的释放,但该文献并未说明具体的删除方式以及删除时间。

综上,现有的研究存在以下问题:

(1) 攻击阻断后,大量无效流表占用交换机的资源。交换机作为 SDN 中数据平面内的一个轻量级的转发节点,内存有限,无效的流表将造成其内存的消耗甚至导致流表溢出。

(2) 网络的可用性有待提升。一方面,遭受攻击的网络将发生拥堵现象,如何降低合法请求受到的负面影响、提升整个网络的效率成为关键问题;另一方面,执行攻击阻断措施时,如何减小合法流量被过滤的概率这一问题尚未得到解决。

基于上述问题,本文提出了一种基于路由动态调整和溯源主动阻断相结合的防御方案。动态路由模块目的是利用网络资源来分散和吸收攻击流量,其核心是基于带宽的最优路由算法,即优化的 K 最短路径算法。该算法根据流量和带宽信息制定路由方案,通过控制器下发流表的方式将该路由方案告知交换机,由此整个网络可以动态地根据拓扑和流量的变化灵活地进行路由,在面临攻击时,其可用性和抗压能力得以提升。攻击阻断模块包括基于主机行为的攻击溯源阶段和基于动态流表的报文实时过滤阶段:溯源阶段,提取基于主机行为的特征 8 元组,使用 AdaBoost 算法基于 8 元组来识别异常主机;过滤阶段,提出了一种基于流表保护机制的报文过滤方案,该方案制定针对攻击源的动态流表来实时过滤攻击报文,并为流表项定义较短的闲置超时时间,自动删除无效流表以释放资源,提出全局白名单机制来保护合法主机被误判和阻断,增强系统的灵活性与网络的可用性。

4.2 防御系统架构

图 4.1 描述了 DDoS 防御方案的整体架构，分为动态路由和攻击阻断两个主要模块，此外还包括负责与控制平面进行交互的流表管理和消息发送模块。

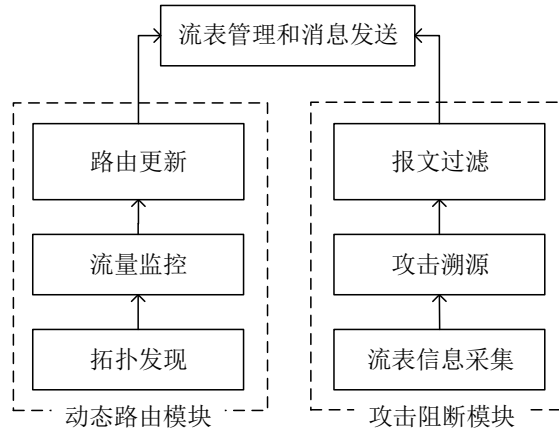


图 4.1 防御系统架构

动态路由模块根据不断变化的网络拓扑，基于最优带宽的原则实时更新路由规则，提升通信质量。该系统主要包括三个模块：拓扑发现模块，流量监控模块和动态更新模块。拓扑发现与流量监控模块负责构建全网拓扑图并统计带宽信息；路由更新模块的核心是基于带宽的最优路由算法：在 K 最短路径算法的基础上进行改进，基于网络拓扑和带宽信息，计算带宽最优的路由方案。在获取路由方案后，流表管理与消息发送模块负责制定流表并发送消息至交换机，告知新的转发规则。

攻击阻断模块目的是发现攻击后在最短时间内阻止攻击。该系统包括三个模块：流表信息采集模块、攻击溯源模块以及报文过滤模块。流表信息采集模块通过轮询的方式收集全网的流量信息。攻击溯源模块根据全网的流量信息，提取基于源地址和目的地址的特征 8 元组来表示主机的行为特征，并使用 AdaBoost 分类算法基于 8 元组来识别异常主机，定位攻击源。报文过滤模块进一步确认目标主机以及是否阻断其通信行为，其中，基于黑名单的流表保护机制针对攻击源的流表项定义较短的闲置超时时间，保证无效流表被自动删除，减小了交换机被无效流表占满导致溢出的可能，全局动态白名单机制提升了系统的可用性以及网络的自由度。此外，流表管理与消息发送模块负责制定动态流表并发送消息至交换机，命令交换机丢弃来自攻击源的数据包以实现攻击阻断。

4.3 动态路由模块

第二章指出，DDoS 的防御方式可以从以下三个方面考虑：流速限制、报文过滤、系统重构。本节从系统重构，即改变受害者网络拓扑或增加系统资源的角度出发，设计了基于带宽的动态最优路由解决方案，以适应网络流量、拓扑结构的变化，同时在网络面临攻击时，利用网络资源来分散和吸收攻击流量，提升网络性能以及抗压能力。

基于带宽的动态最优路由方案采用优化的 K 最短路径算法来实时计算路由，并通过下发流表的方式通知交换机执行路由规则，实现分散 DDoS 流量的目的。该方案主要包括三个模块：拓扑发现模块，流量监控模块以及基于优化的 K 最短路径算法的路由模块。控制器首先通过拓扑发现模块获

取交换机、链路、主机等信息，生成网络拓扑图并实时更新；其次采用轮询的方式统计全网的流量和带宽信息；获取这些关键信息后，基于路由模块中优化的 K 最短路径算法，实现基于带宽的最优路由制定；最终控制器将路由规则下发至交换机。由此，整个网络可以动态的根据拓扑和流量的变化更新路由方式，实现基于带宽的最优路由。

4.3.1 拓扑发现

拓扑发现指的是控制器发现网络中各种资源的过程，资源包括交换机、链路、终端主机等等，拓扑发现是实现动态路由的基础服务。下面阐述交换机发现、链路发现以及主机发现的过程。

(1) 交换机发现

首先，交换机向控制器发起一个 TCP 连接请求；控制器收到请求后，向该交换机发送 FEATURE_REQUEST 类型的消息并等待回复；接着，交换机将其特征信息（交换机 ID、端口属性和缓冲区数量等）封装至 FEATURE_REPLY 消息中，发送至控制器；最终，控制器收到消息，记录交换机的完整信息，完成交换机的发现过程。

(2) 链路发现

一旦交换机与控制器成功建立连接，控制器将定期地给交换机发送消息，令其从每个端口泛洪链路层发现协议（LLDP, Link Layer Discovery Protocol）和广播域发现协议（BDDP, Broadcast Domain Discovery Protocol）数据包，发现协议的数据包中包含发送方交换机的 ID 以及端口号。其中，LLDP 用于发现直连的 OpenFlow 设备，BDDP 用于发现非 OpenFlow 设备，而本文不涉及非 OpenFlow 设备，忽略 BDDP 数据包。

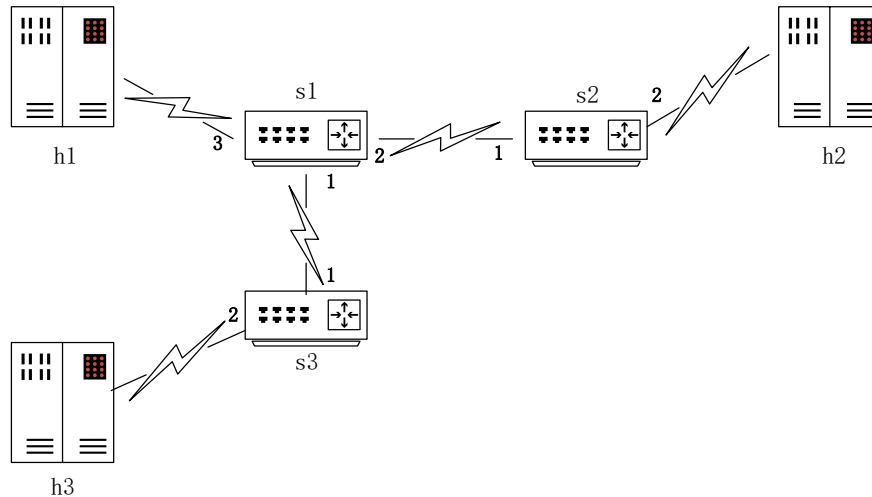


图 4.2 链路发现和主机发现示例拓扑

图 4.2 所示的网络拓扑，s1，s2，s3 均连接至同一个控制器，且各自连接一个终端主机。

- 当 s1，s2，s3 完成与控制器的连接建立后，开始泛洪 LLDP 数据包；
- 由 s1 的 1 号端口发出的数据包将抵达 s3 的 1 号端口，并由 s3 将该数据包转发至控制器，控制器收到数据包后，根据收发双方的信息（交换机 ID 和端口号）得出该链路信息并记录；
- 同理，由 s2 的 1 号端口发出的数据包将抵达 s1 的 2 号端口，控制器将记录下这条链路；依次类推，控制器可以发现全部的链路信息，从而发现完整的网络拓扑。

(3) 主机发现

控制器在完成交换机发现与链路发现的基础上，进行主机发现。图 4.2 所示网络，假设 h1 向 h2 发送 ping 请求。

- a) 当请求的数据包到达 s1 时，由于之前没有安装匹配的流表，s1 将把该消息转发至控制器；
- b) 控制器第一次收到源自 h1 的消息，发现 s1 的 3 号端口是 h1 的接入点，由于此时不知道 h2 的接入点，则命令 s1 泛洪该数据包；
- c) 在数据包被泛洪的过程中，s2 和 s3 将收到数据包。假设 s2 先收到数据包，由于之前没有安装匹配的流表，s2 将把该消息转发至控制器，此时控制器依旧不知道 h2 的接入点，则命令 s2 再次泛洪该数据包；
- d) 在这次泛洪过程中，收到数据包的 h2 将发出回应；s2 的 2 号端口收到 h2 的应答消息时，将转发至控制器；
- e) 收到应答消息的控制器将成功发现 s2 的 2 号端口是 h2 的接入点，同时，由于控制器已经记录 h1 的接入点，应答消息的转发路径则被确定：从 s2 的 2 号端口至 1 号端口，再由 s1 的 2 号端口至 3 号端口；
- f) 最终，控制器将向交换机 s1 和 s2 下发合适的流表以告知转发规则，且通知 s2 转发来自 h2 的应答消息。

根据上述流程，控制器可以获取到网络中的交换机和端口信息、链路信息、主机接入信息等。同时，控制器通过实时检测网络变化的异步事件来更新网络资源信息，以保障获取到最新的拓扑信息。

4.3.2 流量监控和带宽获取

在完成拓扑发现的基础上，设计流量监控模块。流量监控主要包括两个功能：第一，监控 SDN 网络中的流量，计算出实时的流量统计数据；第二，获取网络资源带宽，以作为底层服务为实现基于带宽的动态路由提供带宽信息。

(1) 流量监控

如第二章所言，根据 OpenFlow 协议，每个 OpenFlow 标准端口以及每个流表项均包含一个计数器，用以统计流量的一些信息，例如活动表项、查找次数、发送包数等。根据上述特性，在控制器中，通过轮询的方式，即周期性地调用请求流表统计信息模块，下发统计信息请求报文，同时监听对应的回复请求报文事件，在收到交换机的回复信息时统计并更新数据，完成流量统计。图 4.3 所示表示流量监控中控制器和交换机的交互过程，控制器发起端口信息和流信息的请求，交换机收到请求后回复对应的信息。

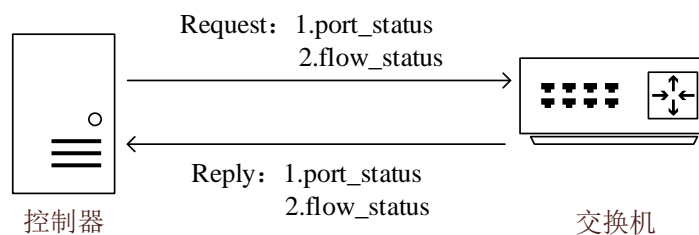


图 4.3 流量监控

表 4.1 和表 4.2 所示为本文统计的两部分信息：交换机端口信息和流表项统计信息。

表 4.1 交换机端口统计信息

统计量	Bits	统计量	Bits
Received Packets	64	Transmit Errors	64
Transmitted Packets	64	Receive Frame Alignment Errors	64
Received Bytes	64	Receive Overrun Errors	64
Transmitted Bytes	64	Receive CRC Errors	64
Receive Drops	64	Collisions	64
Transmit Drops	64	Duration (seconds)	32
Receive Errors	64	Duration (nanoseconds)	32

表 4.2 流表项统计信息

统计量	Bits
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32

(2) 带宽获取

一条链路的带宽由两个端口的能力决定，本文通过端口的流量来计算链路的流量。表 4.1 所示已统计端口信息，从统计量中可以获取到收发的包数、字节数以及统计持续的时间。将两个不同时间的统计消息的字节数相减，再除以两个统计时间差，则可以得到传输速率；将端口最大带宽减去当前流量带宽，则得端口剩余带宽。公式如下：

$$Speed = \frac{(rb[t_2] + tb[t_2]) - (rb[t_1] + tb[t_1])}{t_2 - t_1}, \quad (4.1)$$

$$FreeBw = MaxBw - Speed, \quad (4.2)$$

其中， $Speed$ 表示传输速率， $FreeBw$ 表示剩余带宽， $rb[t]$ 表示 t 时刻接收的比特数， $tb[t]$ 表示 t 时刻发出的比特数。

4.3.3 基于优化的 K 最短路径算法的动态路由

本节提出优化的 K 最短路径算法，实现基于带宽的最优路由计算。

首先分析 K 最短路径算法。K 最短路径问题是最短路径问题的扩展，对于网络图来说，最短路径是指源点和终点之间所经过边上的权值之和最少的路径。很多问题仅运用最短路径方法时无法满足需求，例如对备用路径的需求，K 最短路径算法便应运而生。

K 最短路径算法有三种常见的解决思路：递推法、直接法和综合法。权衡不同算法的复杂度和适用场景后，本文选择使用 Yen 算法^[73]来进行改进，Yen 算法属于递推法的一种，采用了偏离路径算法思想，适用于非负权边的有向无环图结构。

Yen 算法的计算过程分为两部分：（1）算出第 1 条最短路径 P^1 ；（2）在此基础上依次算出其他

的 $K-1$ 条最短路径: $P^2, P^3, \dots, P^i, \dots, P^K$ 。在求时 P^{i+1} , 将 P^i 上除了终止节点外的所有节点都视为偏离节点, 并计算每个偏离节点到终止节点的最短路径, 再与之前的 P^i 上起始节点到偏离节点的路径拼接, 构成候选路径, 进而求得最短偏离路径。

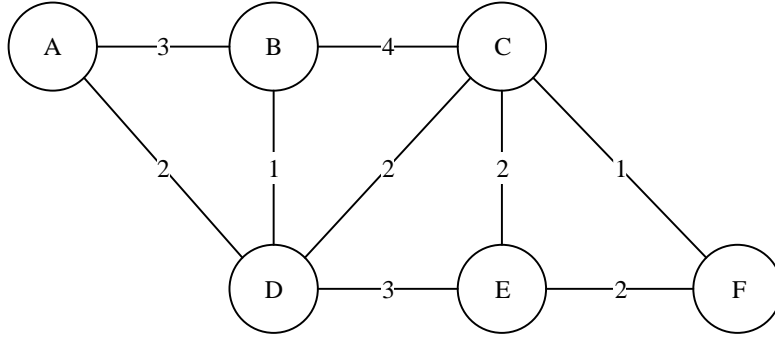


图 4.4 K 最短路径示例拓扑

如图 4.4 所示网络拓扑, 共有 A、B、C、D、E、F 等 6 个节点, 每条边的权重标记如上, 源点为 A, 终点为 F, 记偏移路径集合为 B , 假设 $K=3$, Yen 算法的步骤如下:

- (1) 通过 Dijkstra 算法计算得到最短路径: A-D-C-F, 花费为 5, 该路径作为 K 最短路径的第一条路径, 记入结果集, P^1 : A-D-C-F;
- (2) 将 P^1 作为迭代路径, 开始第一次迭代:
 - a) 部分迭代路径: A 路径中, A 点为激励点, 将 A-D 路径之间的权值设为无穷大, 使用 Dijkstra 算法, 得到路径 P_1^2 : A-B-C-F, 花费为 8, 将 P_1^2 加入第二条最短路径 P^2 的候选集 B ;
 - b) 部分迭代路径: A-D 路径中, D 为激励点, 将 D-C 路径之间的权值设为无穷大, 使用 Dijkstra 算法, 得到路径 P_2^2 : A-D-E-F, 花费为 7, 将 P_2^2 加入第二条最短路径 P^2 的候选集 B ;
 - c) 部分迭代路径: A-D-C 路径中, C 为激励点, 将 C-F 路径之间的权值设为无穷大, 使用 Dijkstra 算法, 得到路径 P_3^2 : A-D-C-E-F, 花费为 8, 将 P_3^2 加入第二条最短路径 P^2 的候选集 B ;
 - d) 迭代完成, P^2 的候选集合 B 中有三条路径: A-B-C-F, A-D-E-F, A-D-C-E-F; 选出花费最小的偏离路径 A-D-E-F, 记为 P^2 ;
 - e) 同时将其移出 B 集合。
- (3) 将 P^2 作为迭代路径, 开始第二次迭代:
 - a) 部分迭代路径: A 路径中, A 点为激励点, 将 A-D 路径之间的权值设为无穷大, 使用 Dijkstra 算法, 得到路径 P_1^3 : A-B-C-F, 已存在于 B 集合, 故不存在偏移路径;
 - b) 部分迭代路径: A-D 路径中, D 点为激励点, 将 D-C 和 D-E 路径之间的权值设为无穷大 (注意, 这里设置两条路径的权值原因是这两条路径分别存在于 P^1 和 P^2 中), 使用 Dijkstra 算法, 得到路径 P_2^3 : A-D-B-C-F, 花费为 8, 将 P_2^3 加入第三条最短路径 P^3 的候选集 B ;
 - c) 部分迭代路径: A-D-E 路径中, E 点为激励点, 将 E-F 路径之间的权值设为无穷大, 经计算发现, 不存在偏移路径;
 - d) 迭代完成, P^3 的候选集中有三条路径: A-B-C-F, A-D-C-E-F, A-D-B-C-F;

- e) 由于三条路径花费均为 8, 则根据最小节点数进行判断, 选出偏离路径 A-B-C-F, 记为 P^3 ;
- (4) 此时, 已选出三条最短路径 P^1 , P^2 , P^3 , 分别是: A-D-C-F, A-D-E-F, A-B-C-F, 算法结束。

为了实现基于带宽的最优路由, 最直接的算法如下: 首先, 获取网络链路的剩余带宽, 然后从源头开始, 选取途经路径中带宽最大的路径。由于一条链路中的最大剩余带宽取决与剩余带宽最小的那一条, 若使用贪心算法逐个排除, 很可能计算错误, 所以每遇到一个分支就需要选择一个路径, 并保存其他未选择的路径数据; 每一个节点都需要对所有的数据进行对比, 从而选择当下最优的路径, 直至所有的链路都比较完成。该算法可以通过修改 Dijkstra 算法完成, 但复杂度较大, 效率很低。

因此, 本文提出基于优化的 K 最短路径算法的路由方案: 首先根据拓扑信息基于跳数计算出最短的 K 条路径, 然后从这些路径中选择可用带宽最大的路径。其中, 最短 K 路径算法以及链路的带宽数据在上文均有详细的阐述。同时, 该方案可以通过设置 K 的不同数值, 在复杂度和精度之间进行权衡, 以满足不同系统的需求。综上所述, 实现该路由方案的基本步骤如下:

- (1) 基于拓扑发现、流量监控和带宽获取模块, 获取全网拓扑结构及流量带宽信息;
- (2) 通过 K 最短路径算法, 求出基于跳数的 K 条最短路径;
- (3) 比较最短的 K 条路径中各路径的剩余带宽, 选择最优路径, 剩余路径为备份路径;
- (4) 基于路径信息, 控制器生成流表项并下发至交换机, 完成路由更新。

图 4.5 描述了将基于带宽的动态路由方案应用到 SDN 网络中的流程:

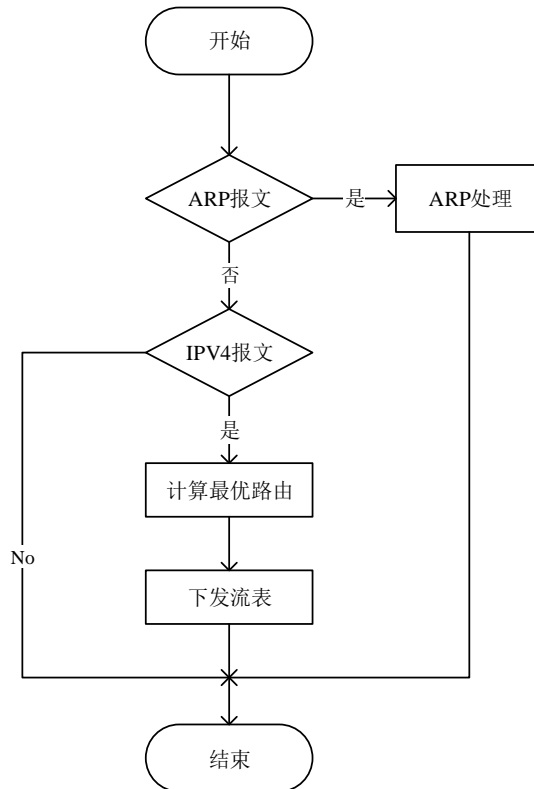


图 4.5 SDN 中的动态路由过程

如图 4.5 所示, 在 SDN 网络进行初始化时, 由于未进行 DHCP 分配, 控制器无法发现沉默的主机, 则无法掌握主机的 IP/MAC 信息。所以第一步需要处理的数据包是 ARP; 完成 ARP 数据包的处理后, 控制器则掌握了主机的接入信息以及网络信息, 当接收到新的数据包时, 则可根据源地址和目的地址查询到接入交换机, 再依据拓扑信息, 通过优化的 K 最优路由算法, 获取最优路径; 最后, 控制器以流表的形式将路由规则下发至交换机中, 交换机依据规则执行路由转发功能。

4.4 攻击阻断模块

攻击阻断模块由检测系统触发启动。基于“找到攻击者、找到所在的交换机、阻止攻击”的思路, 本文提出的阻断方案包括两大阶段: 基于主机行为的攻击溯源阶段, 基于动态流表的报文实时过滤阶段。

攻击溯源阶段利用 SDN 集中控制的特性, 提取基于主机行为的特征, 使用 AdaBoost 分类算法基于特征识别异常主机, 定位攻击源 IP 地址, 并根据全网拓扑找到其所在交换机。报文过滤的核心即通过控制器向指定交换机下发针对攻击源的动态流表, 命令交换机丢弃攻击数据包, 以实现攻击阻断, 该阶段由三个模块构成: 流表构造和消息发送模块、基于黑名单的流表保护模块以及全局动态白名单模块, 三个模块实时交互、分工合作, 完成整个报文过滤的工作。

图 4.6 描述了攻击阻断系统的工作流程。首先, 攻击溯源模块接收到攻击警报后, 定位攻击源, 获取攻击者 IP 和交换机 ID。然后, 控制器查询白名单数据库, 若该 IP 地址存在于白名单中, 则不触发其他模块, 结束此次阻断; 反之, 则表示攻击存在, 基于黑名单的流表保护模块更新黑名单, 新增攻击者 IP 地址。最后, 流表构造和消息下发模块完成动态流表项的构造以及 C-S (Controller-to-Switch) 消息的下发, 对应的交换机接收消息后将根据流表项的命令执行数据包丢弃操作, 完成攻击阻断。

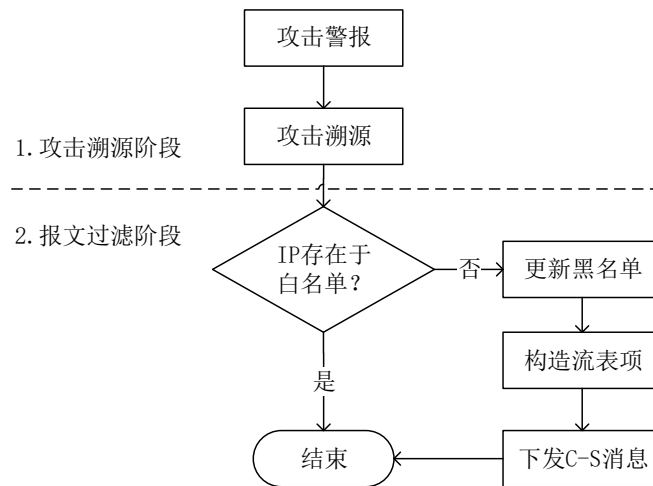


图 4.6 攻击阻断模块

4.4.1 基于主机行为的攻击溯源

攻击溯源的目的是定位到攻击源, 即攻击者的 IP 地址及其所连接的交换机。在攻击检测阶段, 其检测结果以及采集的网络模型相关信息将被传递至攻击溯源模块。利用网络流量信息, 溯源模块将深入分析主机的行为并通过机器学习的方法基于训练好的模型来对主机进行分类, 识别异常主机,

定位攻击源；此外，溯源模块还将基于网络的拓扑信息，找到攻击源所在的交换机。下面将阐述具体流程。

(1) 定位攻击源 IP

在网络中，对于每个主机而言均有特定的行为模式，深入分析其行为模式，通过机器学习的方法构建行为模型，判定异常主机是定位攻击源的核心方法。本节采用第三章中提到的基于决策树的 AdaBoost 分类算法来构建分类器，算法原理和流程不再赘述，重点阐述对主机行为进行特征分析和提取的过程。

在流量监控模块的基础上，从流表信息中提取基于 IP 维度的流量统计信息成为可能。为了使得特征尽可能细粒度地表示出主机流量的变化情况，本文从两个角度来提取特征：主机作为报文发送方以及主机作为报文接收方，也就是说，提取某个 IP 地址的特征时，同时考虑所有以该 IP 为源地址的数据包的统计信息以及以该 IP 为目的地址的数据包的统计信息。

因此，本文提出基于源地址的 4 元组以及基于目的地址的 4 元组，合称为溯源 8 元组来表示某一个主机行为特征，将其作为判定是否异常的依据，8 元组即 8 个特征，其计算方法见表 4.3 和表 4.4。

a) 该 IP 作为源地址时，基于源地址的 4 元组如表 4.3 所示：

表 4.3 基于源地址的 4 元组

特征量	计算方法
总流表数	$flowInCount = n$
接收数据包总数	$pktInCount = \sum_{i=1}^n pktsIn_i$
接收字节总数	$byteInCount = \sum_{i=1}^n bytesIn_i$
目的 IP 地址的熵	$H(dstIP) = \sum_{i=1}^n p_i \log p_i$

表 4.3 中， n 表示源地址为该 IP 的流表项总数， p_i 表示这 n 个流表项中第 i 个目的 IP 地址出现的概率， $pkgsIn$ 和 $bytesIn$ 分别表示该 IP 接收到的数据包数和字节数。

b) 该 IP 作为目的地址时，基于目的地址的 4 元组如表 4.4 所示：

表 4.4 基于目的地址的 4 元组

特征量	计算方法
总流表数	$flowOutCount = m$
发送数据包总数	$pktOutCount = \sum_{i=1}^m pktsOut_i$
发送字节总数	$byteOutCount = \sum_{i=1}^m bytesOut_i$
源 IP 地址的熵	$H(srcIP) = \sum_{i=1}^m p_i \log p_i$

表 4.4 中, m 表示目的地址为该 IP 的流表项总数, p_i 表示这 m 个流表项中第 i 个源 IP 地址出现的概率, $pkgsOut$ 和 $bytesOut$ 分别表示该 IP 发出的数据包数和字节数。

结合表 4.3 和表 4.4, 攻击溯源模块提取的 8 元组特征向量如下:

$$T = \begin{bmatrix} flowInCount \\ pkgInCount \\ byteInCount \\ H(dstIP) \\ flowOutCount \\ pkgOutCount \\ byteOutCount \\ H(srcIP) \end{bmatrix}. \quad (4.3)$$

完成特征向量的构造后, 使用 AdaBoost 分类算法来训练主机的行为模型, 并基于模型识别异常主机。训练和识别的流程如下:

- a) 训练阶段: 使用训练集训练出分类器, 即正常主机的行为模式和异常行为模式。在网络中模拟 DDoS 攻击, 将正常主机的流量信息以及攻击主机的流量信息作为训练样本, 提取 8 元组特征向量并打上对应的标签, 送入基于决策树的 AdaBoost 分类算法, 构建主机的行为模型, 训练分类器;
- b) 识别阶段: 使用训练好的分类器进行判决, 识别主机状态的是否异常。在真实网络环境中, 一旦攻击检测系统发起攻击警报, 立刻根据当前周期采集的数据提取基于源地址和目的地址的溯源 8 元组并送入分类器, 分类器进行判决并找出异常主机。

(2) 定位攻击源所在的交换机

发现攻击者 IP 地址的下一步是找到与攻击者相连的交换机。基于 4.3.1 小节中的拓扑发现模块, 完成交换机发现、链路发现和主机发现这一系列步骤后, 可以实时获取网络拓扑图。构造数据结构 $ovsToIP$, 表示交换机端口与其相连主机列表的映射, 定义为

$$ovsToIP = \{ (ovs, port) : [hostIP] \},$$

其中, ovs 指交换机的 ID, $port$ 指交换机的端口, $hostIP$ 则为连接到该交换机的某个端口的不同主机的 IP 地址。通过遍历查询的方式, 由主机 IP 地址获取到所连接的交换机 ID; 在攻击溯源这个应用场景中, 我们可以由攻击者 IP 地址查询到其所连接的交换机 ID, 最终将攻击源信息攻击者 IP 和交换机 ID 传递至下一个模块。

4.4.2 基于动态流表的报文实时过滤

通过攻击溯源, 找到攻击者 IP 以及与其相连的交换机后, 利用 SDN 集中控制、动态可编程的特性, 对攻击报文进行实时过滤, 以实现阻止攻击的目的。具体来说, 控制器向指定的交换机下发 “Controller-to-Switch” 消息, 核心内容是针对攻击者的动态流表项, 命令其对来自攻击者 IP 的数据报文执行丢弃操作, 达到报文过滤的效果。图 4.7 描述了报文过滤的流程, 由控制器与交换机合作完成, 控制器基于攻击者 IP 构造针对攻击源的流表项并下发流表项, 交换机收到流表后执行过滤报文的操作。

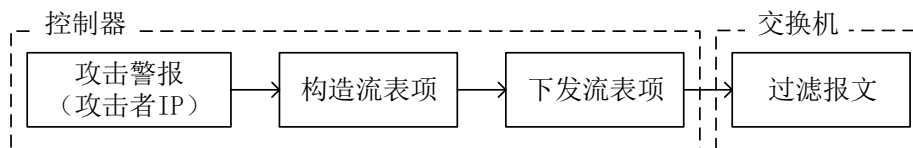


图 4.7 报文过滤流程

针对攻击者的动态流表项是报文过滤策略的关键。第二章中指出，每个流表项包括匹配域、优先级、指令、计数器、有效时间、cookie 等几个主要部分，图 4.8 表示了针对攻击者的流表项的简化形式：为了执行对指定 IP 地址的报文过滤，在匹配域中，将源 IP 地址设置为攻击源 IP，其余则为通配项；指令设置为“DROP”，表示对匹配的报文执行丢弃操作；同时，为了避免匹配域的雷同，即一条报文同时与多条流表项匹配成功，将优先级这一字段设置为 10，表示高优先级，而普通流表项（指令为转发的流表项）的优先级默认为 1。

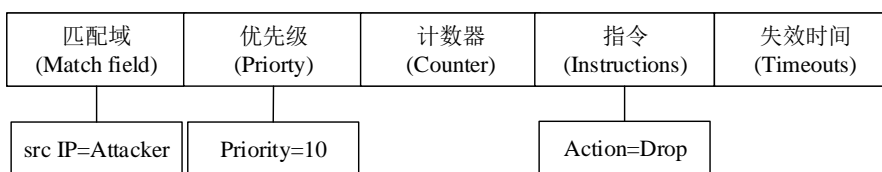


图 4.8 针对攻击者的动态流表项

上述流表项被包装为“Controller-to-Switch”类型的消息下发至交换机。“Controller-to-Switch”类型的消息由控制器主动发起，无需交换机提前发送请求同时也不强求交换机做出回应。注意，此处控制器仅向攻击源所处的交换机发送消息，既能够保障在攻击源头丢弃所有来自攻击主机的数据包，又减小了交换机中流表项的冗余。若控制器向攻击源报文所触达的所有交换机均发送消息，则很多交换机中将出现无效流表项，造成交换机资源的浪费。

攻击源所连接的交换机收到控制器的消息后，解析并安装新的流表项。当该交换机收到来自攻击主机的报文时，将优先匹配新安装的动态流表项，根据指令丢弃该报文。

为了对上述方案进一步完善，减少资源浪费、提升系统效率以及可用性，本文在报文过滤方案中提出了如下两个机制：基于黑名单的流表保护机制和全局动态白名单机制。

(1) 基于黑名单的流表保护机制

文献^[12, 25]以及本文实验（见第五章）均证明，报文过滤的方案能够有效完成对攻击源的攻击阻断，但攻击结束后，大量针对攻击源的动态流表项被遗留在交换机中，成为占用交换机资源的无效流表。由于交换机内流表数量有限，上述无效流表不但浪费了交换机的内存资源，而且容易导致交换机的流表存储空间被占满甚至溢出。

针对上述问题，基于 OpenFlow 协议中流表项的删除规则，本文提出流表保护机制：对于上文提到的针对攻击者的动态流表项，将其闲置超时时间即 `idle_timeout` 这一字段设置为 5 秒，表示若 5 秒内没有匹配该流表项的数据包达到，交换机则删除该流表项。这一措施将及时清除交换机中大量失效的针对攻击者的流表项，减小交换机中流表溢出的可能。

仅将动态流表项的闲置超时时间减小将带来如下问题：当攻击在一定时间后再次发起，曾被识别出的攻击主机再次发出恶意报文，但此时针对攻击主机的流表项因为超出了超时时间而被销毁，交换机则将重新请求控制器以获取处理该数据包的指令，控制器需要等待攻击检测系统再次识别出

攻击后才制定相应的阻断措施，上述过程将造成攻击阻断的延迟以及系统资源的浪费。因此，本文在流表保护机制中加入黑名单模块，即在控制器中建立一个黑名单数据库，用于保存被识别为攻击者的主机的 IP 地址。当攻击再次发生时，控制器无需通过攻击检测模块进行检测，直接查询黑名单即可识别攻击主机，同时立即制定过滤报文的流表来阻断攻击。黑名单的更新规则如下：每当攻击检测系统发现攻击后，溯源模块找到攻击者 IP 地址，并将该 IP 插入黑名单数据库。结合黑名单后，当交换机向控制器发起路由请求后，控制器首先查询黑名单数据库，一旦发现新数据包的源 IP 地址在黑名单数据库中，将直接下发动作为丢弃的流表项，指示交换机丢弃来自这个 IP 地址的所有报文。

综上所述，基于黑名单的流表保护机制其核心包括两大功能：闲置超时时间的设置以及黑名单数据库的建立和更新。图 4.9 描述了结合流表保护机制的报文过滤方案。

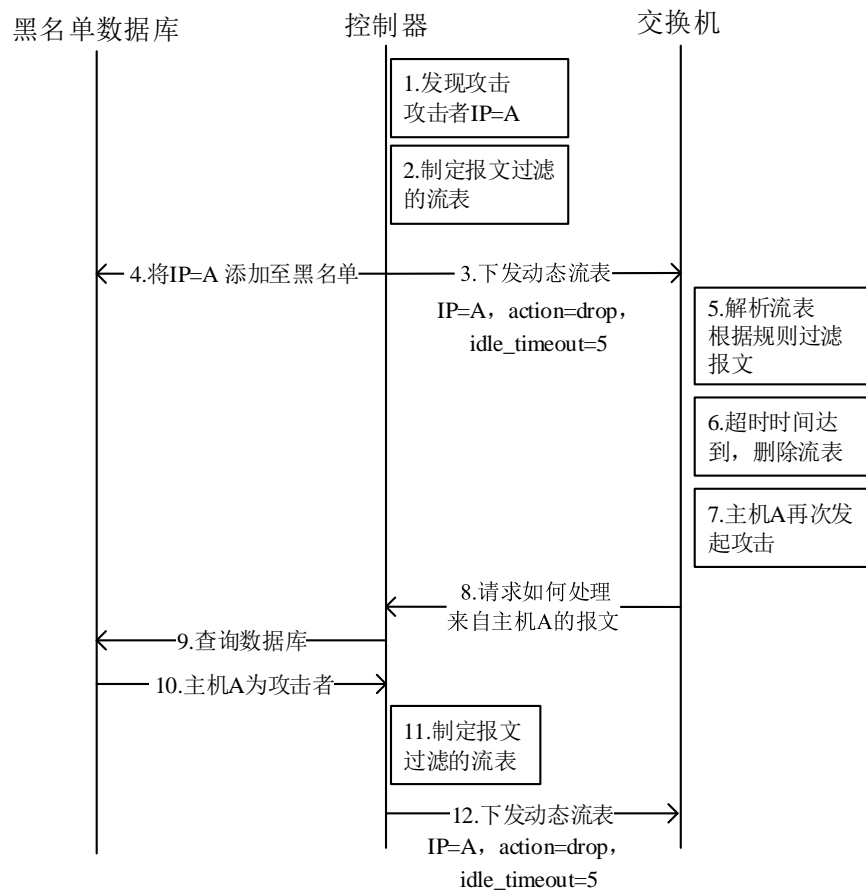


图 4.9 结合流表保护机制的报文过滤方案

基于图 4.9，结合了流表保护机制的报文过滤方案的流程如下：

- a) 攻击检测系统发现攻击，攻击溯源模块定位到攻击源；
- b) 控制器生成流表项（IP=A，action=drop，idle_timeout=5）并下发至对应的交换机；同时，更新黑名单数据库，增加 A 的 IP 地址这一项；
- c) 交换机解析流表，根据规则对来自主机 A 的所有报文执行丢弃操作；该流表项在交换机中的存留时间为 5 秒，超时后，交换机将自动删除该流表项，以防止流表溢出；
- d) 超时后 DDoS 攻击再次发生，主机 A 再次发出数据包，交换机则无法找到与其匹配的流表项，将向控制器发送请求以获取处理该数据包的新的流表项；
- e) 控制器通过查询发现主机 A 存在于黑名单数据库中，直接下发流表项（IP=A，action=drop，

idle_timeout=5), 命令交换机丢弃来自该 IP 的报文。

(2) 全局动态白名单机制

真实的网络环境往往并非完全符合攻击检测系统所建立的数学模型, 一方面, 因为业务需求的突发性可能导致网络流量异常, 被攻击检测系统判定为攻击并发起警报, 部分合法主机的行为被攻击溯源中的算法判定为异常主机; 另一方面, 网络中频繁出现需要自由进行网络管理和测试的需求, 例如, 蜜罐系统、压力测试系统等等。上述两类情况下, 合法主机的非攻击行为极有可能被判定为攻击而导致其报文被过滤。

针对上述问题, 本文提出全局动态白名单机制以避免自合法主机的有效报文被过滤, 同时为网络提供一定的自由度以供测试。由网络管理员在报文过滤模块中创建白名单数据库, 添加白名单主机 IP, 白名单中的主机将视为安全主机, 不被执行攻击阻断措施。全局动态白名单机制保障了网络的可行性, 提升网络的自由度, 赋予网络管理者更大的弹性。

白名单机制具有如下特点:

- a) 全局性: 整个网络建立全局唯一的白名单数据库
- b) 动态性: 数据库中的每一条白名单均可设置过期时间, 即没有永远安全的主机, 防攻击者利用漏洞攻破白名单中的主机对网络进行攻击

图 4.10 所示为结合了全局动态白名单机制的攻击阻断流程。

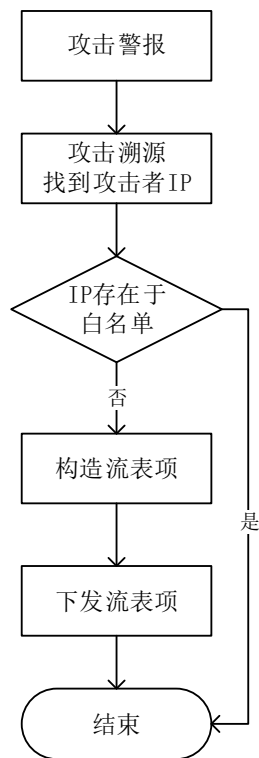


图 4.10 结合全局动态白名单机制的攻击阻断流程

基于图 4.10, 结合了全局白名单机制的攻击阻断流程如下:

- a) 攻击检测系统探测到异常, 发起攻击警报;
- b) 攻击溯源模块提取基于 IP 的特征 8 元组并送入分类器, 定位到攻击源 IP, 然后根据网络拓扑图, 找到攻击者所在的交换机;

- c) 报文过滤模块检测攻击源 IP 是否处于白名单中，若存在，则忽略此攻击源，结束流程；若不存在，则进入第四步；
- d) 流表构造模块构造针对攻击者的动态流表项；
- e) 消息发送模块向特定的交换机下发含有流表项的“C-S”消息。

4.5 本章小结

本章基于 SDN 网络，设计了一个完整的 DDoS 攻击防御方案，包括基于带宽的动态最优路由模块以及攻击阻断模块。

动态最优路由模块是从系统重构的角度出发，以适应网络流量、拓扑结构的实时变化，在网络面临攻击时，提升网络性能以及抗压能力，最大化利用网络资源来分散和吸收攻击流量。该方案主要包括三个模块：拓扑发现模块，流量监控模块和基于带宽的最优路由模块。核心是基于带宽的最优路由算法：在已知网络流量和带宽信息的基础上，根据改进的 K 最短路径算法，获取路由决策并告知交换机转发规则。

攻击阻断模块目的是发现攻击后在最短时间内阻止攻击，包括基于主机行为的攻击溯源阶段和基于动态流表的报文实时过滤阶段。在攻击溯源阶段，提出基于源地址和目的地址的特征 8 元组来表示主机的行为特征，并使用 AdaBoost 分类算法，该算法基于 8 元组来训练分类器以识别异常主机，定位攻击源。报文过滤阶段的核心是通过控制器向指定交换机下发针对攻击源的动态流表，命令交换机丢弃数据包以实现攻击阻断；同时，基于黑名单的流表保护模块有效地减小了交换机被无效流表占满导致溢出的可能性；全局动态白名单模块则提升了系统的可行性以及网络的自由度。

第五章 基于 SDN 的 DDoS 防护系统实现与测试

5.1 引言

为了对第三章和第四章提出的 DDoS 检测方案以及防御方案进行测试和评估,本章搭建了 SDN 网络的仿真环境并在此基础上实现了完整的 DDoS 防护系统,进一步地,通过在仿真环境中模拟 DDoS 攻击,对防护系统进行测试和评估。

目前对软件定义网络中 DDoS 攻击的研究大部分着重于检测部分,且缺乏在 SDN 环境中的验证。文献^[22-24, 30]提出基于 SDN 的 DDoS 检测算法后并没有在 SDN 网络中进行验证,仅通过提取离线数据集进行仿真,由于数据集时间久远因而存在大量无效数据,同时由于数据集并不是基于 SDN 网络产生的,因而适用性不强。另外,虽然文献^[10, 21, 26-29, 31, 33, 35]搭建了 SDN 的环境并进行了仿真,但大多数基于简单的树形或星形拓扑的网络,然而 SDN 最常见的应用场景是数据中心网络,基于树形或星形的网络拓扑与数据中心网络拓扑相差甚远。针对上述不足,本文利用数据中心里广泛使用的 Fat-tree 结构,搭建网络以进行仿真。

本章的内容安排如下:首先概述了搭建 SDN 网络的环境,包括仿真平台 Mininet、控制器 Ryu 以及交换机 Open vSwitch;然后介绍基于 SDN 的 DDoS 防护系统的整体架构、模块设计以及模块间的通信方式;接着描述了仿真实验的设计,包括网络拓扑的选择、数据中心流量的模拟、服务器的搭建以及 DDoS 攻击的模拟;最后在网络中进行不同的实验并分析防护系统各个模块的结果,对系统的可靠性、复杂度、可用性和可拓展性等性能进行评估。

5.2 系统设计和实现

本节主要介绍基于 SDN 的 DDoS 防护系统的网络平台、系统整体架构、系统中每个模块的具体设计以及模块间的通信方式。

5.2.1 SDN 网络搭建

(1) 仿真平台 Mininet

本文选择 Mininet 作为搭建 SDN 网络的平台。Mininet 是一个轻量级网络仿真平台,也是目前研究 SDN 网络的首选仿真平台。Mininet 基于 Linux Container 架构开发,采用虚拟化技术使一个单一的系统看起来像一个完整的网络,运行相同的内核、系统和用户代码。作为进程虚拟化网络仿真工具,它可以创建含有主机、交换机、控制器和链路的虚拟网络。Mininet 中的主机、交换机、链路和控制器是真正的实体,只是使用软件而非硬件来创建的,而且大多数情况下它们的行为特征与硬件元素相似;在 Mininet 上进行的实验,可以无缝的迁移到真实环境中。此外,Mininet 提供丰富的可扩展 python API,本文基于这些 API 拓展了新的功能,以实现仿真实验。

(2) 控制器 Ryu

控制器是 SDN 网络的重要组成部分,是 SDN 网络的大脑。自 SDN 发展以来,许多组织推出了不同特色的控制器: NOX、POX、Ryu、Floodlight、OpenDayLight 和 ONOS 等等。其中, Ryu 因为

其小巧精干、开发效率高、社区活跃度高等原因，更适合科研应用，因此本文选择 Ryu 作为软件定义网络的控制器，在 Ryu 的基础上实现防护系统。表 5.1 列出了 Ryu 的重要组件及其相应的功能。

表 5.1 Ryu 重要组件与功能

组件名	功能说明
base.app_manager	Ryu 组件调度中心
cmd.*	Ryu 的命令解析相关模块
controller.controller	描述控制器的行为，如连接管理
controller.ofp_event	完成 OpenFlow 报文到事件的转换
controller.ofp_handler	处理 OpenFlow 报文的基础模块
lib.*	定义包括 TCP/IP 协议等相关报文
ofproto.ofproto_vx_x	定义 OpenFlow 协议不同版本格式
ofproto.ofproto_vx_x_praser	定义 OpenFlow 报文的类及解析
topology.*	定义拓扑相关的事件，提供 API
app.rest	定义基础的 REST API 接口

(3) 交换机 Open vSwitch

Open vSwitch 是在虚拟化平台上运行的虚拟交换机，以下简称为 OVS。OVS 为虚拟网络提供二层交换功能，支持访问控制、网络隔离、流量监控等服务，原生支持 OpenFlow 协议。当 OVS 不与 SDN 的控制器相连时，可以使用 `ovs-octl` 命令来操作 OVS；本文中，OVS 连接与控制器 Ryu 相连，通过 Ryu 对 OVS 进行远程管理控制。

OVS 内部实现近似一个基于 OpenFlow 的 SDN 结构，图 5.1 表示其结构。User Space 可以看作管理层（Control Plane），Kernel 层看作数据层（Data Plane）。OVS 主要包括三个模块：

- `ovs-vswitchd`：守护进程，实现基于流的交换功能
- `ovsdb-server`：轻量级的数据库服务，主要保存整个 OVS 的配置信息，`ovs-vswitchd` 根据数据库中的配置信息工作
- `kernel Datapath`：Datapath 是流的一个缓存，缓存流的匹配结果，避免下一次流到达时继续到用户空间进行流的匹配操作

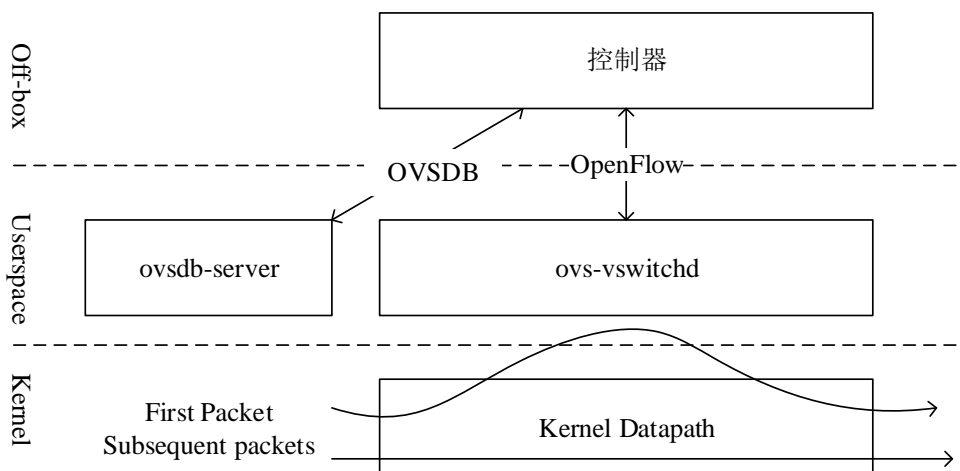


图 5.1 OVS 内部结构

5.2.2 系统架构

本文在 Mininet 平台上实现 DDoS 防护系统，部署了防护系统的 SDN 网络整体架构如图 5.2 所示，包括数据层、控制层和应用层。数据层内，交换机一方面负责网络数据包的转发，另一方面通过南向接口与控制层进行通信，进行信息交互以及流表安装；控制层使用 Ryu 控制器对整个网络进行集中控制，通过南向接口与数据层进行信息传递以及北向接口与应用层进行交互；应用层内则部署了本文设计的 DDoS 防护系统，系统由网络信息处理、触发检测、深度检测、攻击溯源、报文过滤和动态路由等六大模块构成。图 5.2 中，实线箭头表示数据的传递，虚线箭头表示模块之间的触发关系以及数据的传递，加粗部分为防护系统的六大模块。

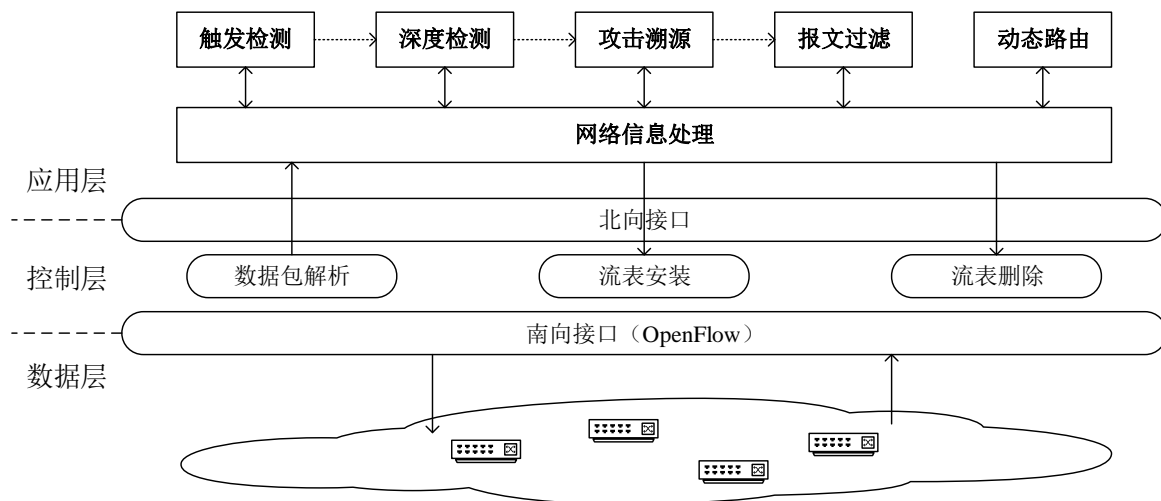


图 5.2 部署了 DDoS 防护系统的 SDN 网络整体架构

5.2.3 模块设计

如图 5.2 所示，基于 SDN 的 DDoS 防护系统由网络信息处理、触发检测、深度检测、攻击溯源、报文过滤和动态路由等六大模块构成。

网络中以下三个模块保持长期运行：网络信息处理模块、触发检测模块和动态路由模块。网络信息处理模块负责与控制层进行直接通信，提供流表信息采集、网络拓扑构造、动态流表构造等底层服务；触发检测模块负责基于网络信息进行多维特征时间序列的构造并使用 SMNA-CUSUM 算法对序列进行统计特性跟踪观测，实现异常流量的粗粒度检测；动态路由模块根据网络信息和拓扑图，采用改进的 K 最短路径算法，实时计算基于带宽的最优路由。

深度检测、攻击溯源、报文过滤等三个模块在触发检测模块发起攻击预警后依次触发。深度检测模块根据网络信息进行基于时间窗的特征构造，使用 AdaBoost 分类算法基于训练好模型来判别攻击是否存在；攻击溯源模块根据网络信息构造基于主机行为的特征向量，使用分类算法定位到异常主机，并基于网络拓扑信息定位主机所在的交换机；报文过滤模块接收溯源模块传递的信息：攻击者 IP 和交换机 ID，查询白名单数据库，若攻击者 IP 不存在于白名单中，则构造针对攻击源的动态流表以过滤报文。

图 5.3 描述了防护系统的流程图，其中省略了网络信息处理部分，从触发检测模块开始，具体流程如下：

- (1) 触发检测模块对全网流量特征的时间序列进行统计特性跟踪, 实现异常流量的粗粒度检测, 若发现异常, 则进入步骤二, 否则基于检测周期 T_{poll} 循环该步骤;
- (2) 深度检测模块启动, 对网络流量信息进行细粒度的特征提取并基于时间窗构造特征向量, 利用基于决策树的 AdaBoost 算法进行攻击检测, 若未检测到攻击, 则回到步骤一; 若检测到攻击, 则进入步骤三;
- (3) 攻击溯源模块启动, 根据网络信息构造基于主机行为的特征向量, 使用分类算法定位到异常主机 IP 地址, 并基于网络拓扑信息定位主机所在的交换机;
- (4) 报文过滤模块启动, 根据攻击源所在的主机 IP 地址来查询白名单数据库, 若该地址属于白名单, 则忽略该地址, 否则, 构造过滤指定报文的动态流表并向交换机下发消息, 收到消息的交换机将基于流表的规则对指定 IP 的数据包进行丢弃操作, 完成攻击阻断。

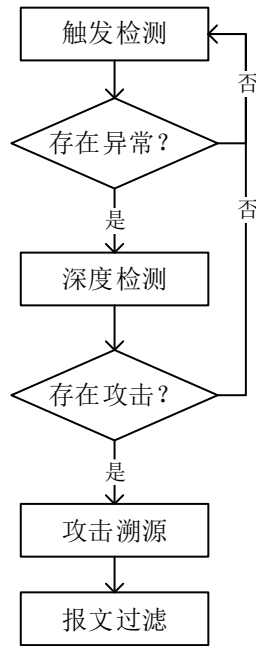


图 5.3 防护系统流程图

5.2.4 模块间通信

本文所设计的防护系统涉及功能众多, 需要在 Ryu 上开发多个模块来协调工作, 共同完成攻击检测以及攻击防御, 因此, 模块间通信成为关键问题, 下面将重点阐述防护系统中不同模块间的通信的方式。

(1) _CONTEXTS

_CONTEXTS 是 RyuApp 类中的一个属性, _CONTEXTS 中的内容将作为当前模块的服务在模块初始化时得到加载。同时, 该模块将具备对这个服务有完全的读写能力。由于 Ryu 不支持多级的服务关系, 若 A 是 B 的服务, 那么 B 就不能作为其他模块的服务, 即这种服务关系只有两层。如图 5.2 所示, 对于防护系统中的动态路由与网络信息处理这两个模块, 其中拓扑发现 (TopologyAwareness)、流表监控 (FlowMonitor) 和 packet-in 消息处理 (PacketInCount) 这三个类, 为动态路由模块中动态路由 (DynamicRouting) 类提供服务, 使用 _CONTEXTS 的方式进行通信, 核心代码如下。


```
class DynamicRouting(app_manager.RyuApp):
    _CONTEXTS = {
        "topology_awareness": topology_awareness.TopologyAwareness,
        "flow_monitor": flow_monitor.FlowMonitor,
        "packet_in_count": packet_in_count.PacketInCount
    }
```

(2) app_manager.lookup_service_brick()

在某些场景中，当一个模块需要使用其他模块的数据但不希望通过_CONTEXTS 的方式将对方作为自己的服务来加载时，可以通过 app_manager.lookup_service_brick() 这个函数来获取其他模块运行中的实例从而获取其数据，该方案与 import 不同，import 引入的是静态的数据，如某个类的函数的定义、静态数据的定义等，当涉及到动态的数据，import 则无法获取到对应的数据。网络信息处理模块中，流量监控这个类需要使用拓扑发现的数据，核心代码如下。

```
class FlowMonitor(app_manager.RyuApp):
    def __init__(self, *args, **kwargs):
        super(FlowMonitor, self).__init__(*args, **kwargs)
        self.topology = lookup_service_brick('topology')
```

(3) Event

Event 方法是 Ryu 中模块间通信最常见的形式，常用于订阅发布模式这类多模块协同工作的场景，实现模块之间解耦。每次交换机和控制器 Ryu 建立连接时，都会实例化一个 Datapath 对象来处理这个连接。在 Datapath 对象中，会将接收到的数据解析成对应的报文，进而转化成对应的事件，然后发布。注册了对应事件的模块将收到事件，然后调用对应的 handler 处理事件，例如 _packet_in_handler 函数，用于处理 packet-in 类型的消息。

```
def _packet_in_handler(self, ev):
    #Hanle the packet in packet, and register the access info.
    msg = ev.msg
    datapath = msg.datapath
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
```

5.3 实验环境搭建

本节首先介绍实验的网络拓扑：数据中心里常见的 Fat-tree 拓扑结构，接着阐述如何模拟真实的数据中心网络流量以及 DDoS 攻击流量，最后展示了 DDoS 攻击的过程以及攻击效果。

5.3.1 网络拓扑的搭建

本文选择 Fat-tree 结构（以下简称胖树结构）作为实验的网络拓扑结构。一方面，胖树结构被广泛地应用于数据中心网络，而数据中心是 SDN 网络的最常见应用场景。另一方面，数据中心是 DDoS

攻击的重点对象，尤其是多租户的数据中心，DDoS 攻击常由内部发起，普通的防护方法难以应对。

胖树结构^[74]在普通树形结构的基础上进行改进，由参数 k 决定具体的网络规模，图 5.4 表示 k 为 2 时的网络结构。整个网络自下而上分为三个层次：边缘层、汇聚层和核心层，边缘层交换机和汇聚层交换机组成一个集合，即 pod。胖树的构建规则是，每颗树包含 k 个 pod 和 $(k/2)^2$ 个核心层交换机，每个核心层交换机连接 k 个 pod，每个 pod 中均包含 $k/2$ 个边缘层交换机以及 $k/2$ 个汇聚层交换机，且每个 pod 连接 k 个客户端。

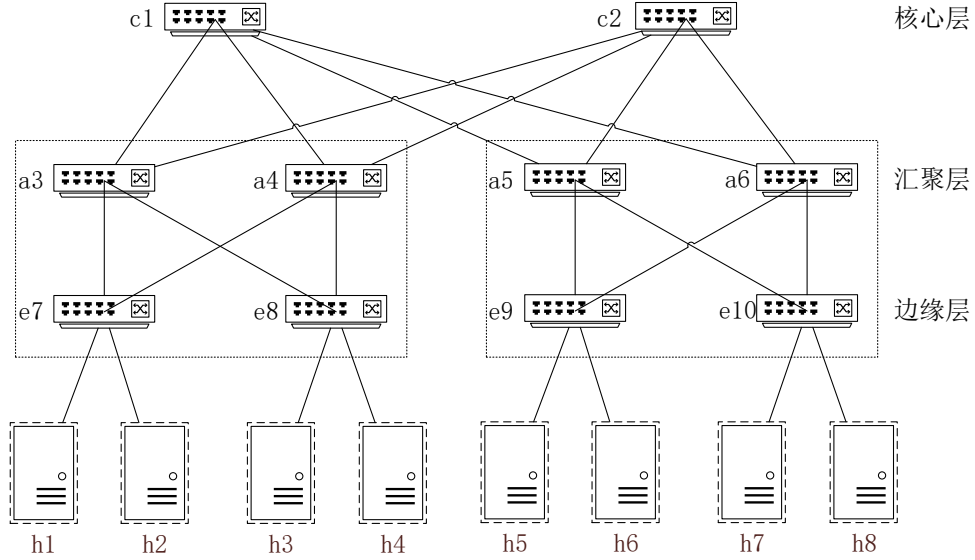


图 5.4 胖树网络拓扑 ($k=2$)

本文设计的实验中，胖树结构的参数 k 设为 2，网络结构与图 5.4 相同，其中包括 2 台核心层交换机、4 台汇聚层交换机以及 4 台边缘层交换机，分别记为 $c1 \sim c2$ 、 $a3 \sim a6$ 和 $e7 \sim e10$ ，边缘层交换机上共连接 8 台主机，记为 $h1 \sim h8$ ，其 IP 地址设置为 10.0.0.1~10.0.0.8。

5.3.2 真实网络流量的模拟

模拟真实的网络流量是测试防护系统的前提条件。在数据中心网络这个应用场景中，有两种常见的流量模型：随机模型和等概率模型。为了简化分析，本文使用随机模型，即主机向网络中的任意一台主机等概率地发送数据包。

为了方便且迅速地在 Mininet 平台中生成满足上述流量模型的流量，本文在分析了 Mininet 源代码的基础上对其进行功能拓展，将网络流量的生成过程封装到自定义命令中，并添加到 Mininet 的控制台命令集。

本文利用 Mininet 中自带的 iperf 工具进行功能拓展。Iperf 用于测试网络性能，基于客户端和服务端模式，通过发送一定速率的数据包测得两个主机间 TCP 或者 UDP 的性能，如带宽质量、延时抖动和数据包丢失率等。测量 UDP 的性能时利用了客户端向服务器发送指定速率及大小的数据包这一底层服务。本文在 iperf 这一底层服务的基础上，设计了批量处理的方案以实现所有主机之间的数据包的交互，实现上文提到的网络流量模型。实现过程分为如下四个步骤：

(1) 实现生成网络流量模型的功能

在 mininet/mininet 目录下，net.py 模块实现了 Mininet 类，即仿真平台 Mininet 的主体类，该类

实现了网络节点的添加配置、网络基本功能以及一些选项功能。本文在这个类中首先实现函数 `iperfTween`，完成一对主机之间的数据包发送以及数据记录；然后，添加函数 `iperfMulti` 完成生成网络流量的功能。核心代码如下。

```
class Mininet( object ):
    def iperfTween(self,hosts=None,udpBw='10M',period=60,port=5001):
        client,server=hosts
        filename=client.name[1:]+'.out'
        output( "%s and %s\n" % ( client.name, server.name ))
        iperfArgs ='iperf -u '
        bwArgs = '-b '+udpBw + ' '
        server.cmd(iperfArgs + '-s -i 1' + ' > /home/YHW/sdn/log/' + filename + '&')
        client.cmd(
            iperfArgs + '-t '+ str(period) + ' -c ' + server.IP() + ' ' + bwArgs
            + ' > /home/YHW/sdnlab/log/' + 'client' + filename + '&')
    def iperfMulti(self, bw, period=60):
        base_port=5001
        server_list = []
        host_list = []
        host_list = [h for h in self.hosts]
        _len = len(host_list)
        for i in xrange(0, _len):
            client = host_list[i]
            server = client
            while( server == client ):
                server = random.choice(host_list)
            server_list.append(server)
            self.iperfTween (hosts = [client, server], udpBw=bw, period= period,
                port=base_port)
            sleep(.05)
            base_port += 1
        sleep(period)
```

(2) 注册自定义命令

在 `mininet/mininet` 目录下，`cli.py` 模块定义了 `CLI` 类，为 `Mininet` 平台提供命令行接口，用于解析用户输入的命令。自定义命令需要在 `CLI` 类中通过注册函数来注册这条自定义命令，因此本文在该类中添加函数 `do_iperfmulti` 用于注册 `net` 模块中的函数 `iperfMulti`，核心代码如下。

```
class CLI( Cmd ):
    def do_iperfmulti(self,line):
```

```

args=line.split()
if len(args)==1:
    udpBw=args[0]
    self.mn.iperfMulti(udpBw)
elif len(args)==2:
    udpBw=args[0]
    period=args[1]
    err=False
    self.mn.iperfMulti(udpBw,float(period))

```

(3) 注册命令与对应执行函数的映射关系

在 mininet/bin 目录下, mn 模块中声明了注册命令与对应执行函数的映射关系, 添加本文的自定义命令声明, 核心代码如下。

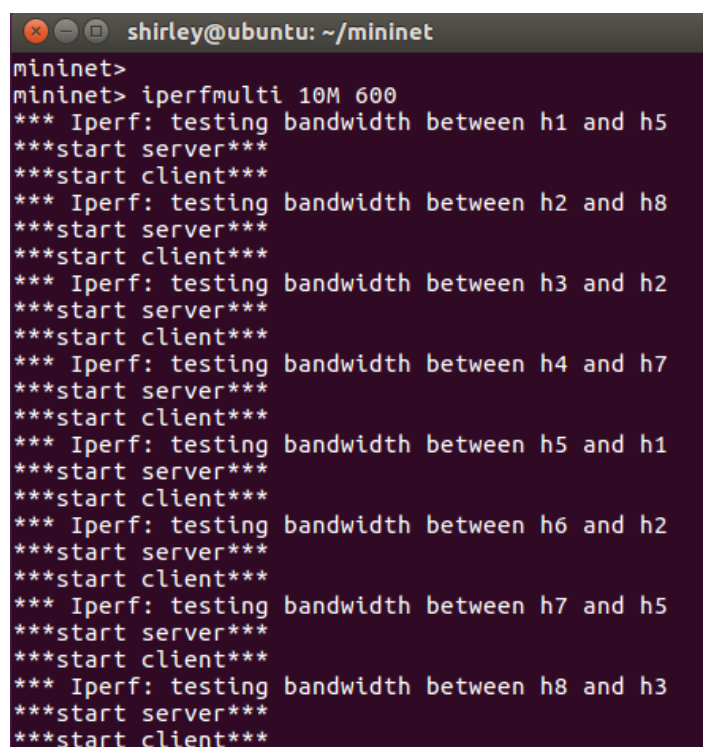
```

ALTSPELLING = { 'pingall': 'pingAll', 'pingpair': 'pingPair',
                 'iperfudp': 'iperfUdp', 'iperfmulti': 'iperfMulti' }

```

(4) 重新编译 Mininet

完成上述三个步骤后, 重新编译安装 Mininet, 新添加的自定义命令即可生效。注意此处只需要重新安装 Mininet 的核心文件, 在 mininet/util 目录下, 输入命令: ./install.sh -n, 完成 Mininet 的重新编译。



```

shirley@ubuntu: ~/mininet
mininet>
mininet> iperfmulti 10M 600
*** Iperf: testing bandwidth between h1 and h5
***start server***
***start client***
*** Iperf: testing bandwidth between h2 and h8
***start server***
***start client***
*** Iperf: testing bandwidth between h3 and h2
***start server***
***start client***
*** Iperf: testing bandwidth between h4 and h7
***start server***
***start client***
*** Iperf: testing bandwidth between h5 and h1
***start server***
***start client***
*** Iperf: testing bandwidth between h6 and h2
***start server***
***start client***
*** Iperf: testing bandwidth between h7 and h5
***start server***
***start client***
*** Iperf: testing bandwidth between h8 and h3
***start server***
***start client***

```

图 5.5 真实网络流量的模拟

以上四步实现了在 Mininet 中新增自定义命令以模拟真实网络流量的功能。通过在 Mininet 控制台中输入 iperfMulti 命令并指定带宽, 我们可以完成对该功能的测试。图 5.5 表示了模拟真实流量的测试结果, 指令表示持续 600s 发送带宽为 10M 的数据。

5.3.3 DDoS 攻击的模拟

完成对真实流量的模拟后,如何模拟 DDoS 攻击的成为最重要的问题。由本文绪论部分对 DDoS 威胁态势的调研可见,SYN Flood 和 UDP Flood 是当前 DDoS 攻击中流量最大的两种攻击类型。因此,本文在胖树结构的网络中模拟这两种 DDoS 攻击。

首先需要进行服务器的搭建。在图 5.4 所示的网络拓扑中,选择编号为 h7 的主机作为服务器。在 python 中,利用封装好的 ThreadingTCPServer 对象以及 ThreadingUDPServer 对象来创建支持并发操作的 TCP 服务器和 UDP 服务器,分别监听 20000 端口以及 30000 端口。

服务器搭建完成后,开始进行 DDoS 攻击的模拟。常见的攻击模拟工具包括 trafgen、Scapy、nping、hping3 以及 Stacheldraht 等等。本文选择 Scapy 工具进行攻击模拟,因为其功能强大、发包自由度高且便于直接在命令行进行交互测试。在 python 中引入 scapy 库,编写攻击脚本。UDP Flood 和 SYN Flood 攻击的核心代码如下。

```
while 1:
    f=open('./load','r')
    size=random.randint(1000,2000)
    data=f.read(size)
    sp=random.randint(1000,65535)
    t=random.randint(50,120)
    #UDP Flood
    packet_udp=(IP(dst=dst,ttl=t)/UDP(sport=sp,dport=dport_udp)/Raw(load=data))
    send(packet_udp)
    f.close()
    #TCP SYN Flood
    packet_tcp=IP(dst=args['dst'],ttl=t)/TCP(flags="S",sport=sp,dport= dport_tcp)
    send(packet_tcp)
    time.sleep(0.2)
```

最后,测试攻击脚本的效果。由于本文篇幅有限,下面仅描述 TCP SYN Flood 攻击的流程。假设 h1, h2 为攻击主机, h7 为 TCP 服务器, h4 为合法的 TCP 客户端。

- (1) 首先在 h1, h2 中添加 iptables 网络数据包过滤指令。因为默认情况下, Linux 主机在收到来自服务器的 SYN-ACK 类型的消息时, 将回复 RST 消息。为了达到攻击效果即, 需要阻断攻击主机发出的 RST 消息以造成服务器处于半连接的状态, 命令为:

```
sudo iptables -A OUTPUT -p tcp -s 10.0.0.1 -tcp-flags RST RST -j DROP
```

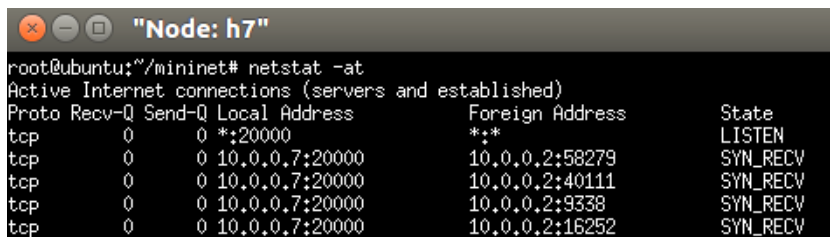
该命令表示, 丢弃该主机发出的所有源地址为 10.0.0.1 且标识为 TCP RST 的数据包;

- (2) 开启 h7 中的 TCP 服务, 即运行 TCP 服务器脚本;
- (3) 发起 SYN Flood 攻击, 即在攻击主机 h1, h2 中启动攻击脚本, 命令为:

```
sudo python synflood.py -d 10.0.0.7 -c 2000 -p 20000
```

该命令表示, 向 IP 地址为 10.0.0.7 的主机的第 20000 端口快速发送 2000 条请求以进行拒绝服务攻击;

- (4) 合法客户端 h4 向服务器 h7 发起 TCP 连接请求，即运行 TCP 客户端脚本；
- (5) 观测服务器 h7 的状态以及 h4 主机是否能正常接受服务，图 5.6 所示为主机 h7 的 tcp 连接，由图可见，存在大量与攻击主机的半连接，图 5.7 所示为合法主机发起请求后的状态，由图可见，发起请求失败，服务器拒绝服务，攻击成功。

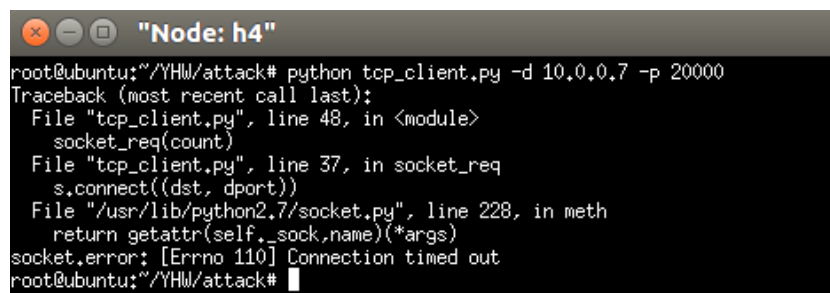


```

root@ubuntu:~/mininet# netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:*                0.0.0.0:*               LISTEN
tcp        0      0 10.0.0.7:20000          10.0.0.2:58279         SYN_RECV
tcp        0      0 10.0.0.7:20000          10.0.0.2:40111         SYN_RECV
tcp        0      0 10.0.0.7:20000          10.0.0.2:9338          SYN_RECV
tcp        0      0 10.0.0.7:20000          10.0.0.2:16252         SYN_RECV

```

图 5.6 服务器内的半连接



```

root@ubuntu:~/YHM/attack# python tcp_client.py -d 10.0.0.7 -p 20000
Traceback (most recent call last):
  File "tcp_client.py", line 48, in <module>
    socket_req(count)
  File "tcp_client.py", line 37, in socket_req
    s.connect((dst, dport))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 110] Connection timed out
root@ubuntu:~/YHM/attack#

```

图 5.7 合法主机发起请求连接失败

5.4 实验结果和分析

本节在搭建好的 SDN 环境中对 DDoS 攻击防护系统进行测试和评估，首先介绍评估本系统的标准，接着依据这些标准通过多次实验分别对检测系统和防御系统进行评估。

5.4.1 系统评估标准

本文从可靠性、复杂度、可用性和可拓展性等方面，对基于 SDN 的 DDoS 防护系统进行评估。

(1) 可靠性

DDoS 攻击防护系统的可靠性主要指攻击检测系统的检测率以及攻击防御系统的有效性。本文中对攻击的检测是一个二分类问题，常见二分类问题的评估指标主要有：准确率、精确率、召回率、F1 度量以及 ROC 曲线等等。防御系统的有效性则包括动态路由的成功率、丢弃数据包的成功率以及流表保护机制删除流表是否见效。

(2) 复杂度

DDoS 攻击防护系统的复杂度主要从时间和空间两个角度来考虑。时间上的复杂度由系统的响应时间来表示，主要包括攻击检测系统的延迟时间。空间上的复杂度由消耗 CPU 的多少来表示，由于防护系统在 Ryu 控制器中开发，因此，本文通过观测控制器中的 CPU 变化来度量空间复杂度。

(3) 可用性

可用性指的是系统在完成自身功能外，是否对其所处的整个网络环境产生不良影响甚至造成网络的不可用。对于本文所提出的攻击检测和防御系统，考虑以下几点：是否将合法流量误判为攻击流量，是否在丢弃报文时将合法请求丢弃，是否造成了控制器或交换机的资源大量占用等。

(4) 可拓展性

系统的可拓展性即架构和模块的拓展性,是否做到了松耦合。举例来说,对于攻击检测系统中的触发检测模块,若使用另一种触发检测算法替代本文提出的 SMNA-CUSUM 算法,是否方便易行。

5.4.2 检测方案结果和分析

首先评估系统的可靠性。本文在不同的网络环境下进行多次实验,以测试攻击检测系统在不同网络带宽以及不同攻击强度下的表现,具体在实验中体现为两个变量,网络的平均流速和攻击数据包的发包速率,表 5.2 所示为四类实验类型的参数。

表 5.2 四种实验类型

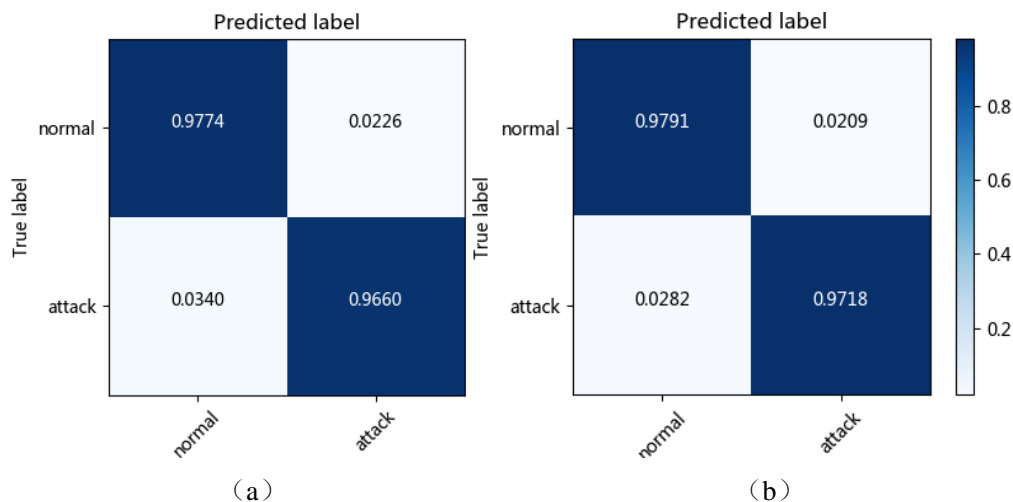
实验类型	平均流速 (Mbps)	攻击流速 (pkg/s)
1	1	20
2	1	200
3	10	200
4	10	2000

如图 5.4 所示的网络中, h7 为服务提供者, h1 和 h2 是向服务器发起 DDoS 攻击的两个僵尸主机, h4 为合法请求者。基于表 5.2 中的不同实验类型, 主机 h1-h8 模拟不同流速的合法网络通信行为, 此外, h1 和 h2 作为攻击源还将根据不同的发包速率向 h7 发送大量攻击数据包, 数据包内容和大小由程序随机生成。表 5.3 列出了检测结果的精确度, 由表格可见在本文模拟的网络环境中, 检测方案的精确度很高。

表 5.3 四种实验的检测精确度

实验类型	精确度(%)
1	97.31
2	97.69
3	97.20
4	98.06

图 5.8 表示四类实验的归一化混淆矩阵示意图, 每个混淆矩阵由大小为 2*2 的表格即 4 个指标构成, 按照顺时针顺序分别为: 真正例、假负例、真负例和假正例。混淆矩阵示意图表明, 检测系统中的分类算法在 4 个指标上均表现优异, 能够很好的区分出当前网络处于攻击状态还是正常状态。



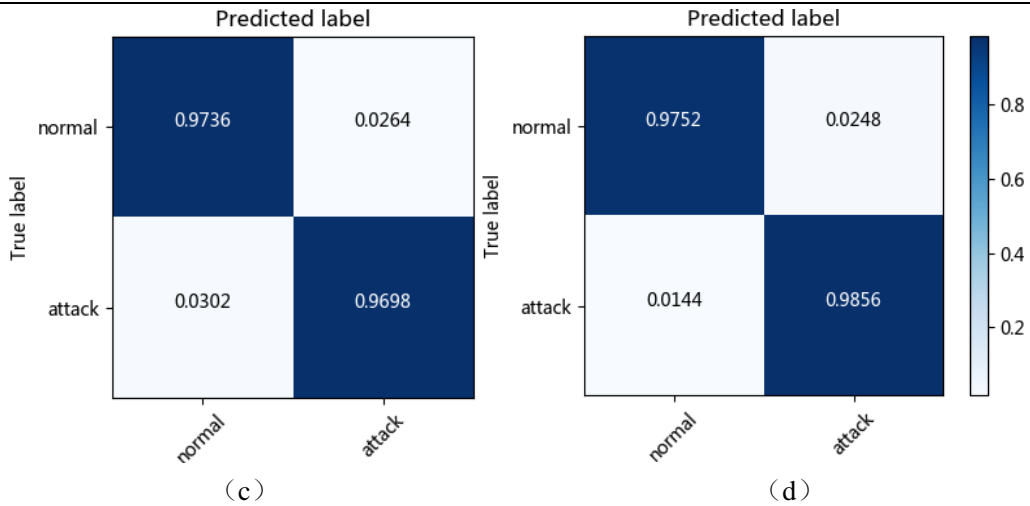


图 5.8 攻击检测算法的归一化混淆矩阵

图 5.9 是四类实验的 ROC (Receiver Operating Characteristic curve) 曲线图。图中，横坐标表示假正例率，纵坐标表示真正例率，AUC 表示曲线包围的面积，用来衡量 ROC 曲线的表现情况，面积越大表示分类器性能越好。由图可见，每类实验中，AUC 均接近于 1，表明了分类器的良好表现。

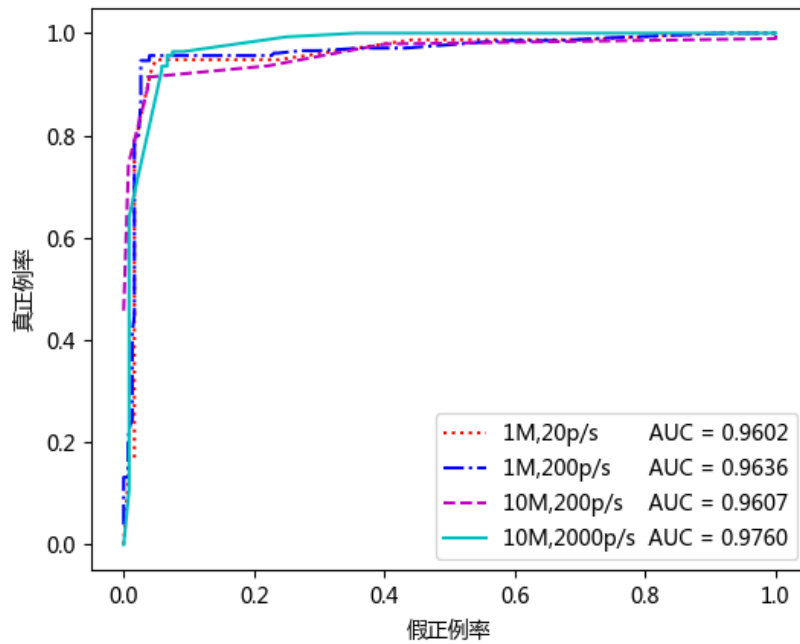


图 5.9 攻击检测算法的 ROC 曲线

表 5.4 展示了本文所提出的检测方案的平均检测准确率与文献^[13]、文献^[10]与文献^[29]的对比。

表 5.4 检测方案准确率对比

论文	精确率 (%)
Mousavi et al.(2015) ^[13]	95.0
Santos(2016) ^[29]	88.7
Conti et al.(2017) ^[10]	96.0
本文	97.7

上述实验结果表明，本文提出的由触发检测和深度检测结合的 DDoS 检测方案在不同类型的实

验中均取得了很好的性能，且与现有研究相比检测率较高，证明了系统较强的可靠性。

接下来对检测系统的复杂度进行评估。使用 CPU 占用率这一指标作为计算复杂度的核心指标。图 5.10 描述了深度检测系统中实时检测阶段的 AdaBoost 分类算法在与触发检测系统中的 SMNA-CUSUM 算法分别保持运行时，操作系统 CPU 占用率的变化。由图 5.10 可见，AdaBoost 算法在基于训练集进行检测判决时，消耗了相对较高的 CPU。经计算，SMNA-CUSUM 算法的 CPU 占用率平均比 AdaBoost 算法的判决部分降低约 4 个百分点，即 CPU 占用率减小了 20%，表明本文提出的由触发检测模块中低开销的算法替代机器学习分类算法，在网络中长期运行实现粗粒度检测的方案，相比直接使用机器学习算法进行检测，计算复杂度得到了明显的降低。

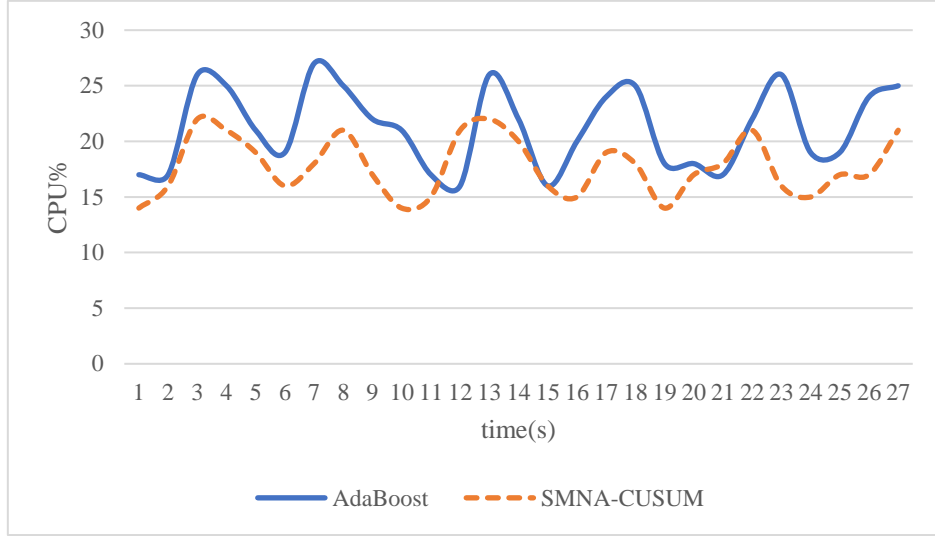


图 5.10 AdaBoost 算法与 SMNA-CUSUM 算法的 CPU 占用率

下面分析系统的延时，即从攻击源发起攻击到系统检测到攻击所需要的时间。检测系统的检测时长 t_{delay} 为触发检测模块的检测延迟时间 $t_{trigger}$ 和深度检测模块的检测延迟时间 t_{deep} 之和，如下所示：

$$t_{delay} = t_{trigger} + t_{deep} \quad (5.1)$$

触发检测模块的延时 $t_{trigger}$ 包括该模块中控制器的轮询周期即信息采集周期 T_{poll} ，以及 SMNA-CUSUM 算法的延时 $t_{SMNA-CUSUM}$ ，如下所示：

$$t_{trigger} = T_{poll} + t_{SMNA-CUSUM} \quad (5.2)$$

经实验表明，后者几乎可以忽略不计，因此 $t_{trigger}$ 近似为信息采集周期 T_{poll} : 5s。深度检测模块的检测延时主要取决于基于决策树的 AdaBoost 算法在进行检测判决时所消耗的时间。图 5.11 通过累计分布函数（CDF, Cumulative Distribution Function），显示了基于滑动时间窗的特征构造对判决时间的影响。累计分布函数是概率密度函数的积分，可以完整描述一个实随机变量的概率分布。由图 5.11 可见，通过滑动时间窗的方式增加特征值并未给系统延时造成明显的增加，80%以上的样本每次判决时间在 5ms 以内。整个检测系统的延时 t_{delay} 约为 5.005s。

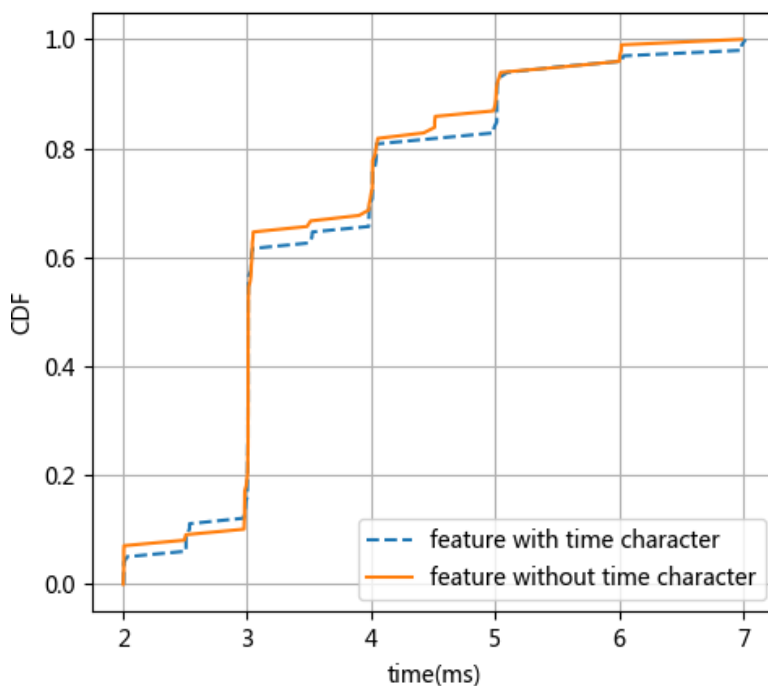


图 5.11 基于滑动时间窗的特征构造对判决时间的影响

最后，笔者经过对系统各模块的分析以及实验的验证得出结论，本文提出的检测系统具有较好的可拓展性。模块化的设计将不同的功能解耦合，每个模块间通过接口进行交互，这种“松耦合”的架构为系统内部不同功能的更新、测试和拓展提供了极大的便利。具体来说，若想要替换系统中的某个功能或者算法，只需修改对应的模块而无需影响其他模块。图 5.12 展示了将深度检测模块中的 AdaBoost 分类算法替换成其他 6 种算法后的结果，由该图可见，本文使用的基于决策树的 AdaBoost 分类算法效果最佳。

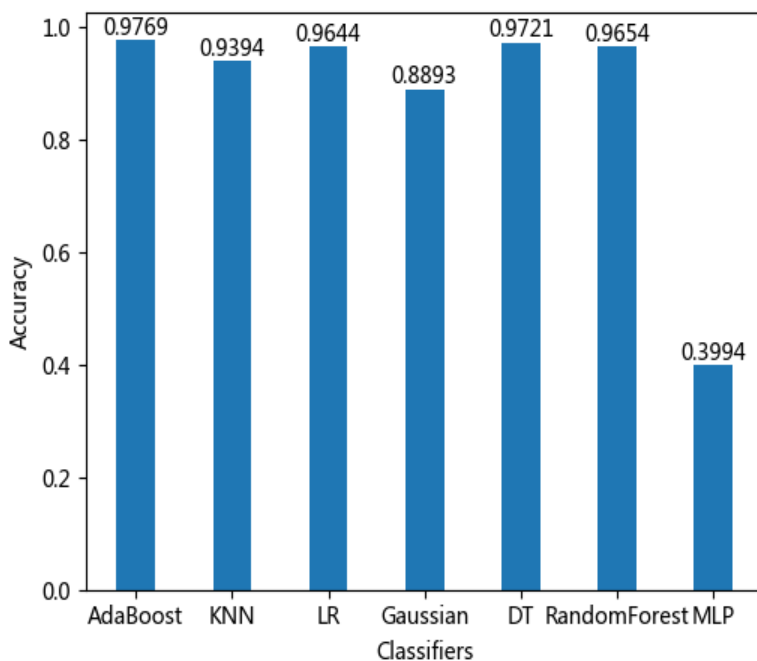
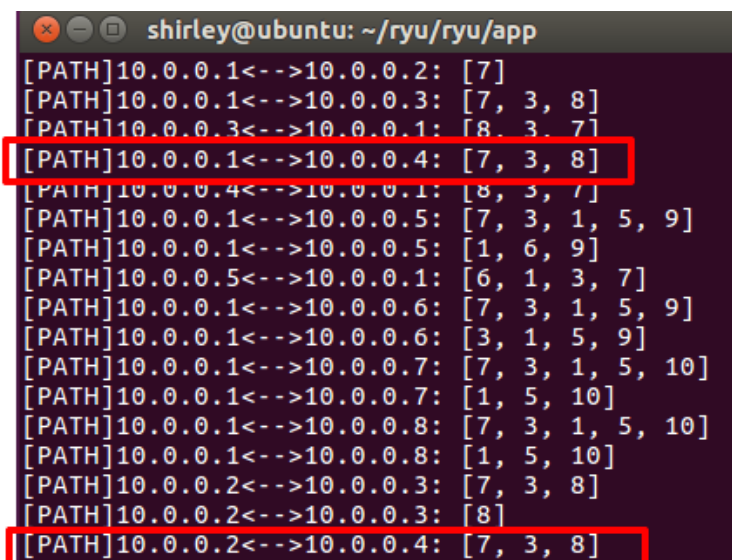


图 5.12 不同分类算法的精确率对比

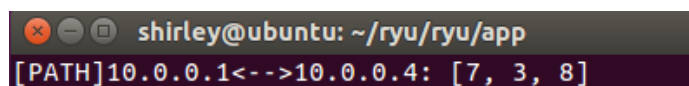
5.4.3 防御方案结果和分析

首先对基于带宽的最优动态路由方案进行测试。图 5.4 所示的 Fat-tree 拓扑中, h1 和 h2 到 h4 分别有两条路径: h1/h2--e7--a3--e8--h4 和 h1/h2--e7--a4--e8--h4, 记为[7,3,8]和[7,4,8], 其中数字表示所经过的交换机的 ID。网络启动时, 在 Mininet 中通过 pingall 命令获取每两个主机之间的路径, 图 5.13 展示了部分路径结果, 由图可见 h1 和 h2 到 h4 的默认路径均为[7,3,8]; 通过 iperf 工具依次进行 h1 向 h4 以及 h2 向 h4 发送大量数据包的操作, 重新查看 h1 和 h2 到 h4 的路径, 分别如图 5.14 和图 5.15 所示。两条路径分别为[7,3,8]和[7,4,8], 可见 h1 和 h2 各自独占了一条路径, 表明控制器基于带宽动态地更新了路由规则并成功地通知交换机执行了路由方案。



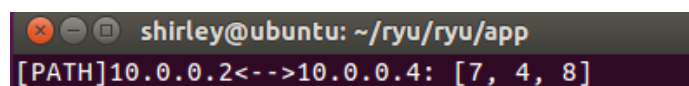
```
shirley@ubuntu: ~/ryu/ryu/app
[PATH]10.0.0.1<-->10.0.0.2: [7]
[PATH]10.0.0.1<-->10.0.0.3: [7, 3, 8]
[PATH]10.0.0.3<-->10.0.0.1: [8, 3, 7]
[PATH]10.0.0.1<-->10.0.0.4: [7, 3, 8]
[PATH]10.0.0.4<-->10.0.0.1: [8, 3, 7]
[PATH]10.0.0.1<-->10.0.0.5: [7, 3, 1, 5, 9]
[PATH]10.0.0.1<-->10.0.0.5: [1, 6, 9]
[PATH]10.0.0.5<-->10.0.0.1: [6, 1, 3, 7]
[PATH]10.0.0.1<-->10.0.0.6: [7, 3, 1, 5, 9]
[PATH]10.0.0.1<-->10.0.0.6: [3, 1, 5, 9]
[PATH]10.0.0.1<-->10.0.0.7: [7, 3, 1, 5, 10]
[PATH]10.0.0.1<-->10.0.0.7: [1, 5, 10]
[PATH]10.0.0.1<-->10.0.0.8: [7, 3, 1, 5, 10]
[PATH]10.0.0.1<-->10.0.0.8: [1, 5, 10]
[PATH]10.0.0.2<-->10.0.0.3: [7, 3, 8]
[PATH]10.0.0.2<-->10.0.0.3: [8]
[PATH]10.0.0.2<-->10.0.0.4: [7, 3, 8]
```

图 5.13 初始状态下的主机之间的路径



```
shirley@ubuntu: ~/ryu/ryu/app
[PATH]10.0.0.1<-->10.0.0.4: [7, 3, 8]
```

图 5.14 主机 h1 到 h4 的路径



```
shirley@ubuntu: ~/ryu/ryu/app
[PATH]10.0.0.2<-->10.0.0.4: [7, 4, 8]
```

图 5.15 主机 h2 到 h4 的路径

攻击阻断的实验在上一节中攻击检测实验的基础上完成。实验方案是, h7 为服务提供者, h1 和 h2 是向服务器发起 DDoS 攻击的两个僵尸主机, h4 为合法流量请求者。检测模块检测到攻击后发起攻击警报, 攻击溯源模块识别出攻击源 h1 和 h2 并通知报文过滤模块; 由于白名单数据库并未添加上述两个主机, 因此报文过滤模块基于流表保护机制直接构造了针对 h1 和 h2 的流表项并下发消息至交换机 e7, 实现攻击阻断; 此外, 攻击源 h1 和 h2 的 IP 地址被加入黑名单数据库。图 5.16 和图 5.17 表示交换机 e7 中流表项的变化: 检测到攻击前, 交换机中包含与 h1 和 h2 相关的正常流表项 (优先级为 1), 检测到攻击并定位至 h1 和 h2 后, 新增了两条 “action=drop” 的流表项并且其优先级为 10, 因此, 交换机 e7 将丢弃所有来自 h1 和 h2 的数据包。

```

shirley@ubuntu: ~/mininet
mininet> sh ovs-ofctl dump-flows e7
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=15.897s, table=0, n_packets=17, n_bytes=1020, idle_age=1, priority=655
35,dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=3.254s, table=0, n_packets=1, n_bytes=98, idle_age=2, priority=1,ip,in
_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:4
 cookie=0x0, duration=3.253s, table=0, n_packets=2, n_bytes=196, idle_age=2, priority=1,ip,i
n_port=4,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:3
 cookie=0x0, duration=3.189s, table=0, n_packets=2, n_bytes=196, idle_age=2, priority=1,ip,i
n_port=1,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:3
 cookie=0x0, duration=3.185s, table=0, n_packets=1, n_bytes=98, idle_age=2, priority=1,ip,in
_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:1

```

图 5.16 攻击发生前交换机 e7 中的流表项

```

shirley@ubuntu: ~/mininet
mininet> sh ovs-ofctl dump-flows e7
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=329.776s, table=0, n_packets=4, n_bytes=240, idle_age=327, priority=65
535,dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=123.802s, table=0, n_packets=2064, n_bytes=1215182, idle_timeout=15, i
dle_age=0, priority=10,ip,nw_src=10.0.0.1 actions=drop
 cookie=0x0, duration=116.141s, table=0, n_packets=1940, n_bytes=1147206, idle_timeout=15, i
dle_age=0, priority=10,ip,nw_src=10.0.0.2 actions=drop
 cookie=0x0, duration=313.902s, table=0, n_packets=3, n_bytes=294, idle_age=254, priority=1,
ip,in_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:4
 cookie=0x0, duration=313.900s, table=0, n_packets=4, n_bytes=392, idle_age=254, priority=1,
ip,in_port=4,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:3

```

图 5.17 攻击发生后交换机 e7 中的流表项

攻击阻断成功后，合法主机 h4 再次向服务器发起请求时，服务恢复正常，h4 收到来自服务器的回复，如图 5.18 所示，服务器的平均响应时间在 2 毫秒内。

```

"Node: h4"
root@ubuntu: ~/YHW/attack# python tcp_client.py -d 10.0.0.7 -p 20000
--1--time_gap is 0.985000 ms
connected!
--2--time_gap is 2.572000 ms
connected!
--3--time_gap is 1.043000 ms
connected!
--4--time_gap is 0.400000 ms
connected!
--5--time_gap is 0.990000 ms
connected!
--6--time_gap is 2.697000 ms
connected!

```

图 5.18 服务恢复后合法主机成功连接

由于本文设计了基于黑名单的流表保护机制，在攻击结束的 15 秒后，上述两个流表项不再有与之匹配的报文，因此被交换机自动删除。表 5.5 为整个网络中交换机内的流表数量变化。表 5.5 表明，攻击阻断后，系统成功删除了较多的无效流表，为交换机的内存节约了空间。

表 5.5 交换机内的流表项数量变化

最大流表数目	阻断后的流表数目	删除比例
256	198	22.7%

图 5.19 描述了网络遭受 DDoS 攻击前后，代表网络流量状态的几个典型特征的变化情况，包括数据包数 *pktCount*、字节数 *byteCount*、源地址的熵 $H(srcIP)$ 和目的地址的熵 $H(dstIP)$ ，图 5.19 中横坐标为时间，每个单位表示 5 秒的时间间隔，蓝色实线表示部署了 DDoS 防护系统的网络流量特征，灰色虚线表示未部署该系统的流量特征。显而易见，部署了防护系统的网络能够很好的实现

攻击检测和防御，在一定时间内将网络恢复至正常状态。

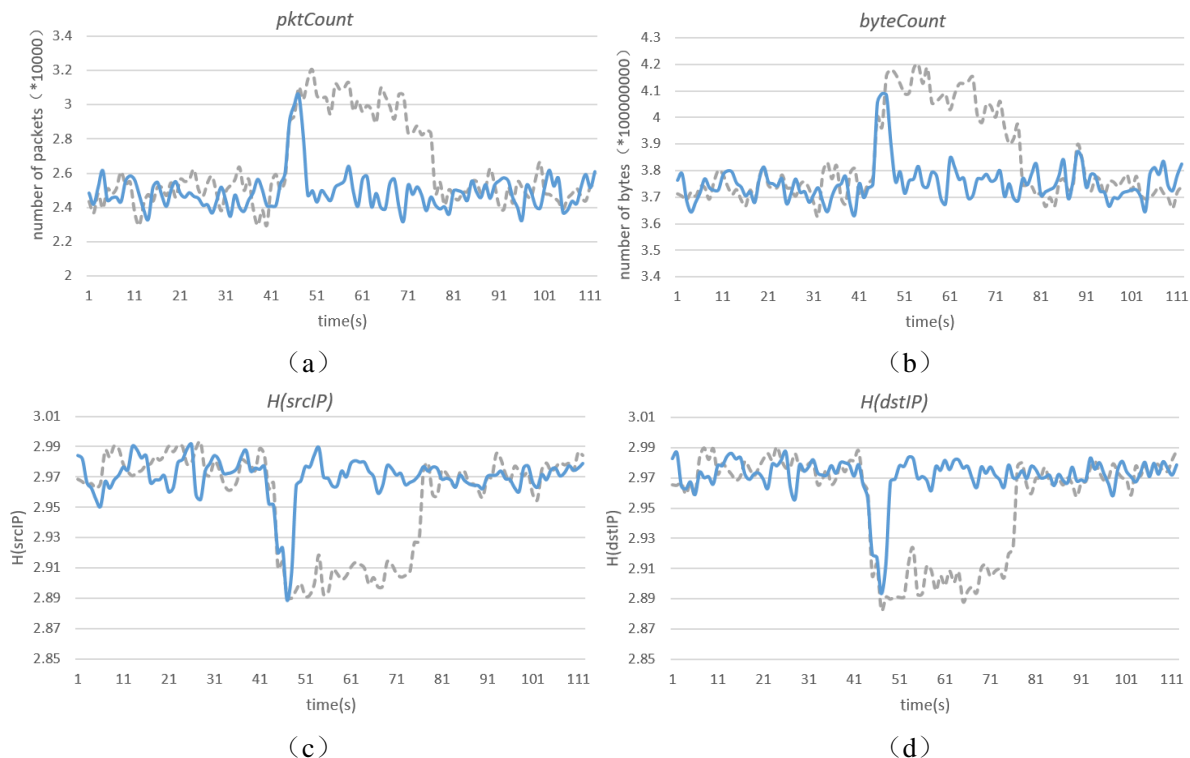


图 5.19 网络流量特征的变化

此外，设计如下场景对白名单机制进行验证：主机 h5 对服务器 h7 进行压力测试。在测试开始时首先将 h5 加入白名单数据库；通过主机 h5 向服务器发送大量数据包的方式来实施压力测试；由于 h5 的行为造成了网络流量状态的异常，攻击检测模块发出攻击警报；进一步地，溯源模块定位到主机 h5；报文过滤模块检测到主机 h5 存在于白名单中，攻击警报被解除，压力测试正常进行。由此可见，系统中的白名单机制增强了网络的可用性和灵活性。

5.5 本章小结

本章在 Mininet 平台上实现了完整的 DDoS 攻击防护系统，并在 SDN 环境中搭建 Fat-tree 拓扑结构的网络，拓展 Mininet 的功能，基于 Iperf 和 Scapy 进行真实流量和 DDoS 攻击流量的模拟，通过多次实验来对防护系统进行测试，从可靠性、复杂度、可用性和可拓展性等方面评估了该防护系统。结果表明，本文提出的由触发检测和深度检测结合的 DDoS 检测方案在不同类型的实验中均取得了很好的表现，检测准确率平均达 97.7%，与现有研究相比较。通过改进的 CUSUM 算法(SMNA-CUSUM)实现低开销、粗粒度的触发检测，并结合 AdaBoost 算法进行高精度、细粒度的深度检测，系统的 CPU 占用率比直接使用机器学习算法下降了约 20%；此外，基于滑动时间窗的特征值构造方案提升了检测率，且未给系统延时造成明显的增加，整个检测系统的延时约为 5.005s。在防御系统中，首先通过动态路由系统实现基于带宽的最优路由，达到了利用网络资源来分散和吸收攻击流量的目的，提升了网络的可用性和抗压能力；攻击阻断系统完成攻击溯源与攻击者报文的实时过滤，且基于黑名单的流表保护机制在阻断结束后成功删除了无效的流表项，在本文的一次实验中，攻击结束后自动删除了 198 条无效流表，占总流表数约 22.7%，节省了交换机的资源。综上所述，该防护

系统具有系统开销小、检测准确率高特性，实用价值较强。

第六章 总结与展望

6.1 本文工作总结

DDoS 攻击目前已成为网络安全的最大威胁之一,且其影响态势日益严重,传统的静态网络架构是对抗 DDoS 攻击的瓶颈。而 SDN 作为一种新的网络体系,其数控分离、集中控制与动态可编程等特性给网络带来了极大的改变,为对抗 DDoS 攻击提供了新的思路。本文在分析了现有研究成果及其存在问题的基础上,提出了触发检测和深度检测相结合的 DDoS 检测方案,以及路由动态调整和溯源主动阻断相结合的 DDoS 防御方案,并基于所提方案,在 Mininet 平台上实现了基于 SDN 的 DDoS 攻击防护系统。

本文首先分析了 DDoS 攻击的威胁态势以及传统的对抗方案,接着介绍 SDN 的出现及其为对抗 DDoS 攻击带来的新的机会,并分析了目前国内外针对在 SDN 中进行 DDoS 攻击检测和防御的研究现状及不足之处,给出了本文设计的防护系统的意义。此外,深入研究了 SDN 的架构、OpenFlow 协议和相关标准,并介绍了本文所涉及的包括变点检测、CUSUM 控制图以及机器学习等相关算法的背景和理论知识。

对基于 SDN 的 DDoS 检测,本文针对现有检测方案中检测周期过小将导致系统开销大的问题,设计了一种触发检测和深度检测相结合的联合检测方案。首先提出了一种改进的累积和算法(SMNA-CUSUM 算法),通过实时跟踪多维度序列并其基于统计特性的累积偏差进行异常检测,引入滑动窗口来构造特征序列,引入动态阈值针对不同漏警率和虚警率的需求进行自适应调整,引入多维度的特征来降低检测的漏警率。该算法应用于触发检测模块,对全网流量实现粗粒度的异常检测;接着引入时间窗特征,采用 AdaBoost 集成学习算法对异常流量进行细粒度的深度检测,最终确定是否存在攻击。联合检测方案在保障较高精确率的同时降低了系统开销,实验表明,该方案的检测精确度可达 97.7%,同时具有开销小、低漏警率的优点。

对基于 SDN 的 DDoS 防御,本文针对现有防御方案中阻断响应延时、网络可用性不强的问题,提出了一种基于路由动态调整和溯源主动阻断相结合的防御方案。该方案首先采用改进的 K 最短路径算法,实现基于带宽的最优路由,使得整个网络可以动态地根据流量变化来进行路由,得以在面临攻击时充分利用网络资源来分散和吸收攻击流量;同时基于主机行为进行攻击溯源,并采用报文实时过滤方案实施主动阻断。针对现有报文过滤方案中阻断后残留大量无效流表占用交换机资源的问题,提出了一种基于流表保护机制的报文过滤方案。该方案制定针对攻击源的动态流表来实时过滤攻击报文,并为流表项定义较短的闲置超时时间,自动删除无效流表以释放资源。实验表明,该防御方案对攻击报文进行了成功的过滤,且过滤后删除无效流表约 22.7%,节省了交换机的资源。

最后,基于上述检测和防御方案,本文在 SDN 仿真环境中实现了基于 SDN 的 DDoS 攻击防护,并对其进行测试和评估。搭建 Fat-tree 拓扑结构的网络,基于 Iperf 和 Scapy 分别实现了真实流量和 DDoS 攻击流量的模拟,通过多次实验,从可靠性、复杂度、可用性和可拓展性等角度评估该系统。结果表明,检测方案在不同类型的实验中均取得了很好的表现,检测准确率平均达 97.7%,与现有研究相比较;通过结合低开销、粗粒度的触发检测系统与高精度、细粒度的深度检测系统,CPU 占

用率比直接使用深度检测系统下降了约 20%；基于滑动时间窗的特征构造方案提升了检测率，且未给系统延时造成明显的增加，整个检测系统的延时约为 5.005s。在防御方案中，动态路由系统实现基于带宽的最优路由，利用网络资源来分散和吸收攻击流量，提升了网络的可用性和抗压能力；攻击阻断系统完成攻击溯源与攻击者报文的实时过滤，且基于黑名单的流表保护机制在阻断结束后成功删除了无效的流表项，为交换机节约了空间。

6.2 未来研究展望

本文研究了软件定义网络中 DDoS 攻击的防护技术，提出了基于 SDN 的 DDoS 检测方案以及防御方案，并基于所提方案，在 SDN 环境中设计和实现了 DDoS 防护系统。进一步地，通过设计仿真实验，对整个防护系统进行测试和评估。今后的研究可以在以下方面进行改进和完善：

（1）进一步探索 SDN 网络中更有效的网络信息采集方式。防护方案中网络信息的采集利用的是 SDN 中控制平面与转发平面间的安全通道来进行的，虽然实验中采集信息的频率不高，未对其他通信过程造成影响，但当网络业务繁忙时，安全通道的资源紧张，通过其他方式进行信息采集将有效减小安全通道的压力。

（2）通过更多类型的 DDoS 攻击对系统进行测试，在此基础上不断发现问题、改善方案。本文模拟了全球 DDoS 攻击中流量最大的两种攻击：SYN Flood 和 UDP Flood，并取得了较好的性能。但由于不同的攻击原理不同，其导致网络流量的变化也不尽相同，因此通过更多的攻击类型来进行测试将为防护方案的改进提供方向。

（3）建立 DDoS 防护系统通用的评估标准。本文从可靠性、复杂度、可用性和可拓展性等角度较全面地评估了系统，但该评估标准仍缺乏一定的理论基础，更通用的、更具科学性的评估标准有待建立。

（4）将防护系统应用到其他类型 SDN 环境。一方面，由于 Mininet 中进行的实验可以无缝的迁移到真实环境中，接下来可以考虑在真实环境中对系统性能进行测试；另一方面，具有多台控制器的 SDN 网络是目前研究的重点领域，也有着更广泛的应用场景，在多台控制器的 SDN 网络中部署防护系统有着重大意义。

参考文献

- [1] Shin Seungwon, Yegneswaran Vinod, Porras Phillip, *et al.* AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks[C]. CCS '13. New York, NY, USA:2013. 413-424.
- [2] Yuan Bin, Zou Deqing, Yu Shui, *et al.* Defending against flow table overloading attack in software-defined networks[J]. IEEE Transactions on Services Computing. 2016..
- [3] Ambrosin Moreno, Conti Mauro, De Gaspari Fabio, *et al.* LineSwitch: Efficiently Managing Switch Flow in Software-Defined Networking While Effectively Tackling DoS Attacks[C]. ASIA CCS '15. New York, NY, USA:2015. 639-644.
- [4] Ambrosin Moreno, Conti Mauro, De Gaspari Fabio, *et al.* Lineswitch: tackling control plane saturation attacks in software-defined networking[J]. IEEE/ACM Transactions on Networking. 2017. 25(2): 1206-1219.
- [5] Afek Yehuda, Bremler-Barr Anat, Shafir Lior. Network anti-spoofing with SDN data plane[C]. IEEE, 2017. 1-9.
- [6] Kalkan K. U. Bra, G U R G. U. Rkan, Alag O Z Fatih. SDNScore: A statistical defense mechanism against DDoS attacks in SDN environment[C]. IEEE, 2017. 669-675.
- [7] Boite Julien, Nardin Pierre-Alexis, Rebecchi Filippo, *et al.* StateSec: Stateful Monitoring for DDoS Protection in Software Defined Networks[C]. Bologna, Italy:2017.
- [8] Rebecchi Filippo, Boite Julien, Nardin Pierre-Alexis, *et al.* Traffic monitoring and DDoS detection using stateful SDN[C]. IEEE, 2017. 1-2.
- [9] Wang Rui, Jia Zhiping, Ju Lei. An entropy-based distributed DDoS detection mechanism in software-defined networking[C]. IEEE, 2015. 310-317.
- [10] Conti Mauro, Gangwal Ankit, Gaur Manoj Singh. A comprehensive and effective mechanism for DDoS detection in SDN[C]. IEEE, 2017. 1-8.
- [11] Wang Xiulei, Chen Ming, Xing Changyou, *et al.* Defending DDoS Attacks in Software-Defined Networking Based on Legitimate Source and Destination IP Address Database[J]. IEICE TRANSACTIONS on Information and Systems. 2016. 99(4): 850-859.
- [12] Giotis Kostas, Argyropoulos Christos, Androulidakis Georgios, *et al.* Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments[J]. Computer Networks. 2014. 62: 122-136.
- [13] Mousavi Seyed Mohammad, St-Hilaire Marc. Early detection of DDoS attacks against SDN controllers[C]. IEEE, 2015. 77-81.
- [14] Mousavi Seyed Mohammad, St-Hilaire Marc. Early Detection of DDoS Attacks Against Software Defined Network Controllers[J]. Journal of Network and Systems Management. 2017. 1-19.
- [15] Feinstein Laura, Schnackenberg Dan, Balupari Ravindra, *et al.* Statistical approaches to DDoS attack detection and response[C]. IEEE, 2003. 303-314.
- [16] Dao Nhu-Ngoc, Park Junho, Park Minho, *et al.* A feasible method to combat against DDoS attack in SDN network[C]. IEEE, 2015. 309-311.
- [17] Siris Vasilios A. Papagalou Fotini. Application of anomaly detection algorithms for detecting SYN flooding attacks[C]. IEEE, 2004. 2050-2054.
- [18] 舒远仲, 梅梦喆, 黄文强, 等. SDN 环境下基于条件熵的 DDoS 攻击检测研究[J]. 无线互联科技. 2016. (5): 75-76.
- [19] SFlow. Traffic Monitoring using sFlow. <https://sflow.org/sFlowOverview.pdf>, 2003.
- [20] Nguyen Hoai-Vu, Choi Yongsun. Proactive detection of DDoS attacks utilizing k-NN classifier in an anti-DDoS framework[J]. International Journal of Electrical, Computer, and Systems Engineering. 2010. 4(4): 247-252.

- [21] Braga Rodrigo, Mota Edjard, Passito Alexandre. Lightweight DDoS flooding attack detection using NOX/OpenFlow[C]. IEEE, 2010. 408-415.
- [22] Umarani S. Sharmila D. Predicting application layer DDoS attacks using machine learning algorithms[J]. International Journal of Computer, control Quantum and information Engineering. 2014. 8(10).
- [23] Ashraf Javed, Latif Seemab. Handling intrusion and DDoS attacks in Software Defined Networks using machine learning techniques[C]. IEEE, 2014. 55-60.
- [24] Kokila R. T., Selvi S. Thamarai, Govindarajan Kannan. DDoS detection and analysis in SDN-based environment using support vector machine classifier[C]. IEEE, 2014. 205-210.
- [25] Wang Bing, Zheng Yao, Lou Wenjing, *et al.* DDoS attack protection in the era of cloud computing and software-defined networking[J]. Computer Networks. 2015. 81: 308-319.
- [26] 肖甫, 马俊青, 黄洵松, 等. SDN 环境下基于 KNN 的 DDoS 攻击检测方法[J]. 南京邮电大学学报: 自然科学版. 2015. 35(1): 84-88.
- [27] 左青云, 陈鸣, 王秀磊, 等. 一种基于 SDN 的在线流量异常检测方法[J]. 西安电子科技大学学报. 2015. 42(1): 155-160.
- [28] Xu Yang, Liu Yong. DDoS attack detection under SDN context[C]. IEEE, 2016. 1-9.
- [29] Da Silva Anderson Santos, Wickboldt Juliano Araujo, Granville Lisandro Zambenedetti, *et al.* Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn[C].: IEEE, 2016. 27-35.
- [30] Nanda Saurav, Zafari Faheem, DeCusatis Casimer, *et al.* Predicting network attack patterns in SDN using machine learning approach[C]. IEEE, 2016. 167-172.
- [31] Barki Lohit, Shidling Amrit, Meti Nisharani, *et al.* Detection of distributed denial of service attacks in software defined networks[C]. IEEE, 2016. 2576-2581.
- [32] Niyaz Quamar, Sun Weiqing, Javaid Ahmad Y. A deep learning based DDoS detection system in software-defined networking (SDN)[J]. arXiv:1611.07400. 2016..
- [33] Tang Tuan A., Mhamdi Lotfi, McLernon Des, *et al.* Deep learning approach for network intrusion detection in software defined networking[C]. IEEE, 2016. 258-263.
- [34] Cui Yunhe, Yan Lianshan, Li Saifei, *et al.* SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks[J]. Journal of Network and Computer Applications. 2016. 68: 65-79.
- [35] Abubakar Atiku, Pranggono Bernardi. Machine learning based intrusion detection system for software defined networks[C]. IEEE, 2017. 138-143.
- [36] Radware. DefenseFlow – Software Defined Networking Application. <http://www.radware.com/Products/DefenseFlow/>, 2014.
- [37] Wang Xiulei, Chen Ming, Xing Changyou. SDSNM: A software-defined security networking mechanism to defend against DDoS attacks[C]. IEEE, 2015. 115-121.
- [38] Buragohain Chaitanya, Medhi Nabajyoti. FlowTrApp: An SDN based architecture for DDoS attack detection and mitigation in data centers[C]. IEEE, 2016. 519-524.
- [39] 李鹤飞. 基于软件定义网络的 DDoS 攻击检测方法和缓解机制的研究[D]. 上海: 华东师范大学. 2015.
- [40] Miao Rui, Yu Minlan, Jain Navendu. NIMBUS: cloud-scale attack detection and mitigation[C]. ACM, 2014. 121-122.
- [41] Thomas Ciza, Sharma Vishwas, Balakrishnan N. Usefulness of DARPA dataset for intrusion detection system evaluation[C].: International Society for Optics and Photonics, 2008. 69730G.
- [42] Ioannidis John, Bellovin Steven M. Implementing Pushback: Router-Based Defense Against DDoS Attacks.[C]. 2002.
- [43] Mirkovic Jelena, Prier Gregory, Reiher Peter. Attacking DDoS at the source[C]. IEEE, 2002. 312-321.
- [44] Jin Cheng, Wang Haining, Shin Kang G. Hop-count filtering: an effective defense against spoofed DDoS traffic[C]. ACM, 2003. 30-41.
- [45] Mirkovic Jelena, Reiher Peter. A taxonomy of DDoS attack and DDoS defense mechanisms[J]. ACM

-
- SIGCOMM Computer Communication Review. 2004. 34(2): 39-53.
- [46] Anderson Tom, Roscoe Timothy, Wetherall David. Preventing Internet denial-of-service with capabilities[J]. ACM SIGCOMM Computer Communication Review. 2004. 34(1): 39-44.
- [47] Douligeris Christos, Mitrokotsa Aikaterini. DDoS attacks and defense mechanisms: classification and state-of-the-art[J]. Computer Networks. 2004. 44(5): 643-666.
- [48] Peng Tao, Leckie Christopher, Ramamohanarao Kotagiri. Survey of network-based defense mechanisms countering the DoS and DDoS problems[J]. ACM Computing Surveys (CSUR). 2007. 39(1): 3.
- [49] Zargar Saman Taghavi, Joshi James, Tipper David. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks[J]. IEEE communications surveys and tutorials. 2013. 15(4): 2046-2069.
- [50] Long Hui, Shen Yao, Guo Minyi, *et al.* LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks[C]. IEEE, 2013. 290-297.
- [51] Kim Hyojoon, Feamster Nick. Improving network management with software defined networking[J]. IEEE Communications Magazine. 2013. 51(2): 114-119.
- [52] Akyildiz Ian F., Lee Ahyoung, Wang Pu, *et al.* A roadmap for traffic engineering in SDN-OpenFlow networks[J]. Computer Networks. 2014. 71: 1-30.
- [53] Dixit Advait, Hao Fang, Mukherjee Sarit, *et al.* Towards an elastic distributed SDN controller[C]. ACM, 2013. 7-12.
- [54] McKeown Nick, Anderson Tom, Balakrishnan Hari, *et al.* OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review. 2008. 38(2): 69-74.
- [55] Leiwo Jussipekka, Zheng Yuliang. A method to implement a denial of service protection base[C]. Springer, 1997. 90-101.
- [56] Spatscheck Oliver, Peterson Larry L. Defending against denial of service attacks in Scout[C]. 1999. 59-72.
- [57] Lau Felix, Rubin Stuart H., Smith Michael H., *et al.* Distributed denial of service attacks[C]. IEEE, 2000. 2275-2280.
- [58] Garg Aman, Reddy AL Narasimha. Mitigation of DoS attacks through QoS regulation[J]. Microprocessors and Microsystems. 2004. 28(10): 521-530.
- [59] Juels Ari, Brainard John. Cryptographic countermeasures against connection depletion attacks[M]. Google Patents, 2007.
- [60] Leiwo Jussipekka, Zheng Yuliang. A method to implement a denial of service protection base[C]. Springer, 1997. 90-101.
- [61] Gil Thomer M., Poletto Massimiliano. MULTOPS: A Data-Structure for Bandwidth Attack Detection.[C]. 2001. 23-38.
- [62] Mahajan Ratul, Bellovin Steven M., Floyd Sally, *et al.* Controlling high bandwidth aggregates in the network[J]. ACM SIGCOMM Computer Communication Review. 2002. 32(3): 62-73.
- [63] Barford Paul, Kline Jeffery, Plonka David, *et al.* A signal analysis of network traffic anomalies[C]. ACM, 2002. 71-82.
- [64] Mirkovic Jelena. D-WARD: source-end defense against distributed denial-of-service attacks[D]. University of California, Los Angeles. 2003.
- [65] Mirkovic Jelena, Prier Gregory, Reiher Peter. Attacking DDoS at the source[C]. IEEE, 2002. 312-321.
- [66] Andersen David, Balakrishnan Hari, Kaashoek Frans, *et al.* Resilient overlay networks[M]. ACM, 2001.
- [67] Yan Jianxin, Early Stephen, Anderson Ross. The xenservice-a distributed defeat for distributed denial of service[C]. 2000.
- [68] Page Ewan S. Continuous inspection schemes[J]. Biometrika. 1954. 41(1/2): 100-115.
- [69] Van Trung Phan, Huong Truong Thu, Van Tuyen Dang, *et al.* A multi-criteria-based DDoS-attack prevention solution using software defined networking[C]. IEEE, 2015. 308-313.
- [70] Dayal Neelam, Srivastava Shashank. Analyzing behavior of DDoS attacks to identify DDoS detection features

- in SDN[C]. IEEE, 2017. 274-281.
- [71] Siris Vasilios A., Papagalou Fotini. Application of anomaly detection algorithms for detecting SYN flooding attacks[C]. IEEE, 2004. 2050-2054.
- [72] Freund Yoav, Schapire Robert E. A decision-theoretic generalization of on-line learning and an application to boosting[J]. Journal of computer and system sciences. 1997. 55(1): 119-139.
- [73] Yen Jin Y. An algorithm for finding shortest routes from all source nodes to a given destination in general networks[J]. Quarterly of Applied Mathematics. 1970. 27(4): 526-530.
- [74] Alfares Mohammad, Loukissas Alexander, Vahdat Amin. A scalable, commodity data center network architecture[J]. acm special interest group on data communication. 2008. 38(4): 63-74.

致谢

九龙湖畔，七年将至。回首往昔，感慨万千。在硕士论文即将完成之际，我最想说的，就是对所有给予过我帮助、关心和支持的大家表示感谢。

首先，最应当感谢我的导师宋宇波教授。宋老师学识渊博、谈吐风趣、为人亲切，亦师亦友。从毕业论文的课题选择、探索研究到论文撰写，都是在宋老师的精心指导和悉心关怀下完成的。每次遇到瓶颈时，他会给我提出很多宝贵的建议以及发散性的想法，让我受益匪浅。能够成为他的学生我感到非常幸运，在此，我要对宋老师致以最衷心的感谢和敬意！

同时，我要感谢信息安全研究中心胡爱群等老师们在我研一上课期间无私的传道授业解惑，是你们的付出为我进入信息安全领域打下了坚实的基础。

感谢小伙伴杨丽娟和李晶琪的陪伴与关心，和你们在一起傻笑、吃饭、拌嘴是我每天最开心的时光，也是值得珍藏的美好回忆。感谢室友樊浩、张苏春家人一样的关心和照顾。感谢好友姚艳和朱芳枚，即使你们远在四牌楼，每次的欢聚让我感到无比愉悦和放松。感谢师兄高尚、师姐李古月、好友陈飞翔、张驰远和吕荣毅在我完成毕业论文期间对我的指导和帮助。

感谢师兄王润、杜周、张天阳，师姐戚姗姗、秦艳荣，你们在我刚到实验室时对我照顾有加，让我感受到了实验室的温暖。感谢张克落、董启宏、蒋程、李莹莹和罗平等同窗好友，与你们一起学习和科研的日子不再孤单。感谢师弟武威、黄强、魏一鸣、刘袁、宋睿、石伟和李轩，你们让实验室充满了欢声笑语。感谢信安的各位同学在生活和学习中的陪伴，让我感受到信安大家庭的温馨。

特别感谢我的父母，你们的关心、支持和鼓励是我学习和工作最坚强的后盾，是我遇到困难时坚持前行的动力。感谢你们对我无微不至的关爱，我爱你们！

最后，非常感谢评审老师百忙之中抽出时间审阅本论文！

作者简介

杨慧文（1992—），女，汉族，江苏东台人，现为东南大学信息安全硕士研究生，研究方向为网络安全。

- 攻读硕士学位期间发表的论文

- [1] 杨慧文, 宋宇波. 基于 WAP PUSH 的手机安全性检测系统[C], 第 31 届南京地区研究生通信年会, 2016.