

面向安全的 SDN 架构的研究与设计

摘 要

SDN 作为一种全新的网络设计理念，与网络安全联系紧密。一方面，它的出现给传统的网络安全研究带来了很多新的思路和解决方式；另一方面，其具有的集中控制性和开放性也会产生很多新的安全问题，比如北向接口的开放性给控制层和应用层之间的安全带来了极大的威胁。恶意应用可以随意接入控制层，访问网络状态信息和操纵网络流量，破坏网络的正常状态，威胁网络安全。

针对 SDN 应用可以随意接入网络，访问网络资源，威胁网络安全这种情况，本文提出了一种面向应用的 SDN 安全架构。本架构具有应用访问控制的安全功能，在控制层和应用层之间加入应用访问控制层，对接入网络的 SDN 应用进行身份认证，权限检查，基于属性的访问控制。

本文对提出的面向应用的 SDN 安全架构进行了设计，重点阐述了 SDN 应用访问控制系统的框架设计和模块划分，还介绍了一种基于属性的 SDN 应用访问控制决策算法，并对系统框架进行了具体的实现，详细说明了各个功能模块的实现方式。最后，将访问控制系统置于 SDN 整体安全架构中进行测试，测试结果表明系统能够对应用进行身份认证、权限检查和基于属性的访问控制，同时也证明了本文提出的安全架构实现了预期的功能。

关键词 SDN 安全架构 SDN 应用 身份认证 权限管理 访问控制

Research and Design of Security Architecture for SDN

ABSTRACT

SDN are closely linked with network security as a new network design concept. On the one hand, its emergence has brought a lot of new ideas and solutions to the traditional network security research; the other hand, its centralized control and openness will also have a lot of new security issues, such as the openness of the north interface poses a great threat to the security between the control layer and the application layer. Malicious applications can freely access the control layer, access network status information and manipulate network traffic, undermine the normal state of the network, threatening network security.

Aiming at the situation that SDN application can access the network, access network resources and threaten network security at will. This paper presents an application-oriented SDN security architecture. This architecture has the security function of applying access control. It adds the application access control layer between the control layer and the application layer, and performs identity authentication, permission checking and attribute-based access control for the SDN application of the access network.

In this paper, the SDN security architecture is designed, and the framework design and module partitioning of SDN application access control system are described. An attribute-based SDN application access control decision algorithm is introduced, and the system framework is achieved. the implementation of each function module is described in detail. Finally, the access control system is placed in the SDN overall security architecture for testing. The test results show that the system can perform identity authentication, permission checking and attribute-based access control. It also proved that the proposed security architecture to achieve the desired function.

KEY WORDS SDN security architecture SDN application Authentication
Authority management Access control

目 录

第一章 绪论	1
1.1 研究背景	1
1.2 国内外研究现状	2
1.3 论文研究内容	3
1.4 论文内容安排	3
第二章 相关技术研究	4
2.1 面向安全的 SDN 架构	4
2.2 SDN 控制器调研分析与选型	6
2.3 访问控制技术	7
2.4 相关开发技术介绍	9
第三章 面向应用的 SDN 安全架构设计	12
3.1 安全架构简述	12
3.2 SDN 应用访问控制系统设计	14
3.2.1 需求分析	14
3.2.2 框架设计	15
3.2.3 模块设计	16
3.3 算法描述	21
第四章 SDN 应用访问控制系统的实现	23
4.1 总体实现	23
4.2 主要功能模块实现	24
4.2.1 逻辑控制层各模块实现	24
4.2.2 前端视图层各模块实现	27
4.2.3 数据存储层各模块实现	31
第五章 系统测试	32
5.1 功能测试	33
5.1.1 网络管理员系统登陆功能测试	34
5.1.2 应用身份信息注册与查询功能测试	34
5.1.3 应用身份信息修改和注销功能测试	34
5.1.4 权限初始化和查询功能测试	34
5.1.5 权限增加与移除功能测试	34
5.1.6 访问控制策略创建功能测试	35
5.1.7 网络视图显示功能测试	35

5.1.8 应用身份认证功能测试.....	35
5.1.9 应用权限检查功能测试.....	37
5.1.10 XACML 访问控制功能测试	38
5.1.11 日志记录功能测试.....	39
5.2 性能测试.....	40
第六章 总结和展望.....	41
6.1 总结.....	41
6.2 展望.....	42
参考文献.....	43
致 谢.....	44

第一章 绪论

1.1 研究背景

随着大数据，云计算等新技术的发展，云服务提供商为了满足不同的用户的各种网络服务需求(如带宽、服务质量、安全或可靠性)，需要网络体系结构具有较高的弹性和灵活性，网络资源可以通过网络虚拟化功能进行灵活地分配。然而，在传统的网络体系结构中，常用的网络设备，如路由器或交换机等，比较封闭，具有以下缺点：a)软件和硬件紧密耦合；b)集成到设备的网络协议过于复杂；c)几乎所有设备都是制造商专有的，改变设备的功能或者更新设备十分困难。并且随着网络规模的不断增加，上述特征使传统的网络越来越复杂，导致云服务提供商不能有效地自定义和优化网络资源，无法实现特定的用户需求。面对云时代、大数据时代的高效、灵活的业务承载需求，传统网络的网络架构日益臃肿，逐渐暴露出管理运维复杂、网络封闭，部署新业务困难，设备更新和维护代价较大等一系列缺点。

在传统的网络架构具有诸多限制的背景下，业界一直在研究和开发更加开放的新型的网络架构，促进网络逐渐向智能、开放、优化整合等方向转变。这种转变推动 SDN 软件定义网络的兴起。软件定义网络作为一种新型的网络架构，逻辑上集中的控制层面能够支持网络资源的灵活调度，灵活开放的接口能够支持网络资源的按需调用，将部分或全部网络功能软件化，更好地开放给用户，让用户更好地使用和部署网络，以适应快速变化的云计算、大数据以及更多的创新业务。SDN 技术是一种对运营商网络具有重大影响的高新技术，其价值已经被业界普遍认可，并在近些年发展迅速。

SDN 作为一种全新的网络设计理念，与网络安全联系紧密。一方面，它的出现给传统的网络安全研究带来了很多新的思路和解决方式；另一方面，其具有的集中控制性和开放性也会产生很多新的安全问题，比如控制器、基础设施层、控制器与应用层之间以及控制器和转发设备之间的安全问题等。随着对 SDN 架构开发和部署的不断深入，安全性问题成为制约其发展的一个重要因素，越来越多的国内外学者开始研究 SDN 的安全性问题。SDN 的开放性和可编程性为网络管理带来便利的同时，不可避免地也给整体的网络架构带来了潜在的安全威胁，比如在北向接口的应用接入方面，恶意应用可以随意接入控制层，访问网络状态信息和操纵网络流量，破坏网络的正常状态，威胁网络安全。攻击者可以通过北向接口暴露出来的漏洞，直接向控制器发起攻击。

针对这些安全问题，我们需要对接入控制层的 SDN 应用进行访问控制，包括授权、认证、隔离等。防止恶意应用对网络的随意访问，增强北向接口的安全性。在应用层和控制层之间，如何有效的对应用进行访问控制，是目前 SDN 安全问题研究的一个重要方面，得到了国内外研究者的普遍关注。

1.2 国内外研究现状

SDN 架构实现了传统网络架构中网络管理功能的集中，是对传统网络架构的革命性创新。这种创新除了带来管理、运营等方面的灵活性，也带来了一系列的安全问题。

按照 SDN 的体系架构进行划分，SDN 面临的安全问题可以划分为 5 类：应用层安全、北向接口安全、控制层安全、南向接口安全和数据转发层安全。应用层的安全威胁主要有未经身份认证的应用程序、恶意流规则下发、恶意应用程序的非法访问，配置缺陷等。北向接口的安全威胁主要有接口标准化问题和对应用的认证问题，涉及到非法访问和数据泄密等。控制层的安全问题有 DDoS/DoS 攻击，单点故障，恶意/虚假流规则的注入，非法访问，配置缺陷等。南向接口的安全问题包括 SSL/TLS 协议本身的不安全性，默认配置的不安全性（如 OpenFlow 1.3.0 后将 TLS 设置为可选项）。数据转发层安全问题的主要表现形式有 DDoS/DoS 攻击，恶意/虚假流规则的注入，非法访问、配置缺陷等。

针对 SDN 各层存在的安全威胁，目前的解决方案主要分为 5 类^[1]：

1. 安全控制器的设计

对 SDN 控制器进行完全重新的设计和开发，将安全性作为控制器的核心功能。主要的实现成果有 Rosemary 控制器和 PANE 控制器。

2. 安全模块/框架的设计

分为两个方面，一方面是基于已有的某特定的 SDN 控制器，改进其现有的安全模块的设计。另一方面是基于已有的 SDN 控制器，设计一套完整的安全框架，其中各类安全功能模块可进行独立开发，并可作为应用安装在该框架内（安全模块可进行组合）。代表性成果有 FortNOX、SE-Floodlight、FRESCO 和 OFX。

3. DDoS/Dos 攻击防御

有四类解决思路：基于流量特征变化的检测防御、基于连接迁移机制的防御、基于威胁建模分析的防御以及基于攻击目标的防御，如主机 IP 地址的动态变换。目前的研究成果有轻量级 DDoS 洪泛攻击检测方法(SOM), TopoGuard 检测模型, AVANT-GUARD 检测模型。

4. 流规则的合法性和一致性检测

有两种解决思路：基于角色和优先级的检测和基于形式化方法的检测，即借助数学模型或工具。代表性的成果有流规则检测模型 VeriFlow，检测系统 FlowChecker 等。

5. 北向接口的安全性

主要是对应用的认证和安全管理，包括权限管理，访问控制和可恢复性等。代表性成果有 FortNOX，SE-Floodlight 和 PermOF。FortNOX 实现了一个基于角色的访问控制管理模块，依据不同的角色对流策略进行授权。SE-Floodlight 是对 FortNOX 的强化，具有一个安全执行内核(SEK)，使北向接口的 API 具有数字认证功能。管理员需要预先对 OpenFlow 应用的 java 类进行签名，应用在运行时由 SEK 进行数字验证。一旦签名和验证，就可以修改或查询网络资源，网络上进行通信。

1.3 论文研究内容

本文的主要研究内容包括：

- 1、研究 SDN 架构的特点和原理，存在的安全问题以及主要解决思路。
- 2、研究当前已有的 SDN 安全架构；对 SDN 开源控制器和目前主流的访问控制策略进行调研分析与选型。
- 3、提出一种面向应用的 SDN 安全架构，具有对应用进行访问控制的安全功能，增强了北向接口的安全性。
- 4、对面向应用的 SDN 安全架构中的 SDN 应用访问控制系统进行设计，使得 SDN 应用访问控制系统能够对欲访问控制器资源的 SDN 应用进行身份认证、权限管理和基于策略的访问控制。以及应用访问控制决策算法的设计。
- 5、应用访问控制系统和访问控制决策算法的实现。
- 6、将 SDN 应用访问控制系统置于面向应用的 SDN 安全架构中，进行系统测试，验证面向应用的 SDN 安全架构的安全功能。

1.4 论文内容安排

本文共分为 6 章，各章内容安排如下：

第一章 绪论。本章首先介绍 SDN 技术发展背景和 SDN 所面临的安全问题，接着简要分析了 SDN 安全问题的研究现状，然后介绍了论文的研究内容以及论文各章节的内容安排。

第二章 相关技术研究。首先介绍目前已有的 SDN 安全架构，接着介绍了 SDN 控制器，并选取一款控制器作为系统实现的基础，然后介绍了访问控制技术，最后介绍了系统实现所用到的相关开发技术。

第三章 面向应用的 SDN 安全架构的设计。本章首先提出了一种面向应用的 SDN 安全架构，该架构具有应用访问控制的安全功能，并对架构的各层进行了阐述；接着对安全架构中的访问控制系统进行详细的模块设计；最后描述了访问控制决策算法的设计。

第四章 SDN 应用访问控制系统的实现。本章根据之前的系统设计，对应用访问控制系统进行了相应的实现。首先介绍了总体实现流程，然后具体介绍了各模块的实现过程。

第五章 系统测试，本章首先搭建好面向应用的 SDN 安全架构的整体测试环境，包括底层网络的模拟，测试应用的编写，以及将系统模块加入控制器后的启动运行。然后模拟实际情况对应用访问控制系统进行了功能测试和性能测试，验证了安全架构的应用访问控制的安全功能。

第六章 总结与展望，对文章内容进行总结，对面向应用的 SDN 安全架构的研究前景和研究方向进行了展望。

2.1 面向安全的 SDN 架构

FRESCO 是由 Shin 和 Porras 等人设计的一种可组装的模块化安全架构。这种安全框架允许开发人员在控制器上创建新的安全模块,且这些安全模块之间可以相互组合并协同工作,从而加快了控制器安全模块库的设计和开发。FRESCO 集成了大量 API 接口,并能够与传统的安全工具(如 BotHunter 等软件)进行通信。如图 2-1 示,FRESCO 部署在开源控制器 NOX 之上,由应用层和安全执行内核两部分组成,同时,它提供了由 Python 脚本语言编写的 API 接口,使得研究人员可以自己编写具有安全监控和威胁检测功能的 Module 安全模块。Module 安全模块是 FRESCO 安全模块库的基本处理单元,不同的 Module 模块用于提供不同的安全功能。同时,这些模块可以被共享或组合,以提供更加复杂的安全防护功能。

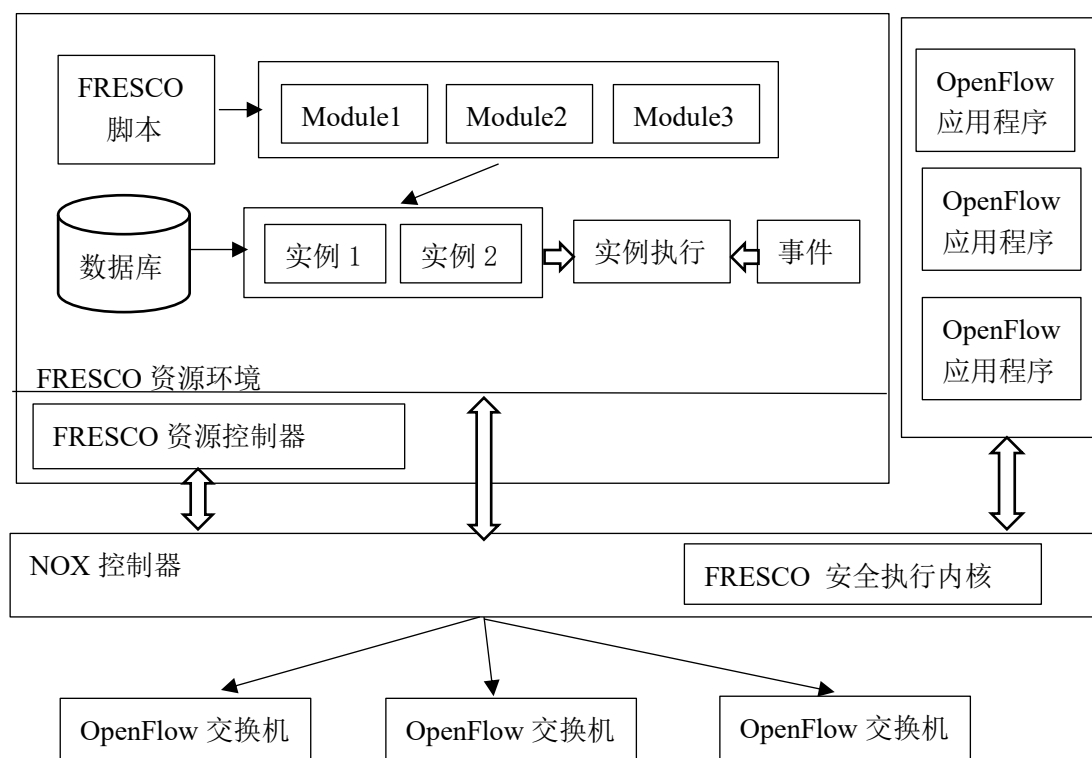


图 2-1 NOX 控制器上的安全框架 FRESCO

SDSA一种将安全数据与控制分离的新型安全架构，包括安全控制器、北向安全APP和南向安全设备。架构如图2-2所示，各类安全APP处于安全架构的顶部，生成并向推送细粒度策略。传统的安全设备和SDN网络设备处于底部，处理网络中的数据和行为；中间的安全控制器是核心组件。在东西方向，安全控制器从网络控制器中获取拓扑和路由信息，从IaaS系统中获取租户和网络信息，建立知识库；在南北方向，安全控制器从安全设备中收集日志，建立日志库和信誉库，并接收各种信息，调度相应安全模块进行响应。SDSA借鉴了软件定义网络的本质思想，利用安全APP、安全控制器和安全设备协同检测SDN环境下的各类威胁，在深度分析安全策略后进行快速响应防护。

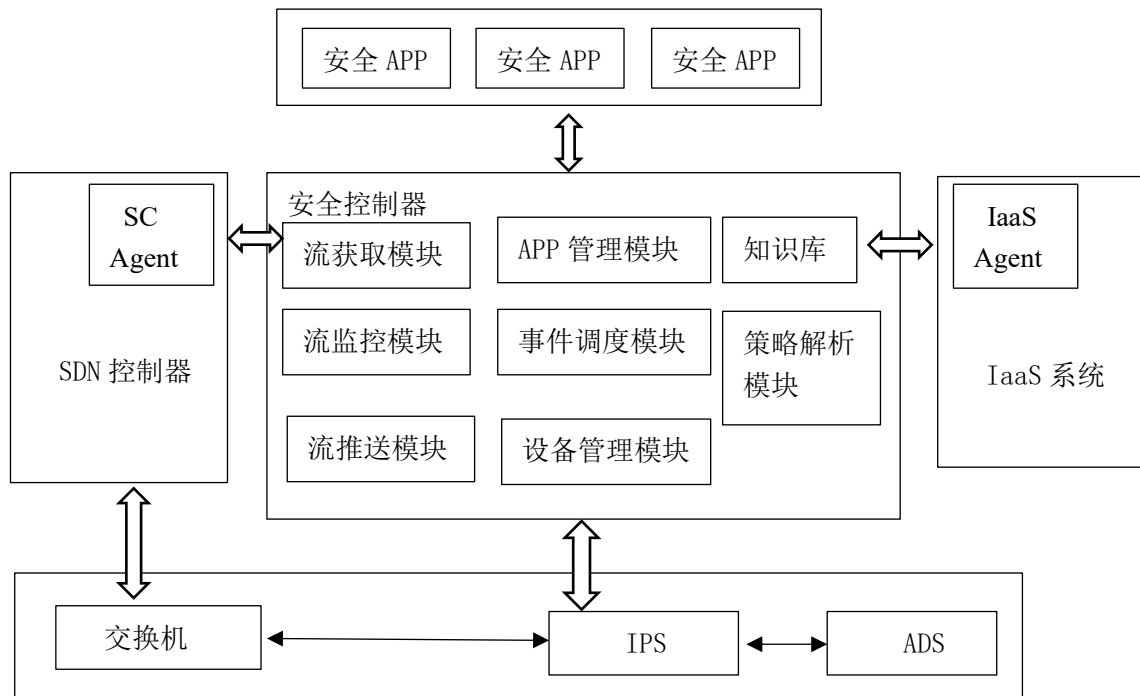


图 2-2 SDSA 安全架构

PermOF 是一种安全隔离框架。如图 2-3 所示，控制器和应用程序被隔离在线程容器中，应用程序从控制器内核中分离出来后不能直接调用控制器的 API。在应用程序和操作系统之间引入访问控制层，由控制器内核进行配置和控制。通过访问控制层来限制非法应用程序对控制器内核资源的直接访问。基于应用程序访问权限最小化的思想，PermOF 分析了控制器上应用程序的 18 种访问权限，设计了一个细粒度的访问控制模块。PermOF 可对应用程序的访问权限进行认证和管理，增强了 SDN 北向接口的安全性。PermOF 通过一个访问控制层，将应用和控制器内核分离开来。提出了一组权限，限制应用程序对控制器资源的直接访问，对 SDN 应用的访问权限进行细粒度的管理。

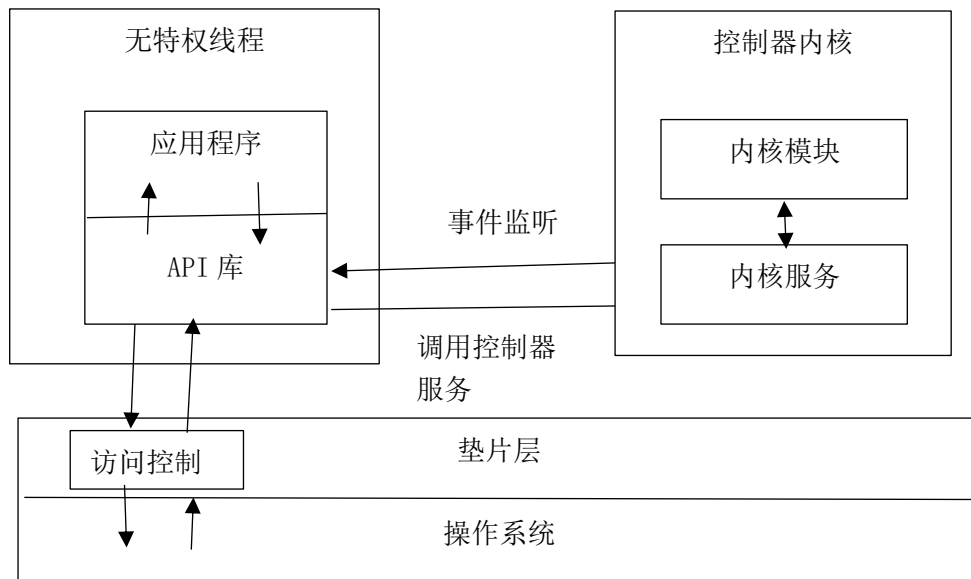


图 2-3 PermOF 安全架构

2.2 SDN 控制器调研分析与选型

SDN 架构的核心层是控制层，而控制层的核心部件则是 SDN 控制器。SDN 控制器使网络控制和网络应用具有可编程性。随着 SDN 技术的不断发展，SDN 控制器也呈现出多样性，主要分为商用控制器和开源控制器两大类。商用控制器有 ActiveBroadbandNetworks 公司的 Active Resource Controller，Adara Networks 公司的 sky，Vmware 的 NSX Controller，华为的 Agile Controller 以及中兴的 ZENIC。开源控制器目前使用较多的有 NOX，POX，RYU，Floodlight，OpenDaylight，ONOS 等。下面将对这几种开源控制器进行详细介绍。

1. NOX/POX

NOX 是业界第一款 OpenFlow 控制器，由 Nicira 开发的，是众多 SDN 研发项目的基础，主要用于研究和教育领域。NOX 的核心模块使用 C++ 编写，上层应用可以用 Python 或 C++ 实现。NOX 采用组件化架构，分为核心组件和应用组件两种类型。POX 是 NOX 的纯 Python 实现版本，支持控制器原型功能的快速开发。目前，NOX 主要面向 Linux 平台，POX 面向 Windows, Linux, Mac OS 等多个平台。NOX/POX 的环境配置和安装使用比较复杂。

2. RYU

由 NTT 开发，能够与 OpenStack 平台整合。由 Python 编写实现，具有丰富的控制器 API，支持网络管控应用的创建。它支持标准的协议，包括 OpenFlow，Netconf 和 OF-config 等。目前，RYU 在 SDN 开发面向云计算的网络产品方面取得了不错的效果。

3. Floodlight

由 Big Switch Networks 开发，遵循 Apache 许可证，是企业级的 OpenFlow 控制器，脱胎于 Beacon。采用 Java 语言编写实现，具有模块化结构，分为控制器模块和应用模

块,同时可以加入自定义功能的模块。支持 OpenStack,在云计算服务领域有很多的应用。此外, Floodlight 还可以支持多个型号的 OpenFlow 交换机,甚至还可以控制混合网络。Floodlight 环境配置和安装使用比较简单,开发文档也比较齐全,便于研究人员进行 SDN 控制器的研究和开发,在科研方面使用较多。

4. OpenDaylight

由 OpenDaylight Project 开发,基于 Java 语言。OpenDaylight 是企业级的控制器,为应用(App)提供开放的北向 API。支持 OSGi 框架和双向的 REST 接口。OSGi 框架提供给与控制器运行在同一地址空间的应用,而 REST API 则提供给运行在不同地址空间的应用。所有的逻辑和算法都运行在应用中。目前,OpenDaylight 已经与众多的开源项目进行了一些整合,包括 OpenDaylight 与 OpenStack、OpenDaylight 与 OPNFV,以及 OpenDaylight 作为一个 SDN 控制器在 ONAP 项目中的重要作用。相比于 Floodlight,OpenDaylight 在控制器的框架特性,源代码等方面比较复杂。

5. ONOS

由 ON.Lab 组织研发,使用 Java 语言实现。是近几年来新出现的一款 SDN 控制器。在 ONOS 出现之前的控制器直接向功能组件发送 OpenFlow 消息,而这些功能组件直接为网络设备创建 OpenFlow 消息,更像是设备驱动。它们不具备一个完整的 SDN 控制器平台所需的性能特征,而 ONOS 是一个真正的一体化的网络操作系统,具有高性能、高可扩展性、高可靠性。与 Floodlight, OpenDaylight 相比,ONOS 具备一个操作系统所具备的所有功能,不仅仅是控制器的功能,是一个重量级的控制器。目前,ONOS 的开发文档和学习资料较少,给研究和学习带来了一定的难度。

上述对目前比较常用的五种开源控制器进行了介绍。从控制器的编程开发语言,控制器功能的完整性,控制器的量级,学习时间等方面综合考虑,Floodlight 使用 Java 语言编写;是一个企业级的控制器,稳定性较强,具有比较完整的功能;同时开发文档丰富,有利于学者快速入门开发。因此,本文选用 Floodlight 作为系统实现所依赖的 SDN 控制器。

2.3 访问控制技术

访问控制技术^[5]是为解决用户访问计算机中存储的数据资源的问题和管理访问权限而被提出的。主要技术有:自主访问控制类型(Discretionary Access Control, DAC)、强制访问控制类型(Mandatory Access Control, MAC)、基于角色的访问控制类型(Role Based Access Control, RBAC)和基于属性的访问控制(Attribute Based Access Control, ABAC)。

1. 自主访问控制(DAC)

自主访问控制是指授权主体即 DO(数据拥有者)拥有对其所拥有资源即客体的分配或者撤销其他用户的访问权限。在采用自主访问控制的系统内,一般是指定的某些特权用户(一般指管理员)或用户组修改管理该策略列表。一般情况下采用访问控制列表 ACL (Access Control List) 管理不同用户对数据的拥有的访问权限,一张表中有不同用户

对这个数据的操作权限。同时 DAC 存在以下缺点：客体复杂问题，即不能对较为复杂的数据资源实现有效保护；访问权限的连续委托问题，即权限不能连续分配；使系统资源开销增大，效率降低，维护更加困难，灵活性不好。

2. 强制访问控制(MAC)

强制访问控制模型（MAC Model）是一种通过安全级别完成数据资源访问控制的技术。MAC 要求超级管理员或者系统对主体（访问者）和客体（数据）设置相关安全级别标签。MAC 决定是否对访问请求授权是通过系统对主、客体的安全级别进行比较来实现的。安全等级设置的标签具有偏序关系，以 SC 表示安全类型是一个偏序关系。TS 对应绝密级别，S 对应秘密级别，信息安全级别上绝密要高于秘密，用偏序关系表示为 $SC(s) \geq SC(o)$ 。一般情况下 DAC 用来保护系统资源，MAC 用来作为补强的安全措施，保证了对数据的安全访问控制。因此强制访问控制模型的适用场景一般是将数据分密级和类进行管理的对安全性要求很高的领域，如：金融银行和军事安全^{[9][12]}等。由其构成的访问控制灵活性和可伸缩性表现一般。

3. 基于角色的访问控制(RBAC)

RBAC 是上世纪 70 年代被提出并很快得到了广泛的关注。1992 年 Ferraiolo 在 AC 模型中引入了角色概念，第一次提出了于角色的访问控制（RBAC）。RBAC 核心思想是了用户和角色相关，角色和权限相关，将用户和权限相分离，用户通过拥有不同角色来实现拥有不同权限^[17]。RBAC 支持三个著名的安全原则：最小权限原则，责任分离原则和数据抽象原则。在 RBAC 中，权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限。这就极大地简化了权限的管理。在一个组织中，角色是为了完成各种工作而创造，用户则依据它的责任和资格来被指派相应的角色，用户可以很容易地从一个角色被指派到另一个角色。角色可依新的需求和系统的合并而赋予新的权限，而权限也可根据需要而从某角色中回收。角色与角色的关系可以建立起来以囊括更广泛的客观情况。从总体上看，RBAC 控制粒度粗，缺乏动态性，不适合开放式的环境

4. 基于属性的访问控制（ABAC）

基于属性的访问控制 ABAC，其核心思想是使用主体、对象、环境和其它相关属性来实现对资源的访问控制和授权，通过属性来定义对服务访问的管理策略。因为它没有直接使用主体和对象之间的关系，所以它可以更好的适应开放的网络环境。与传统的访问控制相比，基于属性的访问控制是基于请求者和资源的属性来进行访问控制的因此具有较好的灵活性和可扩展性。还可以有效地支持匿名访问^[14]。目前，对基于属性的访问控制 ABAC 的已经有较多的研究和实现，使 ABAC 的效率和安全性在不断提高。ABAC 的缺点在于对数据的访问控制是一种粗粒度的访问控制，在动态性控制方面的性能比较差。

5. 访问控制技术对比

表 2-1 常见访问控制技术的优缺点对比

访问控制策略	优点	缺点
自主访问控制	1、简单直观	1、管理分散 2、授权困难 3、效率底下
强制访问控制	1、管理策略简单 2、安全性高	1、缺乏灵活度 2、应用领域窄
基于角色的访问控制	1、安全性高 2、管理方便 3、系统开销小	1、不适合开放式环境 2、控制粒度粗 3、缺乏动态性
基于属性的访问控制	1、适合开放式环境 2、灵活性 3、可扩展性	1、动态性 2、细粒度

考虑到接入网络中的 SDN 应用具有众多属性，请求访问控制器资源时的访问环境和网络资源的状态，也具有一系列属性。因此，对 SDN 应用采用基于属性的访问控制比较合适，在第三章的第三小节对基于属性的应用访问控制决策算法进行了详细描述。

2.4 相关开发技术介绍

1. XACML

XACML 是在 2003 年 2 月由 OASIS（organization for the advancement of structured information standards）制定的一种基于 XML (extensible markup-language) 用于决定请求/响应的通用访问控制策略描述语言和执行授权策略的框架。在基于 Web 的分布式应用环境的安全策略广泛使用 XACML 语言进行表示。

XACML 访问控制主要框架和流程如图 2-4 所示^[9]，现简单介绍如下：

策略执行点(Policy Enforcement Point, PEP): PEP 是在一个具体的应用环境下执行访问控制的实体。具体功能是将各种应用环境下不同访问请求转换为符合 XACML 规范的请求和根据 PDP 判决的请求结果执行相应的判决动作等。

策略决策点(Policy Decision Point, PDP): PDP 是访问控制系统中授权决策的实体。它通过 PAP 和 PIP 对访问请求做出访决策结果。

策略管理点(Policy Administration Point, PAP): PAP 是在访问控制系统中生成和维护数据访问策略的实体。

策略信息点(Policy Information Point, PIP): POP 是可以获取主体、客体和环境的属性信息的实体，并为其他实体提供属性信息

上下文处理器组成:主要功能是完成相关匹配、转换工作。

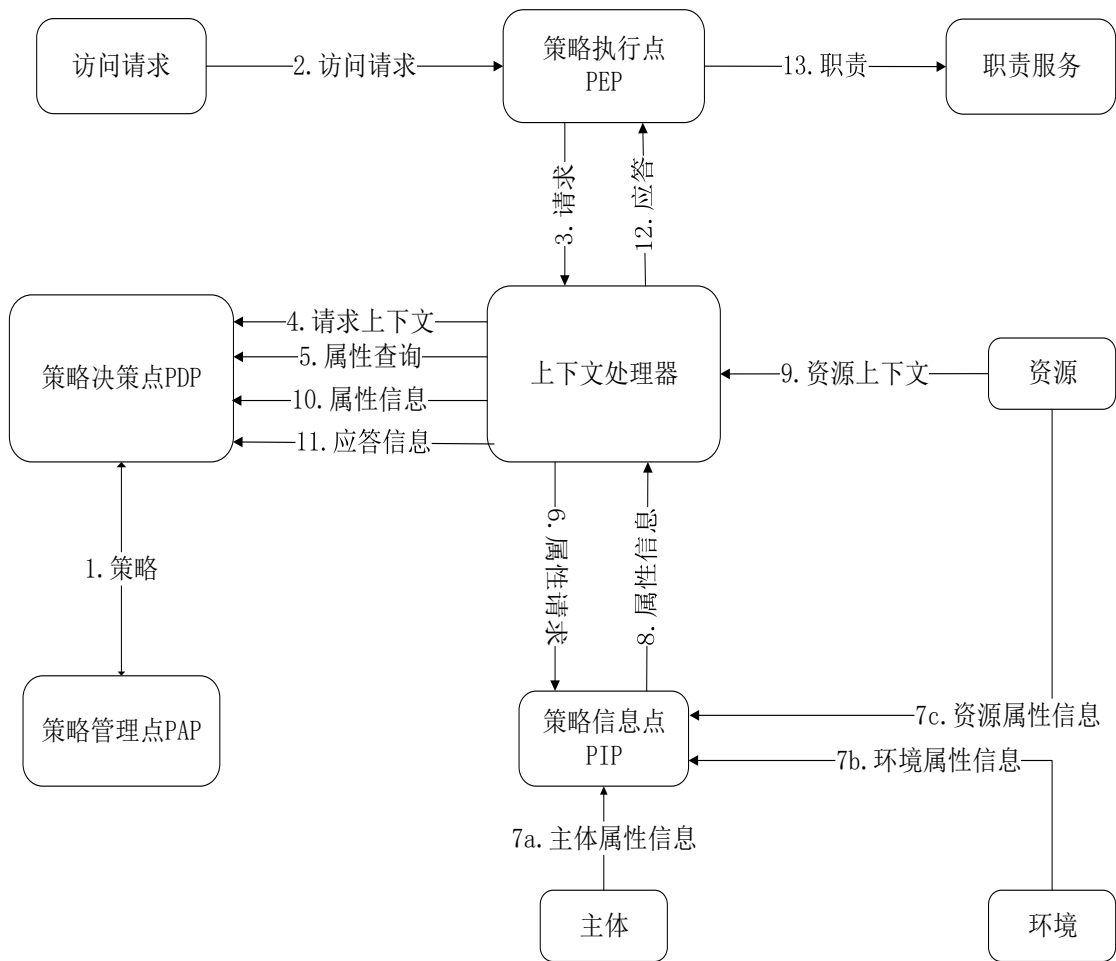


图 2-4 XACML 访问控制框架图

2. Restlet

Restlet 是一个轻量级的 RESTful Web API 框架，基于 Web（REST）的架构风格，通过使用 Restlet 可以开始将 Web 服务，网站和 Web 客户端混合到统一的 Web 应用程序中。Restlet 具有轻巧性和可插拔扩展性，支持所有 REST 概念（资源，表示，连接器，组件等），适用于客户端和服务端 Web 应用程序。同时它还支持主要的 Web 标准，如 HTTP，SMTP，XML，JSON，OData，OAuth，RDF，RSS，WADL 和 Atom。Restlet 通过使用 REST 设计风格模糊了 Web 站点和 Web 服务之间的界限，帮助开发人员构建 Web 应用。每一个主要的 REST 概念（REST concept）都有一个对应的 Java 类。你的 REST 化的 Web 设计和你的代码之间的映射是非常简单直接的。

Restlet 的思想是：HTTP 客户端与 HTTP 服务器之间的差别，对架构来说无所谓。一个软件应可以既充当 Web 客户端又充当 Web 服务器，而无须采用两套完全不同的 APIs。

如图 2-5 所示，Restlet 框架由两个主要部分组成。首先，Restlet API 是一个支持 REST 和 HTTP 概念的中立 API，便于处理客户端和服务端应用程序的调用。此 API 由 Restlet Engine 支持。API 和实现之间的这种分离与 Servlet API 和 Web 容器（如 Jetty 或 Tomcat）之间或 JDBC API 与具体 JDBC 驱动程序之间的分离类似。

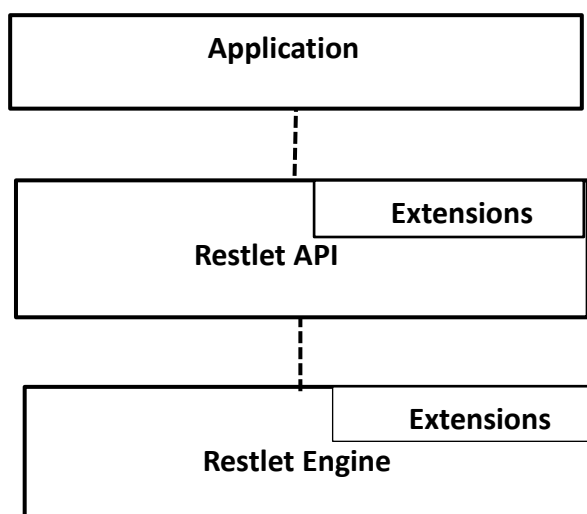


图 2-5 Restlet 框架图

3. Logback

Logback 是由 log4j 创始人设计的一个开源日志组件。logback 当前分成三个模块：logback-core, logback-classic 和 logback-access。logback-core 是其它两个模块的基础模块。logback-classic 是 log4j 的一个改良版本。此外 logback-classic 完整实现 SLF4J API，使开发者可以很方便地更换成其它日志系统如 log4j 或 JDK14-Logging。logback-access 访问模块与 Servlet 容器集成提供通过 Http 来访问日志的功能。Logback 需要与 SLF4J 结合使用，SLF4J 是 The Simple Logging Facade for Java 的简称，是一个简单的日志门面抽象框架，它本身只提供了日志 Facade API 和一个简单的日志类实现。Logback 将日志分为 TRACE、DEBUG、INFO、WARN 和 ERROR 等五个级别，可以把日志按照级别的高低，记录到相应级别的日志文件中。

第三章 面向应用的 SDN 安全架构设计

3.1 安全架构简述

针对北向接口方面，未经认证的应用随意访问网络资源的安全问题，本章设计了一种面向应用的 SDN 安全架构，实现了对 SDN 应用的访问控制，包括身份认证与识别、权限管理与检查、基于属性的访问控制，增强了北向接口的安全性。面向应用的 SDN 安全架构设计如图 3-1 所示：

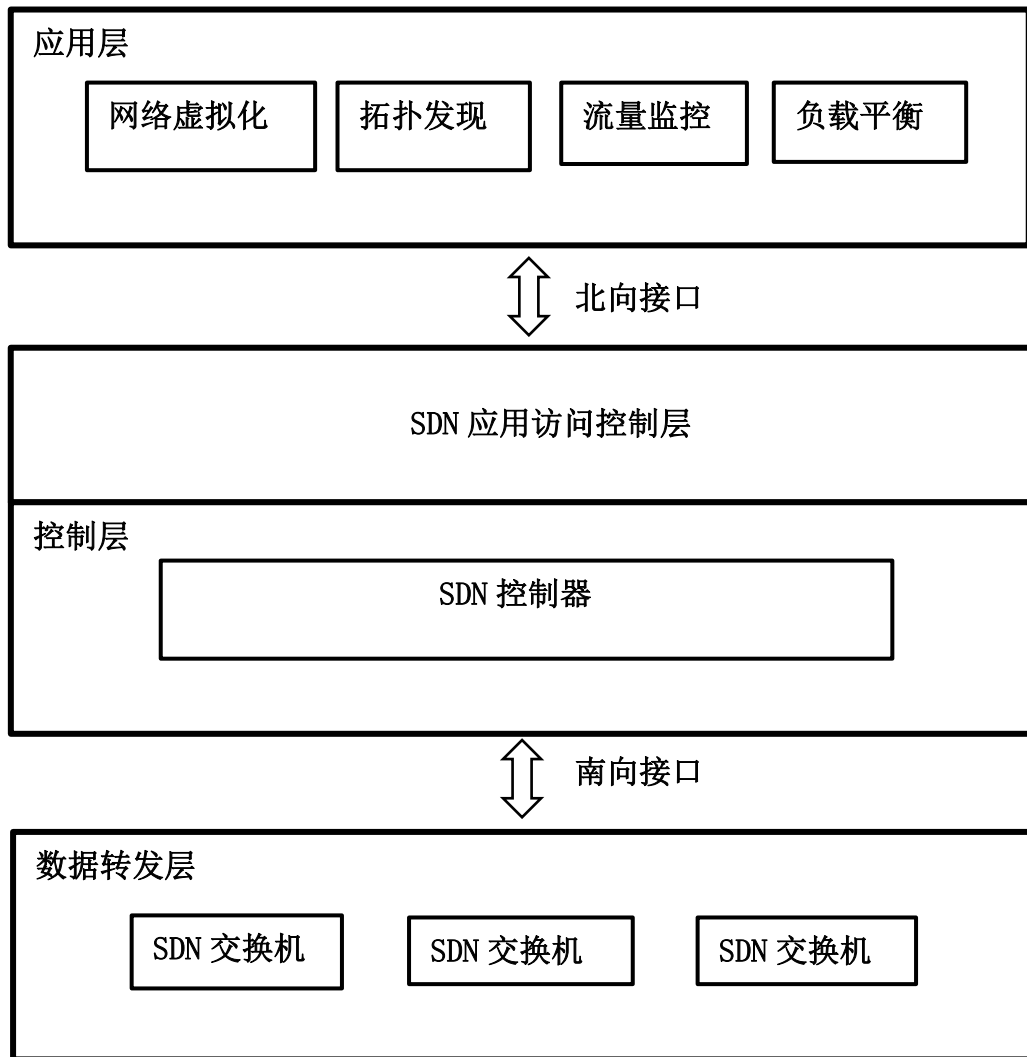


图 3-1 面向应用的 SDN 安全架构

由图 3-1 可以看出，面向应用的 SDN 安全架构是在传统的 SDN 网络架构^[7]的基础上，通过增加 SDN 访问控制层来实现 SDN 整体安全架构的安全功能。一共分为 4 层，分别是数据转发层，控制层，应用层和应用访问控制层。其中，数据转发层，控制层，应用层和 SDN 网络传统架构中的三层结构相同，数据转发层主要是底层的基础网络设

备，包括交换机和路由器等，控制层通过 SDN 控制器管控整个网络体系，实现各种网络需求的 SDN 应用位于应用层。由架构的整体设计来看，本安全架构的重点在于 SDN 应用访问控制层。SDN 应用访问控制层位于控制层和应用层之间，将 SDN 应用和控制器资源隔离开，对接入控制层的 SDN 应用进行访问控制，防止未经审计的应用的随意访问。下面将对 SDN 安全架构的各层进行详细介绍。

1. 数据转发层

数据转发层由许多 SDN 交换机组成，它们通过有线或无线媒体进行物理连接。每个交换机都是一个负责转发网络数据包的简单设备，并有一个转发表，叫做流表，它包含数千条用于制定转发决策的规则。流表中的每个规则项由三个字段组成：操作、计数器和模式。当接收到数据包时，交换机将搜索它的流表，查找与字段相匹配的规则。一旦交换机找到了这样的规则，规则的计数器就会增加，相应的规则就会被执行。否则，该交换机将通知控制器请求帮助，或者干脆丢弃数据包。转发规则不是由交换机节点本身生成的，而是由控制器从控制层向下推送的。

2. 控制层

控制层是作为 SDN 的大脑，管理和控制整个网络。实现这些功能的网络节点叫做 SDN 控制器。SDN 控制器与交换机通过一个标准的南向协议进行通信，比如 OpenFlow。SDN 控制器对数据转发层中整个网络拓扑结构，包括交换机、链路以及各种路由协议等，有一个全局网络抽象视图。SDN 控制器运行时向数据转发层下发指令，控制数据转发层的数据转发。

3. 应用层

应用层允许网络操作员快速响应各种业务需求。在应用层可以开发满足各种需求的 SDN 应用，如网络虚拟化、拓扑发现、流量监控、安全防护、负载平衡等等。应用层通过北向接口的 API 与控制层进行通信，比如 REST API。控制层为应用层提供了一个底层网络资源的抽象视图，网络运营商可以在 SDN 控制器上编写相关应用来改变数据包的转发路径，而不是去手动配置所有的物理交换机。

4. SDN 应用访问控制层

SDN 应用访问控制层是本安全架构的核心层。本层能够对欲访问控制器上网络资源的 SDN 应用进行身份认证，防止未经审计的恶意应用的随意访问；接着对通过认证的应用，依据其所具有的权限，对其访问请求进行权限检查，防止越权访问；在应用通过权限检查后，根据访问控制策略，对应用进行基于属性的访问控制判决，判决结果为允许的 SDN 应用才能最终成功地访问网络资源，实现其网络功能。从总体上，本层通过对应用的身份认证、权限检查、基于属性的访问控制这三个过程，实现了面向应用的 SDN 安全架构的应用访问控制功能。

根据本 SDN 安全架构的安全功能和总体框架，数据转发层、控制层、应用层不需要进行模块划分和设计，以下将主要对 SDN 安全架构的核心层-- SDN 应用访问控制层（即 SDN 应用访问控制系统）进行具体的设计和实现。

3.2 SDN 应用访问控制系统设计

3.2.1 需求分析

在控制层和应用层之间，应用程序通过北向接口与控制器进行交互主要包括两个方面：读取网络状态（Reading Network State）和写入网络策略（Writing Network Policies）。所面临的基本的安全问题是恶意应用程序随意访问网络状态信息和操纵网络流量，破坏网络的正常状态，威胁网络安全。针对这种情况，本系统（SDN 应用访问控制层）需要对 SDN 应用进行应用身份信息的注册和权限的授予、以及创建较为灵活的基于属性的访问控制策略，当应用访问控制器上的资源时，先对应用进行身份认证，防止未经注册的非法应用对控制器的访问尝试；然后对经过身份认证的应用进行权限检查，确保应用只能访问具有访问权限的那些控制器资源，对权限范围外的控制器资源的试图访问予以拒绝，防止越权访问，最后对通过权限检查的应用依据之前设定好的访问控制策略进行访问控制判决，应用通过判决后才能访问控制器资源，读取相关的网络状态或写入网络策略。同时，网络管理人员可以通过系统中的 UI 界面查看控制器上接入的所有应用的状态信息，进行相应的操作。

本 SDN 应用访问控制系统的重点在于对 SDN 应用的身份认证、权限核查、以及进行基于属性的访问控制。因此，本系统实现的功能需求如下：

- 1 网络管理员能够对应用程序进行注册，在注册时除了进行身份信息的记录外，还可以对应用程序的访问权限在初始化时按应用的功能需求进行分配。同时，根据不同情况，创建合适的访问控制策略，为应用的访问控制提供策略依据。

- 2 当应用程序试图访问控制器资源时，对其进行身份认证，若身份信息验证通过则可以继续访问，否则视为非法应用，对访问请求予以阻止，并写入系统异常行为监测日志中。

- 3 当应用程序通过身份认证后，对其试图使用的权限进行检查，若满足权限集合的要求则请求可顺利通过，否则将请求信息视为无效请求，写入系统异常行为监测日志中。在该应用程序的生命周期内，访问控制系统严格按照应用程序访问权限集合对应用程序的每个访问请求进行检验，通过这种细粒度的访问权限检查方法，实现了对应用程序访问行为的监测，防止越权访问。

- 4 当应用程序通过权限检查后，视为合法应用。依据设定好的基于属性的访问控制策略，对应用进行访问控制判决，只有通过判决的应用才能在最终成功地访问控制器上的网络资源。同时将未通过判决的应用的有关信息写入系统异常行为监测日中。

- 5 网络管理员可以对应用程序的身份信息和其所具有的访问权限进行查询、修改和删除。同时，可以重置和修改访问控制策略。

- 6 网络管理员可以通过系统查看到全局的网络状态信息，包括网络拓扑、网络节点和网络设备等。还能够查看接入控制器的所有应用的信息。

- 7 将所有非法的应用访问请求记录到日志文件中以进行审计。

3.2.2 框架设计

依据上一小节的需求分析，提出了系统的整体框架图，如图 3-2 所示：

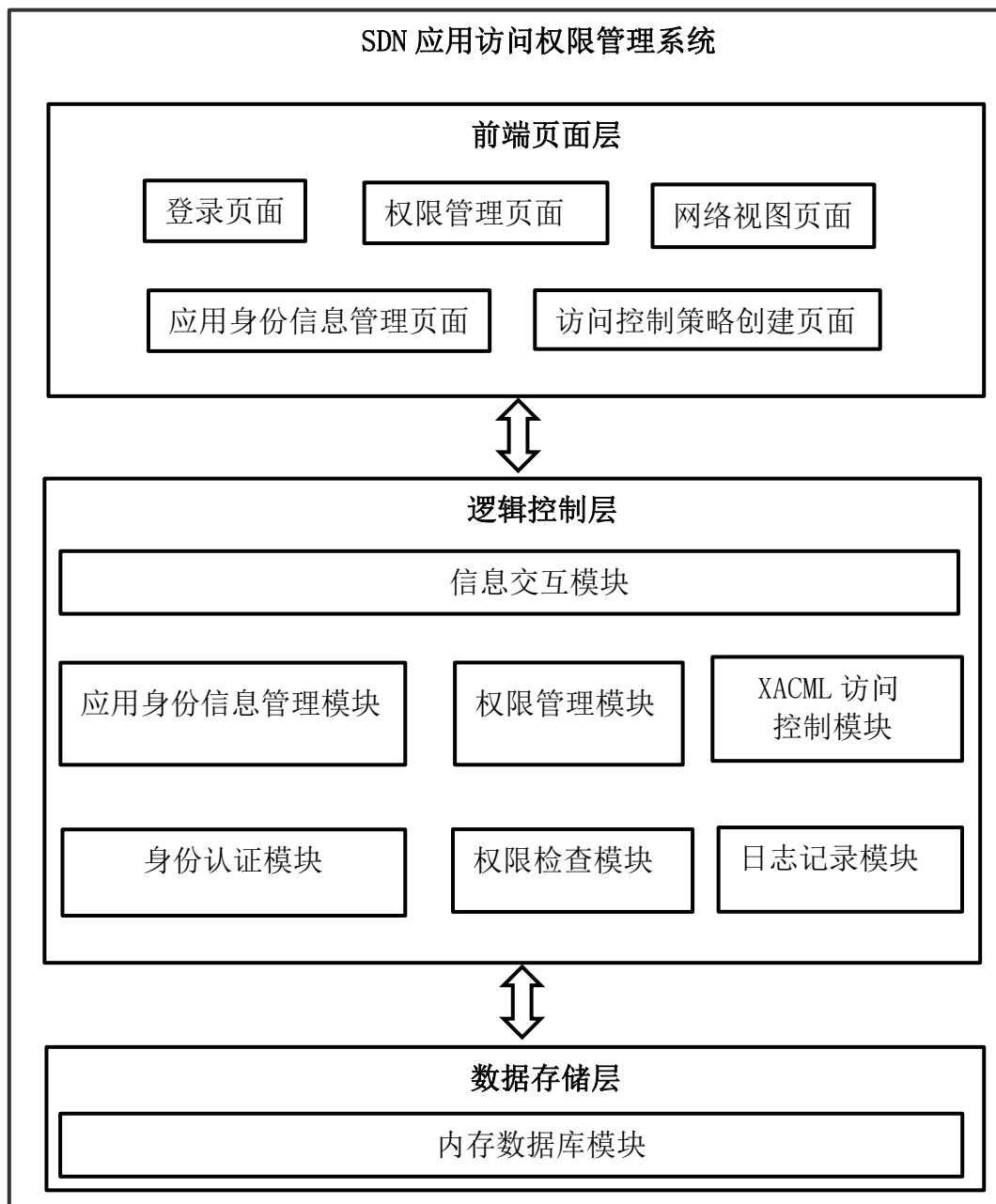


图 3-2 系统总体架构图

由图 3-2 可以看出，本文设计的访问控制系统分为三层，分别是前端视图层、逻辑控制层和数据存储层。

1.前端视图层：网络管理员可以通过登录页面进入系统，能够查看整体的网络状态，包括控制器信息、网络拓扑信息、交换机和主机信息等。同时通过身份信息管理页面、权限管理页面、策略创建页面等三个页面与逻辑层进行数据交互，给逻辑层传递相关的数据信息。并显示应用身份、权限、策略等信息。

2.逻辑控制层：逻辑层作为本系统的核心部分，与前端视图层和数据存储层进行交互，完成对应用访问控制的大部分功能。逻辑控制层完成对访问控制器资源的应用进行身份认证、权限检查、XACML 访问控制。SDN 应用只有全部通过这三个流程的审查后，才能顺利的建立网络策略，获得需要的网络信息。同时逻辑层还负责管理和维护应用的身份信息、权限信息以及访问控制的策略信息。能够把非法的访问请求记录到特定的日志文件中，为以后恶意应用的检测提供依据。

3.数据存储层：负责存储应用的身份信息、权限信息等。由于逻辑层对应用进行身份认证、权限检查时会频繁的查询数据存储层中的数据，为了提高系统认证和检查的效率，数据存储层实际的数据储存是在内存中进行的。

3.2.3 模块设计

3.2.3.1 前端视图层

前端视图层负责系统 UI 界面的显示工作，同时方便网络管理员进行相应的操作。实质上是一个 HTTP 客户端，与逻辑控制层的信息交互模块使用 HTTP 协议进行数据交互。按功能需求可划分为五个模块，分别是管理员登录页面、应用身份信息管理页面、权限管理页面、访问控制策略创建页面、网络视图页面。下面对各模块进行详细的介绍。

管理员登录页面：网络管理人员使用用户名和管理员密码登录 SDN 应用访问控制系统，进行网络的监管。

应用身份信息管理页面：此模块完成应用的注册、应用身份信息列表的显示、应用身份信息的修改和应用注销等功能。

权限管理页面：此模块具有应用权限的授予、应用权限显示、应用权限增加与移除等四个功能。

访问控制策略创建页面：网络管理员可以通过此页面录入有关属性的信息，为逻辑控制层中 XACML 访问控制模块创建基于属性的访问控制策略 policy 提供信息来源。

网络视图页面：网络管理员可以在此模块查看到控制器的运行状态信息和全局的网络视图，包括网络拓扑图、交换机和主机设备等网络节点的详细信息。

3.2.3.2 逻辑控制层

逻辑控制层实现了本系统的核心功能，包括对应用的身份认证，权限检查，基于属性的访问控制等。如图所示，共划分为七个模块，其中身份认证模块、权限检查和 XACML 访问控制模块为本层的核心模块，而应用身份信息管理和权限管理模块则是上述三个重要模块的基础，负责维护所需的信息。日志记录模块负责对非法应用接入、越权访问等事件的监测记录。信息交互模块通过 HTTP 协议，解析前端视图层发送过来的 HTTP 请求，并返回相应的数据。以下是本层七个模块的详细介绍：

1. 应用身份信息管理模块

应用的身份信息（身份证书）的结构如表 3-1 所示：

表 3-1 应用身份信息结构表

APPID	APPNAME	APPKEY	REGISTRY	REGDATE	EXPDAE	ATL
-------	---------	--------	----------	---------	--------	-----

APPID: SDN 应用的 ID,作为其标识, 每个 SDN 应用的 ID 是唯一的。

APPNAME: SDN 应用的名称。

APPKEY: SDN 应用的密钥。

REGISTRY: SDN 应用所属的注册商。

REGDATE: SDN 应用的注册时间。

EXPDATE: SDN 应用的有效期。

ATL: SDN 应用的信用评级, 为以后恶意应用的检测提供一个参考指标。

应用身份信息管理模块实现了对每个 SDN 应用身份信息的增加、删除、更新、查找等四个功能。

2. 权限管理模块

权限定义: Scotthayward^[8]等人通过对控制器的分析, 得出了应用程序的 15 种访问权限, 如表 3-2 所示。主要分为 read、write、notification 三类, 并将每种权限和控制器上特定的资源做了一一映射, 应用程序需要访问该资源, 必须具有对应的权限。

表 3-2 权限目录分类表

Category	Permission	Screening method(s)
Read	read_topology	getAllSwitchMap:Controller.java
		getLinks:LinkDiscoverManager.java
	read_all_flow	getFlows:StaticFlowEntryPusher.java
	read_statistics	getSwitchStatistics:SwitchResourceBase.java
		getCounterValue:SimpleCounter.java
	read_pkt_in_payload	get:FloodlightContextStore.java
	read_controller_info	retrieve:ControllerMemoryResource.java
Notification	pkt_in_event	addToMessageListeners:Controller.java
	flow_removed_event	addListener:ListenerDispatcher.java
	error_event	
Write	flow_mod_route	insertRow:AbstractStorageSource.java
	flow_mod_drop	deleteRow: AbstractStorageSource.java
	set_flow_priority	insertRow: AbstractStorageSource.java
	set_devices_config	setAttribute:OFSwitchBase.java
	set_pkt_out	write:IOFSwitch.java
		writeThrottled:IOFSwitch.java
	flow_mod_modify_hdr	parseActionString:StaticFlowEntries.java
	modify_all_flows	setCommand:OFFlowMod.java

每个 SDN 应用都具有一个权限列表 PERLIST，记录应用所具有的权限集合，描述了应用可以访问控制器资源的最大范围。

权限管理模块负责对应用的权限列表 PERLIST 的初始化、查询、权限增加、权限移除等工作。

3. 信息交互模块

采用 REST 思想，把信息看成一种资源，通过 URL 定义资源的访问方式，通过 GET/POST/PUT/DELETE 方法实现对资源的 CRUD 操作,接受来自前端视图层的 HTTP 请求，响应，最后返回 RESTFUL JSON 格式的数据。

4. 身份认证模块

身份认证模块的流程图如下图 3-3 所示：

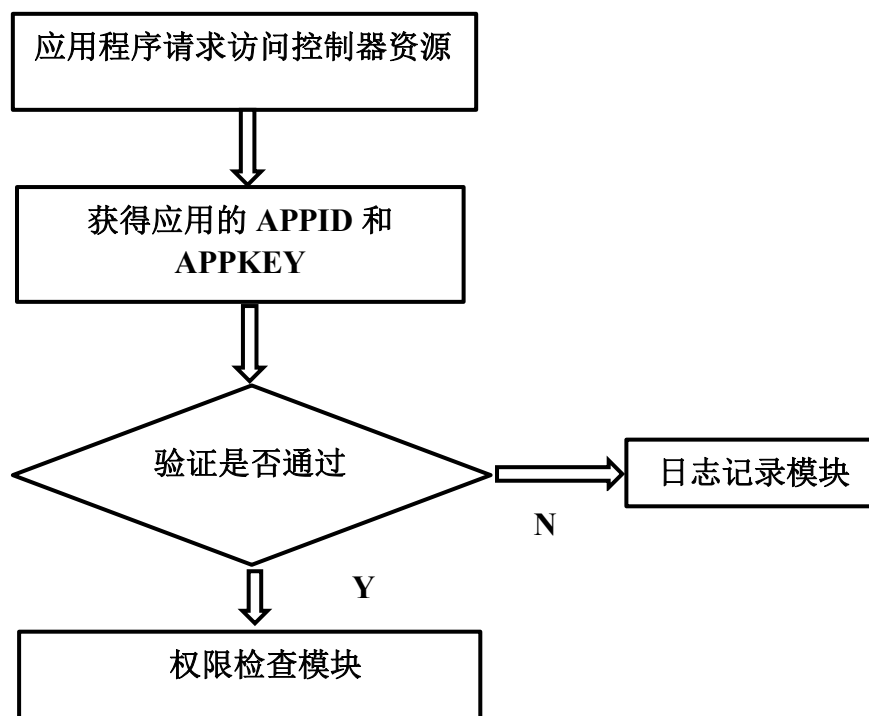


图 3-3 身份认证流程图

应用通过北向接口接入 floodlight 控制器，开始对控制器的资源的发起访问时：

身份认证模块首先获取应用请求中携带的 ID 和应用密钥 KEY，然后利用 APPID 在数据存储层（内存数据库）中查找相关的应用信息，对 ID 和 KEY 进行匹配核对，完成对身份信息的识别，生成认证结果。若识别为未经注册的非法应用，则认证结果为拒绝，同时交给日志记录模块处理；若识别为已注册应用，则认证结果为通过，再交给权限检查模块进行权限的核查。

5. 权限检查模块

权限检查模块的流程图如下图 3-4 所示：

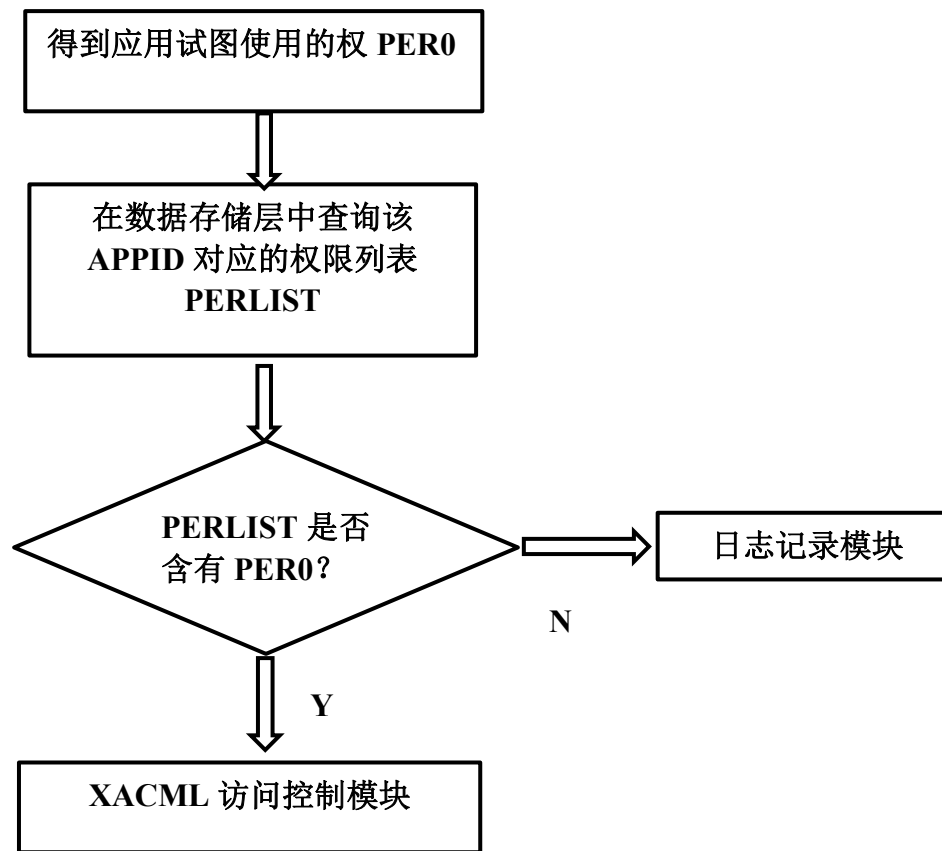


图 3-4 权限检查流程图

应用成功通过身份认证后，首先根据应用的访问请求，得到应用试图使用的权限 PER0，然后在数据存储层（内存数据库）中通过根据应用的 APPID 查找到对应的权限列表 PERLIST，最后判断 PER0 是否在 PERLIST 中，即是否应用具有访问该资源的权限。如果具有，则通过审核，再交给 XACML 访问控制模块进行基于属性的访问请求的判决。如果不具有，则被视为越权访问，应用无法访问该资源，并交给日志记录模块处理。

6. XACML 访问控制模块

XACML 访问控制模块是 SDN 访问控制系统的核心模块，实现了对应用的基于属性的访问控制。网络管理员首先设置好创建基于属性的访问控制策略所需要的相关信息，应用通过身份认证和权限检查后，根据应用和访问环境的属性，查找相应的访问控制策略，进行访问控制判决。若判决通过，则应用可以成功访问，否则拒绝访问。

XACML 访问控制模块的工作流程如图 3-5 所示：

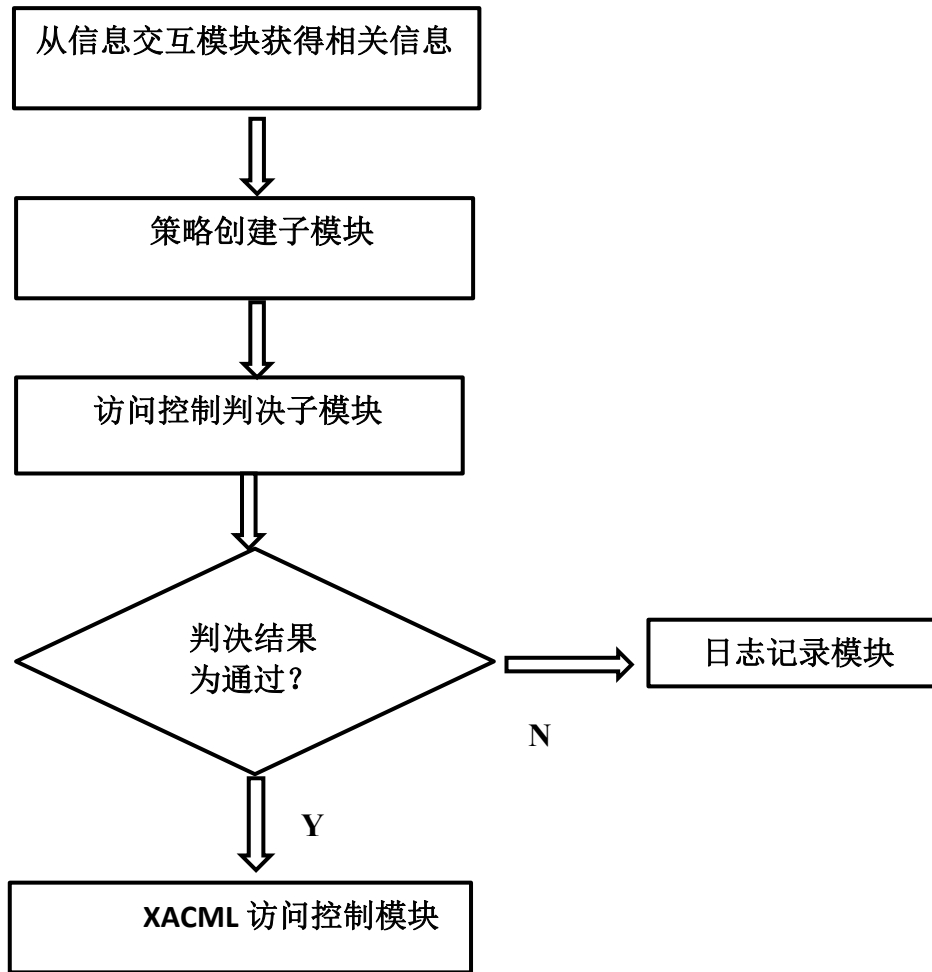


图 3-5 XACML 访问控制模块流程图

由图 3-5 可以看出，XACML 访问控制模块主要分为：策略创建子模块和访问控制判决子模块。策略创建子模块完成基于属性的访问控制策略的创建功能，为访问控制判决子模块提供判决依据。访问控制判决子模块执行基于属性的访问控制算法，产生访问控制的判决结果，这种算法将在 3.3 节中进行详细描述。

7. 日志记录模块

应用的访问请求在身份认证模块、权限检查模块、XACML 访问控制模块中的判决结果为拒绝（即不允许访问）的时候，调用日志记录模块。将这些非法的访问请求记录到日志文件中，为以后恶意应用的检测提供依据。

日志记录的信息采用以下形式：

<date><time><appid>< exception-event><denied-permission>

date 和 time：表示非法访问请求发生的时间。

appid：表示发起访问的 SDN 应用 ID。

exception-event：描述非法访问请求的详细信息，分为三类。分别是应用未通过身份认证、应用没有该权限、应用未通过 XACML 访问控制。

denied-permission：表示应用缺少的权限类型。

3.2.3.3 数据存储层

数据存储层通过自定义的 NOSQL 内存数据库模块完成信息的存储功能。把数据存储在内存中，大大加快了有关数据的查询速率。对于数据查询频繁的身份认证和权限检查模块而言，有效的提高了检查效率，有助于提高系统的性能。

NOSQL 内存数据库是键值(Key-Value)存储数据库，数据表设计分为两种，分别是应用信息表 APPTABLE 和应用权限表 PERTABLE。应用信息表 APPTABLE 使用键 APPID 来对应一个应用的身份信息类 AppIdentityCertificate。应用权限表 PERTABLE 使用键 APPID 来对应一个应用的权限列表 PERLIST。内存数据库还实现了对数据表的增删改查功能。

3.3 算法描述

本文提出了一种基于属性的应用访问控制算法，该算法能够根据基于属性的访问策略进行对应用进行访问控制判决。当应用程序访问控制器资源时，应用的身份属性和访问时外部环境的当前状态信息都可以成为决定访问请求能否被通过的属性。例如应用的注册商、注册日期、信用评级、访问的当前时间等，都可以视为访问控制决策中的重要属性。

参考文献^[9]给出的一个基于 ABAC 的 API 访问控制模型，并对其进行改进，得到了一种基于属性的应用访问控制算法，如图 3-6 所示：算法的输入为 REQ、POL 和 ATT，其中 REQ 表示应用的访问请求，请求中包括应用的身份信息 APP 和当时的访问环境信息 ENV；POL 表示策略的集合，集合中的每个 pol_i 都为设定的一个访问控制策略；ATT 是该次访问的属性集，属性集中的各属性 att_i 可能来自于 REQ，也有可能来自外部环境。算法的输出 Decision 表示对于请求的判决结果。

算法 基于属性的应用访问控制决策算法
输入： $REQ < APP, ENV >$ $POL = \{pol_1, pol_2, \dots, pol_m\}$ $ATT = \{att_1, att_2, \dots, att_m\}$
输出： Decision : Boolean
<pre> BEGIN For all($pol_i \in POL$) do IF $ATT \not\supseteq pol_i$ THEN Decision \leftarrow Deny END IF END For Decision \leftarrow Permit END.</pre>

图 3-6 基于属性的访问控制算法描述图

基于属性的应用访问控制决策算法首先通过策略执行点 PEP 把应用的访问请求转化为 Request，再将 Request 交给策略决策点 PDP 进行评估，最后策略决策点 PDP 查找相应的访问控制策略 Policy 和有关的属性信息，产生评估结果 Decision。若评估结果 Decision 为 true，则允许访问，若是 false 则拒绝。

第四章 SDN 应用访问控制系统的实现

4.1 总体实现

本系统是在 floodlight 的基础上使用 java 语言进行模块扩展来实现的。按照先前的系统设计实现系统的各个模块，将模块的信息添加到 floodlight 的配置文件中，当 floodlight 启动运行时按照配置文件加载各个模块。采用模块管理器 FloodlightModuleLoader 实现对模块的自动加载、启动和初始化功能，首先在配置文件 floodlightdefault.properties 中声明各个模块的路径，然后 FloodlightModuleLoader 读取配置文件，根据路径查找到相应模块，查看该模块启动所需要依赖的其它模块，并将其加入到模块加载列表中，最后 FloodlightModuleLoader 根据模块加载列表依次初始化和启动各个模块。模块的加载过程如图 4-1 所示。

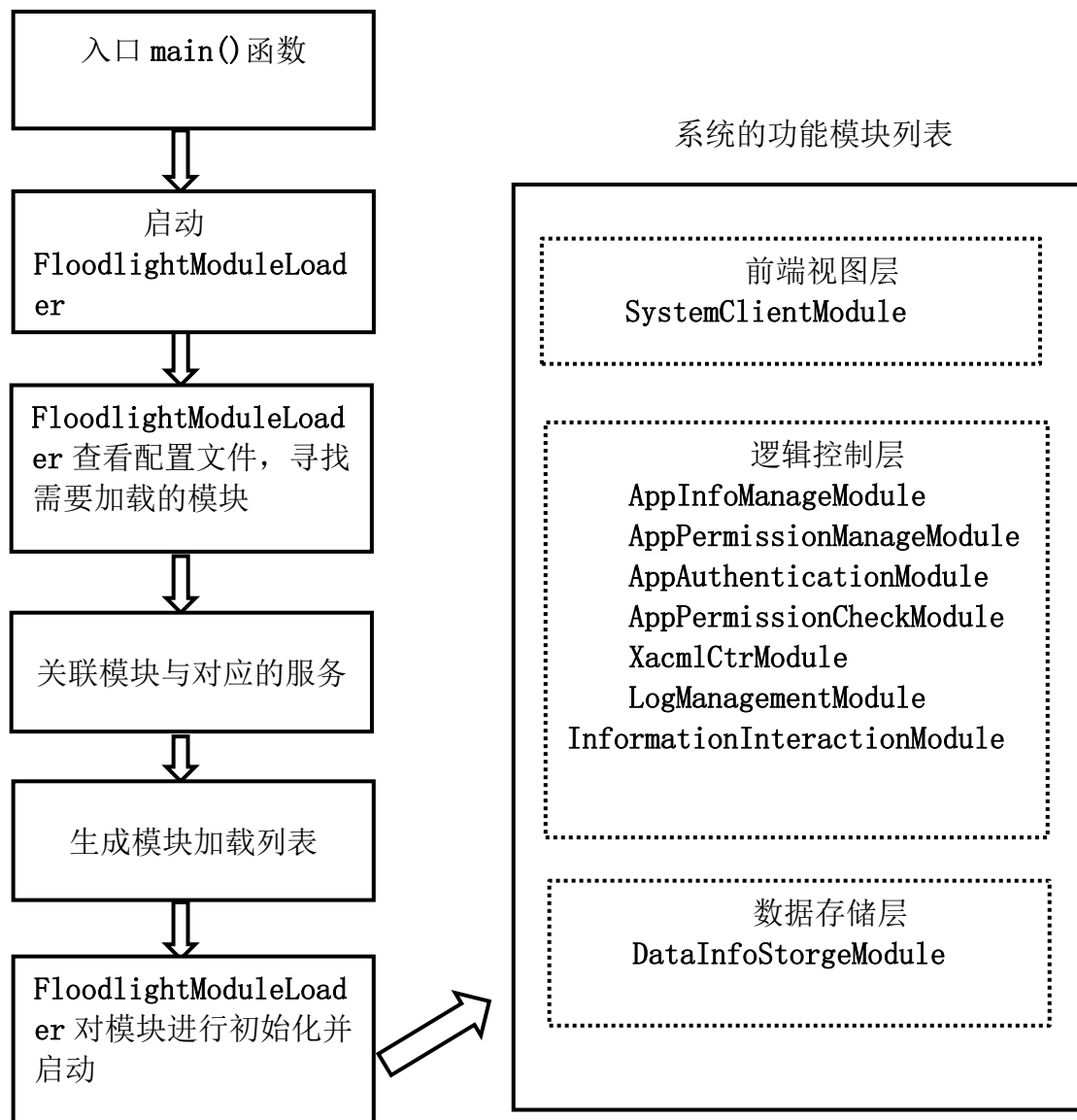


图 4-1 模块的加载流程图

由图 4-1 可以看出系统的功能模块列表分为：

AppInfoManageModule,AppPermissionManageModule,AppAuthenticationModule,AppPermissionCheckModule,XacmlCtrModule,LogManagementModule,InformationInteractionModule,SystemClientModule 和 DataInfoStorgeModule。

逻辑控制层的功能由前七个模块实现，前端视图层的功能由 SystemClientModule 实现，DataInfoStorgeModule 实现数据存储层的功能。

4.2 主要功能模块实现

4.2.1 逻辑控制层各模块实现

逻辑控制层实现了 AppInfoManageModule,AppPermissionManageModule,AppAuthenticationModule,AppPermissionCheckModule,XacmlCtrModule,LogManagementModule,InformationInteractionModule 七个模块。

其中，AppInfoManageModule 实现应用的身份信息管理功能，AppPermissionManageModule 实现应用的权限管理功能，AppAuthenticationModule 实现应用的身份认证功能，AppPermissionCheckModule 实现应用的权限检查功能，XacmlCtrModule 实现基于属性的访问控制功能。

4.2.1.1 AppInfoManageModule 实现

AppInfoManageModule 模块主要实现对应用身份信息的管理，包括应用身份的增加，删除，修改，查询等。主要方法有 applicationRegistration()、applicationUpdate()、applicationCancellation()、applicationFinder()和 applicationFinderAll()。

SDN 应用的身份信息的数据结构表示如下：

```
AppIdentityCertificate{
String  appId; //应用标识
String  appName; //名称
String  appKey; //应用密钥
String  registry; //注册商
String  registrationDate; //注册时间
String  expDate; //有效期
int     ATL //信用级别
}
```

applicationRegistration()方法实现在应用列表 APPLIST 中注册一个新的应用。

applicationUpdate()方法实现对应用列表中应用身份信息的更新。

applicationCancellation()方法实现应用的删除注销功能。

applicationFinder()和 applicationFinderAll()实现对应用身份信息的查询功能。

4.2.1.2 AppPermissionManageModule 实现

AppPermissionManageModule 模块实现对应用权限的初始化, 权限增加, 权限移除, 权限查询等功能。主要方法有:

appPermissionSetInitialization().

appPermissionSetAdd().

appPermissionSetDelete().

appPermissionSetFinderALL().

应用的权限类型定义的数据结构表示如下:

```
PermissionType{read_topology,read_all_flow,read_statistics,read_pkt_in_payload,read_controller_info,pkt_in_event,flow_removed_event,error_event,flow_mod_route,flow_mod_drop,set_flow_priority,set_devices_config,set_pkt_out,flow_mod_modify_hdr,modify_all_flow_s}
```

一共有 15 种权限类型, 每个应用的权限列表 PERLIST 使用 ArrayList 储存, 并通过 hashmap 来实现应用 ID 和权限列表 PERLIST 的一一映射。

AppPermissionSetInitialization() 方法实现对应用权限列表的初始化功能。appPermissionSetAdd() 方法实现对应用权限列表的增加功能, appPermissionSetDelete() 实现对应用权限的移除功能。appPermissionSetFinderALL() 实现应用的权限查询显示功能。

4.2.1.3 AppAuthenticationModule 和 AppPermissionCheckModule 实现

AppAuthenticationModule 负责应用的身份认证功能, 主要方法是 appAuthentication-(String appId,String appKey)。appAuthentication() 通过查询数据存储层中应用的身份信息, 把身份信息中的 ID 和 KEY 与传入的 appId 和 appKey 进行核对, 若 appId 和 appKey 完全正确, 则身份认证成功, 返回认证结果。若失败则交给 LogManagementModule 处理。

AppPermissionCheckModule 实现对应用的权限检查功能。主要方法有 appPermissionSetFinder(String appId,PermissionType permission)。appPermissionSetFinder() 首先通过 appId 在数据存储层查找得到应用的权限列表 PERLIST, 然后再判断 PERLIST 是否含有 permission, 即应用是否具有 permission 权限, 若 PERLIST 含有 permission, 则返回通过的检查结果, 反之则拒绝, 并交给 LogManagementModule 处理。

4.2.1.4 XacmlCtrModule 实现

XacmlCtrModule 主要负责对应用的访问控制功能, 实现基于属性的访问控制算法。采用开源框架 XACML 实现, 主要方法包括 getInfo(), createPolicy(), creatPAP(), createRequest(), createPDP(), findPolicy() 和 getResult()。XacmlCtrModule 模块的实现流程如图 4-2 所示。

XacmlCtrModule 模块的实现流程主要分为七个步骤: 首先 getInfo() 通过信息交互模块得到前端页面网络管理员设定的相关属性策略信息, 然后 createPolicy() 创建访问控制策略 Policy, creatPAP() 负责把生成的 Policy 交给策略管理点 PAP, 接着 createRequest() 把应用的访问请求转化为 Request, createPDP() 创建策略决策点 PDP, PDP 读入 Request

后通过 `findPolicy()` 在 PAP 中查找相应的策略 Policy，最后通过 `getResult()` 得到最后的判决结果。

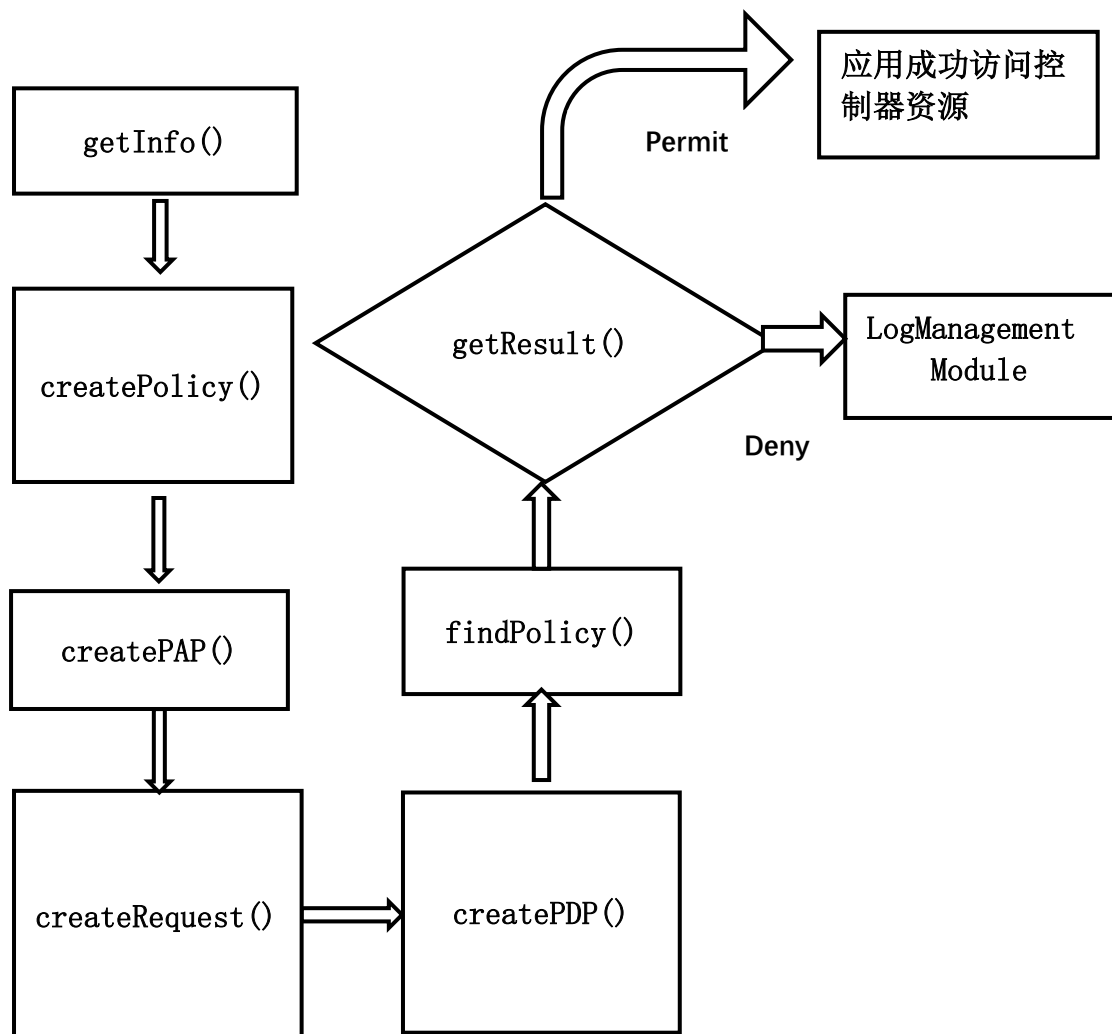


图 4-2 XacmlCtrModule 模块实现流程图

4.2.1.5 LogManagementModule 实现

LogManagementModule 采用 java 日志框架 logback 实现，把异常信息存储在日志文件中。主要方法有 `getLogger()` 和 `logger.info()`。生成的日志文件的相关信息在配置文件 `logback.xml` 中定义。应用的访问请求在身份认证模块、权限检查模块、XACML 访问控制模块中的判决结果为拒绝（即不允许访问）时，LogManagementModule 首先通过日志工厂 `loggerFactory` 的 `getLogger()` 方法生成日志管理器 `logger`，然后日志管理器 `logger` 使用 `logger.info()` 方法把有关信息写入日志文件中，最后关于异常的访问行为的详细信息在日志文件被记录。

4.2.1.6 InformationInteractionModule 实现

InformationInteractionModule 模块负责作为前端视图层的后台模块，实现与客户端的数据交互功能。InformationInteractionModule 模块的实现框架如图 4-3 所示。

采用 Restlet 框架，继承了 Application 类，通过 AppListResource、PermissionListResource、PolicyCreatResource 把将应用身份信息，权限信息，访问控制策略信息封装成 ServerResource 形式的 Rest 资源，定义相应的 URL 访问路径，并实现了资源的 get/put/post/delete 方法。Restlet 中的 Application 监听本地的 8080 端口，SystemClientModule 直接通过 HTTP 请求，实现 InformationInteractionModule 模块中资源的 CRUD 操作。

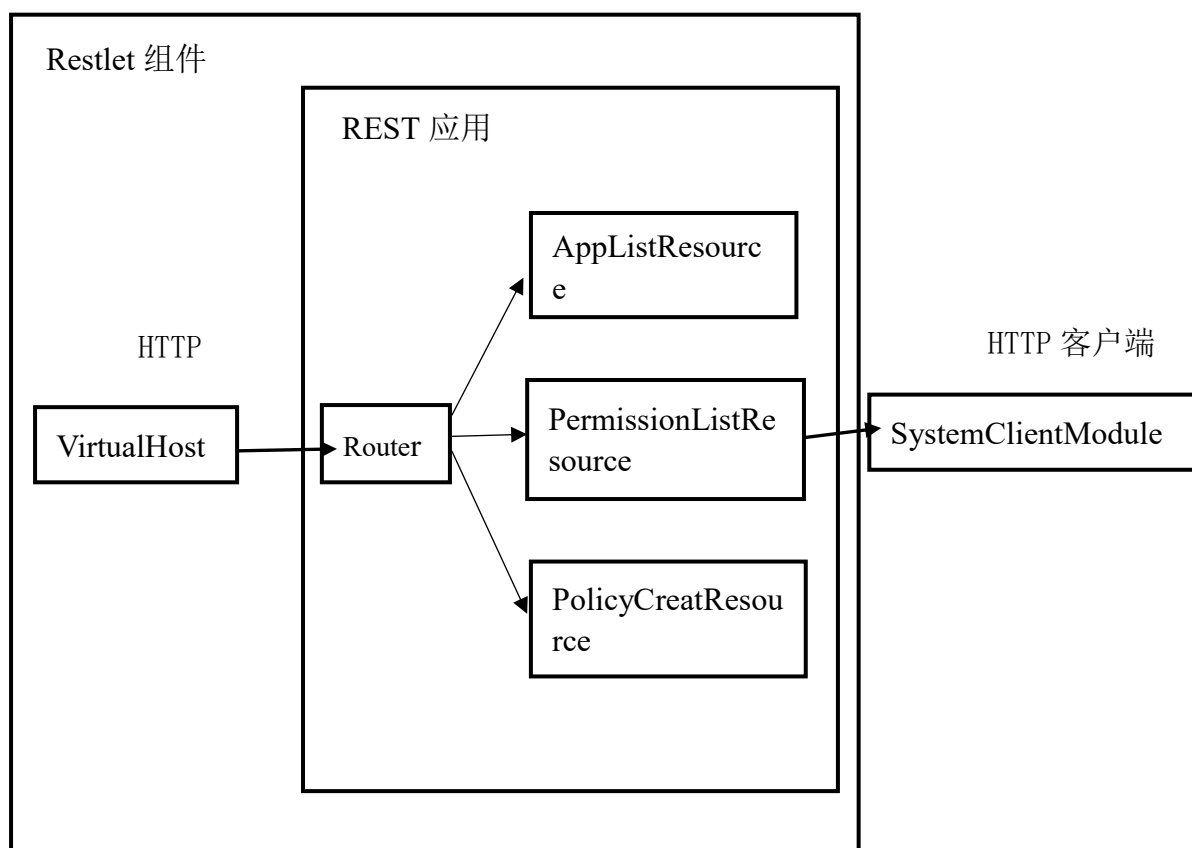


图 4-3 InformationInteractionModule 模块的实现框架

4.2.2 前端视图层各模块实现

前端视图层主要实现 SystemClientModule 模块。SystemClientModule 采用 html/css 和 js 技术编写，使用前端开源工具包 bootstrap 对界面进行美化，利用 javascript 框架 jquery 对页面的动作进行响应。把数据用 ajax 进行封装，向逻辑控制层的 InformationInteractionModule 模块发送 http 请求，并接收 InformationInteractionModule 模块返回的 json 数据，从而实现数据的交互。

SystemClientModule 模块包含五个子模块，分别是分别是管理员登录页面 loginPage、应用身份信息管理页面 appInfoManagePage、权限管理页面 permissionManagePage、访问

控制策略创建页面 `policyCreatePage`、网络视图页面 `networkPage`。下面对各个子模块的实现进行详细介绍。

4.2.2.1 管理员登录页面 `loginPage`

网络管理员在文本框中输入用户名和密码，点击登录按钮后，使用 `jquery` 获取输入的用户名和密码，并对其进行验证。若用户名和密码正确，则登录成功，跳转至系统应用身份信息管理页面 `appInfoManagePage`。实现后的 `loginPage` 如图 4-4 所示：

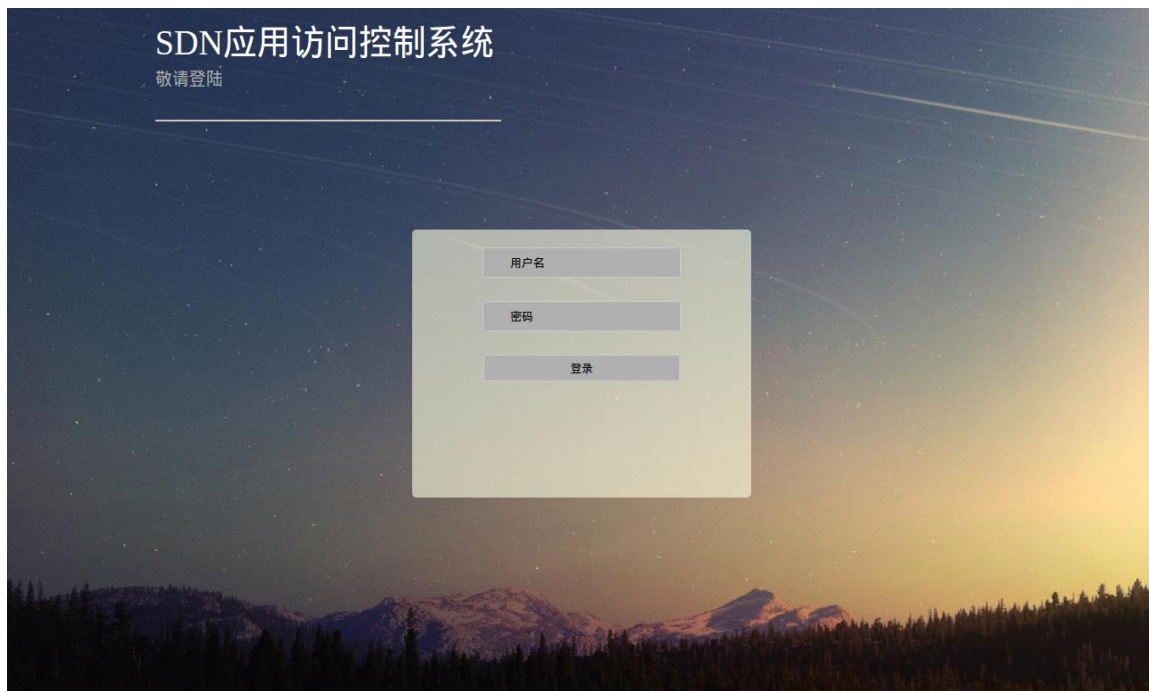


图 4-4 登录页面

4.2.2.2 应用身份信息管理页面 `appInfoManagePage`

`AppInfoManagePage` 实现应用身份信息的查询，注册，修改和注销功能，含有四个方法：

`showAppListInfo()`, `$("#registityButton").click(function())`, `$("#updateButton").click(function())`和`$("#deleteButton").click(function())`。

其中，`showAppListInfo()`使用 `ajax` 向 `InformationInteractionModule` 以 `get` 方式发送 `http` 请求，得到 `json` 格式封装的关于应用身份信息的数据，最后解析并展示在页面上。`$("#registityButton").click(function())`通过表单获得网络管理员输入的注册数据，封装成 `json` 格式后，在 `ajax` 中以 `put` 方式向后台模块发送 `http` 请求，实现应用的注册功能。同理，`$("#updateButton").click(function())`通过表单获得网络管理员要修改的数据，在 `ajax` 中以 `post` 方式向后台模块发送 `http` 请求，实现应用身份信息的修改功能。`$("#deleteButton").click(function())`在 `ajax` 中以 `delete` 方式向后台模块发送 `http` 请求，实现应用注销的功能。图 4-5 是实现后的界面图：



图 4-5 身份信息管理页面

4.2.2.3 权限管理页面 permissionManagePage

和 appInfoManagePage 类似，permissionManagePage 实现应用权限的初始化，增加，移除和查询功能。主要有四个方法：

```
$("#submitButton").click(function());
```

```
$("#checkButton").click(function());
```

```
$("#addButton").click(function());
```

```
$("#removeButton").click(function());
```

\$("#submitButton").click(function())使用 ajax 发送 put 方式的 http 请求，实现应用权限的初始化功能。\$("#checkButton").click(function())使用 ajax 发送 get 方式的 http 请求，实现应用权限的查询功能。\$("#addButton").click(function())和\$("#removeButton").click(function())分别使用 post 和 delete 方式发送 http 请求来实现应用权限的增加和移除功能。图 4-6 是实现后的界面图：



图 4-6 权限管理页面

4.2.2.4 访问控制策略创建页面 policyCreatePage

policyCreatePage 把允许访问的时间段 time-quantum 和允许的应用注册商 registry 等数据转化为 json 格式后，以 ajax 的 put 方式向 InformationInteractionModule 发送 http 请求，为访问控制策略的创建提供必要的信息。实现后的界面图如图 4-7 所示：

图 4-7 访问控制策略创建页面

4.2.2.5 网络视图页面 networkPage

networkPage 通过对 floodlight 的 RestApi 调用，得到 floodlight 控制器，网络拓扑，交换机和网络节点的详细信息，解析并展示在页面上。实现后的界面图如图 4-8 所示：

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:01	/192.168.1.100:60962	Nicira, Inc.	0	0	0	2017/4/18 上午9:19:21

图 4-8 网络视图页面

4.2.3 数据存储层各模块实现

数据存储层实现应用列表 APPLIST 和权限列表 PERLIST 的存储功能。主要模块 DataInfoStorgeModule 使用一个 Hashmap<String tableName,Object table> 存储和管理 APPTABLE 和 PERTABLE 这两张表。APPTABLE 使用 Hashmap<String appid,AppIdentityCertificate app> 存储每个应用的身份信息，PERTABLE 使用 Hashmap<String appid,Arraylist permissionlist > 每个应用的权限列表。在 floodlight 控制器启动时自动加载 DataInfoStorgeModule 模块，分配相应的内存存储空间。

第五章 系统测试

在任何系统的开发中，测试都是其中很重要的一部分。测试有助于发现系统的漏洞和不完善的地方，有助于改进系统功能、提高系统性能。本章对实现后的 SDN 应用访问控制系统进行功能以及性能上的测试，以期发现错误，保证系统的完整性和可靠性。本系统是在 floodlight 的基础上进行模块扩展实现的，测试时通过 mininet 模拟底层测试网络，用 Python 编写 Rest 应用 circuitpusher.py 作为测试应用来验证系统有关的访问控制功能。

测试前的准备工作：

表 5-1 测试环境配置信息表

名称	版本
操作系统	ubuntu Kylin 14.04 Lts
JAVA 环境	jdk-1.7.0_91
IDE	eclipse neon.2
SDN 控制器	floodlight-0.91V
Mininet	2.1.0

测试环境的配置信息如表 5-1 所示，下面将进行测试环境的搭建。

1. 启动运行 floodlight 控制器。

将系统的各个模块加入 floodlight 控制器的模块目录下，使用 ant 命令编译构建得到 floodlight.jar。

在 linux 的终端使用 `java -Dlogback.configurationFile=logback.xml -jar /floodlight-0.91/target/floodlight.jar` 命令启动运行 floodlight。控制器的 IP 地址为 192.168.1.101，监听底层网络的端口为 6633。

2. 搭建底层测试网络。使用 mininet 模拟器创建一个虚拟的网络模拟 SDN 架构中的基础设施层。

创建网络拓扑的命令：`sudo mn --controller=remote,ip=192.168.1.101,port=6633 --mac --topo=tree,2`

在 mininet 中使用 `pingall` 命令使各个主机间相互 ping 通后，在 floodlight 中输入 `nodes` 命令能够查看到 h1~h4,s1~s3 的详细信息。

虚拟网络的拓扑结构如图 5-1 所示：网络设备包括三个交换机和四台主机，四台主机的 IP 地址分别为 10.0.0.1~10.0.0.4，h1 和 h2 连接交换机 s1，h3 和 h4 连接交换机 s3，s1 和 s3 通过交换机 s2 连接。

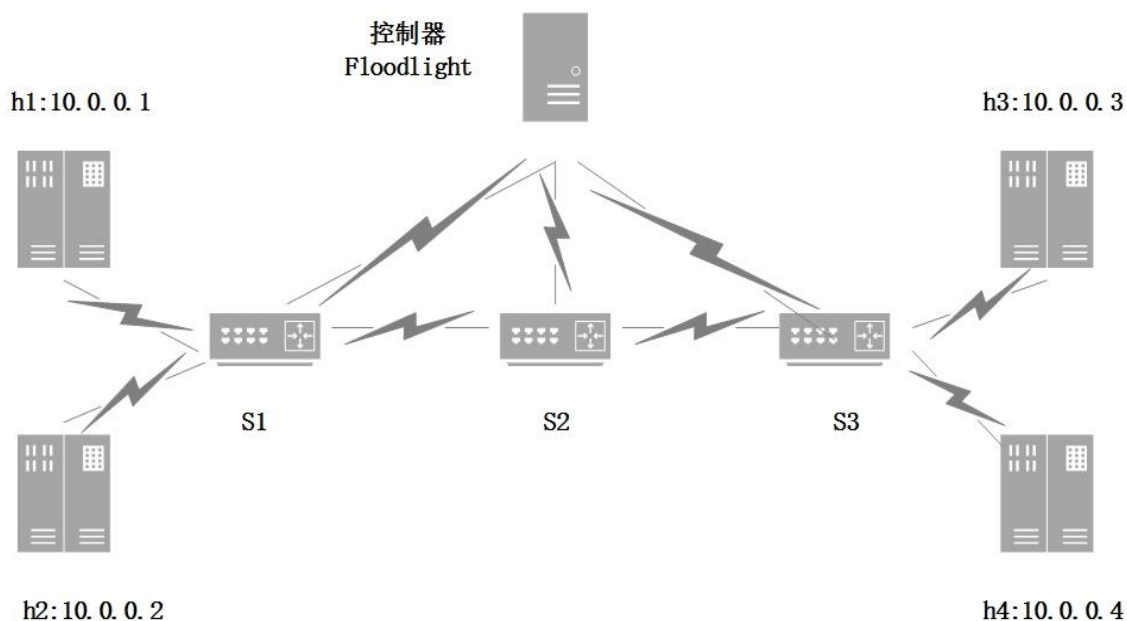


图 5-1 测试网络拓扑图

3. 编写 SDN 应用。

SDN 应用是本访问控制系统针对的重点对象，因此需要编写 SDN 应用作为测试用例，以期验证系统的功能是否完整。

应用 `circuitpusher.py` 使用 python 编写，调用 `floodlight` 的 `rest api` 来实现。`circuitpusher.py` 能够在源节点和目的节点之间创建一条双向电路，在电路上下发永久的流条目，存在于两个设备节点间路由上的所有交换机中。选定主机 `h1` 作为源节点，主机 `h2` 作为目的节点，成功加载运行 `circuitpusher.py` 运行后，可以在 `s1`、`s2`、`s3` 等交换机上查找到 `circuitpusher.py` 下发的流条目。

`circuitpusher.py` 应用的实现流程如下：首先通过对 `floodlight devices` 类型的 `rest api` 调用，获得与 `h1` 和 `h3` 相连的交换机的 ID 和连接的端口信息。然后通过对 `floodlight topology` 类型的 `rest api` 调用，获得源交换机和目的交换机之间的链路，并解析链路上的各个交换机信息。最后通过对 `floodlight flow` 类型的 `rest api` 调用，对链路上的所有交换机下发流表。同理，通过 `rest api` 调用，也可以删除之前下发的流表。

由以上过程可以看出，`circuitpusher.py` 需要 `read_topology`，`flow_mod_route`，`set_flow_priority`，`flow_mod_drop` 这四种权限。

5.1 功能测试

本小节对系统进行功能测试，以验证系统功能的完整性和正确性。根据多个测试用例，对系统的各个功能进行测试，检查系统是否实现了需求分析中所描述的功能。

5.1.1 网络管理员系统登陆功能测试

1. 测试目的：验证网络管理员是否能够成功登陆应用访问控制系统。
2. 测试方案：网络管理员在 floodlight 启动后，在浏览器中通过网址 <http://localhost:8080/ui/login/index.html> 跳转到系统的登陆页面，输入默认的用户名和密码 karaf，点击登陆按钮。
3. 预测结果：网络管理员输入网址后成功跳转到系统登陆页面；输入用户名和密码后成功进入 SDN 应用访问控制系统主页。
4. 测试结果：与预期一致。

5.1.2 应用身份信息注册与查询功能测试

1. 测试目的：1) 验证网络管理员点击注册按钮后能否对 SDN 应用进行身份信息的注册；2) 验证网络管理员点击查询按钮后能否对 SDN 应用进行身份信息的查询。
2. 测试方案：1) 网络管理员在访问控制系统的身份信息管理页面注册一个新的 SDN 应用；2) 网络管理员在身份信息管理页面查询已注册的应用信息。
3. 预测结果：网络管理员能成功地输入有关信息进行应用的注册，并且能够查询到注册的应用信息，与之前输入的身份信息相同。
4. 测试结果：与预期一致。

5.1.3 应用身份信息修改和注销功能测试

1. 测试目的：1) 验证网络管理员点击修改按钮后能否成功修改应用的身份信息；2) 验证网络管理员点击注销按钮后能否删除应用的身份信息。
2. 测试方案：1) 在身份信息修改页面修改已注册的应用的身份信息；2) 在身份信息注销页面注销某个应用的身份信息。。
3. 预测结果：点击修改按钮后，在查询页面可以看到相应的信息修改正确；点击注销按钮后，在在查询页面可以看到该应用已被删除。
4. 测试结果：与预期一致。

5.1.4 权限初始化和查询功能测试

1. 测试目的：1) 验证网络管理员在权限管理页面勾选权限并点击初始化按钮后能否对应用实现权限的授予；2) 验证网络管理员在权限管理页面点击查询按钮后能否对应用权限实现查询操作。
2. 测试方案：网络管理员首先在权限初始化页面勾选权限并点击初始化按钮，然后点击权限查询按钮查看权限信息。
3. 预测结果：点击初始化按钮后提示初始化成功，点击查询按钮后显示应用的权限信息，并与初始化授予的权限相同。
4. 测试结果：与预期一致。

5.1.5 权限增加与移除功能测试

1. 测试目的：1) 验证网络管理员点击增加按钮后能否实现应用的权限增加操作；2) 验证网络管理员点击移除按钮后能否实现权限删除操作。

2. 测试方案：1) 网络管理员首先在页面上勾选新增权限，点击增加按钮，再查询应用的权限信息；2) 勾选要删除的权限，再点击移除按钮，再查询应用的权限信息。

3. 预测结果：点击增加按钮后，提示增加成功，点击查询后显示新增的权限；点击移除按钮后，提示移除成功，点击查询后显示该权限被删除。

4. 测试结果：与预期一致。

5.1.6 访问控制策略创建功能测试

1. 测试目的：验证网络管理员输入相应信息，点击策略创建按钮后能否实现访问控制策略的创建的操作。

2. 测试方案：管理员输入策略的有关信息，点击创建按钮，最后查看策略文件 fdIPolicy.xml。

3. 预测结果：点击策略创建按钮后，提示创建成功，查看策略文件 fdIPolicy.xml，显示策略已经被成功创建。

4. 测试结果：与预期一致。

5.1.7 网络视图显示功能测试

1. 测试目的：验证网路管理员点击网络试图菜单后能否查看到控制器、网络拓扑、网络节点的相关信息。

2. 测试方案：网路管理员点击网络视图菜单，查看显示信息。

3. 预测结果：网路管理员查看到网络拓扑，控制器，交换机，主机等信息，并且网络拓扑和准备工作中 mininet 创建的网络拓扑相同。

4. 测试结果：与预期一致。

5.1.8 应用身份认证功能测试

1. 在应用注册页面注册 circuitpusher.py 应用，circuitpusher.py 的身份信息如图 5-2 所示。



The screenshot displays the '应用管理系统后台' (Application Management System Backend) interface. At the top, there is a header with a user icon, the title '应用管理系统后台', and the timestamp '2017-5-18 16:47:12'. Below the header, the interface is divided into three main sections: '身份信息管理' (Identity Information Management), '权限信息管理' (Permission Information Management), and '控制策略创建' (Control Policy Creation). The '身份信息管理' section is active and contains four sub-sections: '应用信息查询' (Application Information Query), '应用信息注册' (Application Information Registration), '应用信息修改' (Application Information Modification), and '应用信息注销' (Application Information Cancellation). The '应用信息注册' sub-section is selected, showing a registration form for the application 'circuitpusher'. The form includes fields for 'AppID' (circuitpusher), 'AppName' (NetworkSwitchCheck), 'AppKey' (pcf199502), '注册商' (chinaMobile), '注册日期' (2017-3-16), '有效期' (2018-6-20), and '信用评级' (2). A '注册' (Register) button is located at the bottom of the form.

身份信息管理	权限信息管理	控制策略创建
应用信息查询	AppID: circuitpusher	
应用信息注册	AppName: NetworkSwitchCheck	
应用信息修改	AppKey: pcf199502	
应用信息注销	注册商: chinaMobile	
	注册日期: 2017-3-16	
	有效期: 2018-6-20	
	信用评级: 2	
	注册	

图 5-2 注册测试应用

2. 在 linux 终端分别使用 1) `/circuitpusher.py --appid circuitpusher --appkey pc10000 --controller=192.168.1.100:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit1` 和 2) `/circuitpusher.py --appid circuitpusher --appkey pcf199502 --controller=192.168.1.101:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit2` 两条命令创建两个 circuitpusher.py 应用访问控制器资源，在主机 h1 和 h2 之间创建双向电路。其中应用 1 以 `appid = circuitpusher&& appkey= pc10000` 发起访问；应用 2 以 `appid = circuitpusher&& appkey= pcf199502` 发起访问。

3. 由图 5-3 可以看到，应用 1 以错误的 appid 和 appkey 发起访问时，应用身份认证模块拒绝了它的访问请求，无法访问控制器资源。由图 5-4 可以看到，应用 2 以正确的 appid 和 appkey 发起访问时，应用身份认证模块通过了它的访问请求。表明系统的身份认证功能得到验证，功能正确。

```
pengcong@pengcong-Lenovo-Product:~/floodlight-0.91/apps/circuitpusher$ ./circuitpusher.py --appid circuitpusher --appkey pc10000 --controller=192.168.1.100:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit
Namespace(action='add', appid='circuitpusher', appkey='pc10000', circuitName='testCircuit', controllerRestIp='192.168.1.100:8080', dstAddress='10.0.0.3', srcAddress='10.0.0.1', type='ip')
Authentication failed

Traceback (most recent call last):
  File "./circuitpusher.py", line 100, in <module>
    parsedResult = json.loads(result)
  File "/usr/lib/python2.7/json/__init__.py", line 338, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python2.7/json/decoder.py", line 366, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/usr/lib/python2.7/json/decoder.py", line 384, in raw_decode
    raise ValueError("No JSON object could be decoded")
ValueError: No JSON object could be decoded
```

图 5-3 应用身份认证失败

```
pengcong@pengcong-Lenovo-Product:~/floodlight-0.91/apps/circuitpusher$ ./circuitpusher.py --appid circuitpusher --appkey pcf199502 --controller=192.168.1.100:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit
Namespace(action='add', appid='circuitpusher', appkey='pcf199502', circuitName='testCircuit', controllerRestIp='192.168.1.100:8080', dstAddress='10.0.0.3', srcAddress='10.0.0.1', type='ip')
Authentication success

Traceback (most recent call last):
  File "./circuitpusher.py", line 100, in <module>
    parsedResult = json.loads(result)
  File "/usr/lib/python2.7/json/__init__.py", line 338, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python2.7/json/decoder.py", line 366, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/usr/lib/python2.7/json/decoder.py", line 384, in raw_decode
    raise ValueError("No JSON object could be decoded")
ValueError: No JSON object could be decoded
```

图 5-4 应用身份认证成功

5.1.9 应用权限检查功能测试

1. 在应用注册页面注册 circuitpusher.py 应用，circuitpusher.py 的身份信息如图 5-2 所示。应用 circuitpusher.py 创建双向电路所需要的的权限有：read_topology，flow_mod_route，set_flow_priority，flow_mod_drop。

2. 在 linux 终端使用 /circuitpusher.py --appid circuitpusher --appkey pcf199502--controller=192.168.1.101:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit 命令运行应用 circuitpusher.py，发现应用没有任何权限，访问控制器资源失败，情况如图 5-5 所示。

```
pengcong@pengcong-Lenovo-Product:~/floodlight-0.91/apps/circuitpusher$ ./circuitpusher.py --appid circuitpusher --appkey pcf199502 --controller=192.168.1.100:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit
Namespace(action='add', appid='circuitpusher', appkey='pcf199502', circuitName='testCircuit', controllerRestIp='192.168.1.100:8080', dstAddress='10.0.0.3', srcAddress='10.0.0.1', type='ip')
Authentication success
No Access Permissions
Traceback (most recent call last):
  File "./circuitpusher.py", line 101, in <module>
    parsedResult = json.loads(result)
  File "/usr/lib/python2.7/json/__init__.py", line 338, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python2.7/json/decoder.py", line 366, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/usr/lib/python2.7/json/decoder.py", line 384, in raw_decode
    raise ValueError("No JSON object could be decoded")
ValueError: No JSON object could be decoded
```

图 5-5 应用无访问权限

3. 在权限管理页面给应用 circuitpusher.py 初始化 read_topology 权限后，再次运行应用，发现应用初始的几条命令运行成功，然而使用 ovs-ofctl dump-flows s2 查看交换机上的流表时，流表为空，没有被安装。如图 5-6 和 5-7 所示。

```
pengcong@pengcong-Lenovo-Product:~/floodlight-0.91/apps/circuitpusher$ ./circuitpusher.py --appid circuitpusher --appkey pcf199502 --controller=192.168.1.101:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit
Namespace(action='add', appid='circuitpusher', appkey='pcf199502', circuitName='testCircuit', controllerRestIp='192.168.1.101:8080', dstAddress='10.0.0.3', srcAddress='10.0.0.1', type='ip')
Authentication success
curl -s http://192.168.1.101:8080/wm/device/?ipv4=10.0.0.1&appid=circuitpusher\&appkey=pcf199502

curl -s http://192.168.1.101:8080/wm/device/?ipv4=10.0.0.3&appid=circuitpusher\&appkey=pcf199502

Creating circuit:
from source device at switch 00:00:00:00:00:00:00:02 port 1
to destination device at switch 00:00:00:00:00:00:00:03 port 1
curl -s http://192.168.1.101:8080/wm/topology/route/00:00:00:00:00:02/1/00:00:00:00:00:00:00:03/1/json?appid=circuitpusher\&appkey=pcf199502

[{"switch": "00:00:00:00:00:00:00:02", "port": 1}, {"switch": "00:00:00:00:00:00:00:02", "port": 3}, {"switch": "00:00:00:00:00:00:00:01", "port": 1}, {"switch": "00:00:00:00:00:00:00:01", "port": 2}, {"switch": "00:00:00:00:00:00:00:03", "port": 3}, {"switch": "00:00:00:00:00:00:00:03", "port": 1}]
```

图 5-6 加入 read_topology 权限

```

pengcong@pengcong-Lenovo-Product:~$ sudo su
[sudo] password for pengcong:
root@pengcong-Lenovo-Product:/home/pengcong# ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
root@pengcong-Lenovo-Product:/home/pengcong#

```

图 5-7 交换机流表

4. 在之前的基础上给应用增加 `flow_mod_route`, `set_flow_priority` 两个权限后, 运行应用, 查看交换机上的流表, 可以观察到流表被正确的安装, 双向电路创建成功。如图 5-8 所示。

```

pengcong@pengcong-Lenovo-Product:~$ sudo su
[sudo] password for pengcong:
root@pengcong-Lenovo-Product:/home/pengcong# ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
  cookie=0xa000001869bae5, duration=284.665s, table=0, n_packets=0, n_bytes=0, idle_age=284, hard_age=35, arp,in_port=1 actions=output:3
  cookie=0xa000001f21cf69, duration=35.352s, table=0, n_packets=0, n_bytes=0, idle_age=35, arp,in_port=3 actions=output:1
  cookie=0xa00000d49b8f6e, duration=35.364s, table=0, n_packets=0, n_bytes=0, idle_age=35, ip,in_port=3,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:1
  cookie=0xa00000d49b8f62, duration=284.675s, table=0, n_packets=0, n_bytes=0, idle_age=284, hard_age=35, ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:3
root@pengcong-Lenovo-Product:/home/pengcong#

```

图 5-8 交换机流表安装成功

5. 以上过程表明应用的权限检查功能得到验证。实现了对应用的权限的严格审查, 只有具有相应的权限的应用才能访问对应的资源。

5.1.10 XACML 访问控制功能测试

1. 在应用注册页面注册 `circuitpusher.py` 应用, `circuitpusher.py` 的身份信息如图 5-2 所示, 应用注册商为 `chinaMobile`。并给应用初始化下列权限: `read_topology`, `flow_mod_route`, `set_flow_priority`, `flow_mod_drop`。

2. 在访问控制策略创建页面创建如下的访问控制策略: 允许访问的时间段为 10:00-15:00, 允许的注册商为 `chinaMobile`。

3. 在 linux 终端, 时间为 9:00, 使用 `/circuitpusher.py -appid circuitpusher -appkey pcf199502 -controller=192.168.1.101:8080 -type ip -src 10.0.0.1 -dst 10.0.0.3 -add --name testCircuit` 命令运行应用 `circuitpusher.py`, 发现应用运行失败, 没有通过 XACML 访问控制。如图 5-9 所示。

```

pengcong@pengcong-Lenovo-Product:~/floodlight-0.91/apps/circuitpusher$ ./circuit
pusher.py --appid circuitpusher --appkey pcf199502 --controller=192.168.1.100:8
080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit
Namespace(action='add', appid='circuitpusher', appkey='pcf199502', circuitName='
testCircuit', controllerRestIp='192.168.1.100:8080', dstAddress='10.0.0.3', srcA
ddress='10.0.0.1', type='ip')
Authentication success
XACML Access Deny
Traceback (most recent call last):
  File "./circuitpusher.py", line 101, in <module>
    parsedResult = json.loads(result)
  File "/usr/lib/python2.7/json/__init__.py", line 338, in loads
    return _default_decoder.decode(s)

```

图 5-9 应用没有通过 XACML 访问控制

4. 在访问控制策略创建页面创建新的访问控制策略：允许访问的时间段为 10: 00-15: 00,允许的注册商为 chinaUniform。

5. 在 linux 终端,时间为 11: 00,使用/circuitpusher.py --appid circuitpusher - -appkey pcf199502--controller=192.168.1.101:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit 命令运行应用 circuitpusher.py,发现应用运行失败,没有通过 XACML 访问控制。情况和图 5-9 相同。

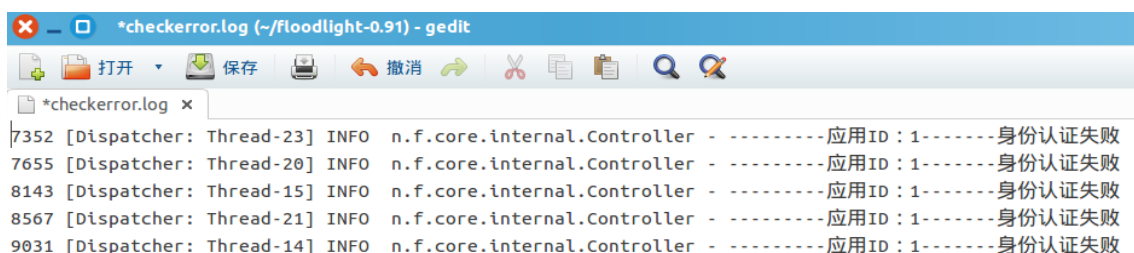
6. 在访问控制策略创建页面创建新的访问控制策略：允许访问的时间段为 10: 00-15: 00,允许的注册商为 chinaMobile。

7. 在 linux 终端,时间为 11: 00,使用/circuitpusher.py --appid circuitpusher - -appkey pcf199502 --controller=192.168.1.101:8080 --type ip --src 10.0.0.1 --dst 10.0.0.3 --add --name testCircuit 命令运行应用 circuitpusher.py,发现应用成功运行,查看交换机上的流表,可以观察到流表被正确的安装,双向电路创建成功。情况和图 5-8 相同。

8. 以上过程表明 XACML 访问控制功能得到验证。实现了对应用基于属性的访问控制,只有符合访问控制策略的访问请求才能够被允许。

5.1.11 日志记录功能测试

1. 测试目的: 验证系统能否对所有拒绝的非法访问请求进行记录。
2. 测试方案: 查看日志文件 checkerror.log。
3. 预测结果: 日志文件 checkerror.log 显示被拒绝的非法访问请求。
4. 测试结果: 与预期一致,如图 5-10 所示。



```

7352 [Dispatcher: Thread-23] INFO n.f.core.internal.Controller - -----应用ID : 1-----身份认证失败
7655 [Dispatcher: Thread-20] INFO n.f.core.internal.Controller - -----应用ID : 1-----身份认证失败
8143 [Dispatcher: Thread-15] INFO n.f.core.internal.Controller - -----应用ID : 1-----身份认证失败
8567 [Dispatcher: Thread-21] INFO n.f.core.internal.Controller - -----应用ID : 1-----身份认证失败
9031 [Dispatcher: Thread-14] INFO n.f.core.internal.Controller - -----应用ID : 1-----身份认证失败

```

图 5-10 日志记录

5.2 性能测试

系统对应用进行身份认证、权限检查、XACML 访问控制等过程会在时间上给应用访问请求带来一定的延迟，因此，访问控制系统给应用访问请求带来的时间延迟是衡量系统性能的一个重要指标。系统性能测试方案的步骤如下：首先记录 SDN 应用在有访问控制系统的环境下的执行时间，然后记录下同一个 SDN 应用在无访问控制系统的环境下的执行时间，最后将两种环境下的执行时间进行对比，得到系统的延迟时间。性能测试所使用的工具为 httping，测试重复了 8 次，测试结果如表 5-2 所示。

表 5-2 系统性能测试表

测试编号	1	2	3	4	5	6	7	8	平均
无访问控制系统的执行时间 (us)	9	12	12	9	30	33	12	18	16.875
有访问控制系统的执行时间 (us)	1536	1368	1252	1332	681	1206	633	939	1118.25
时间延迟 (us)	1527	1356	1239	1323	651	1173	621	921	1011.735

由表 5-2 可以看出，平均时间延迟是 1011.735us，在 floodlight 控制器上没有增加比较大的额外开销。因此本系统在实现系统功能的同时，在处理时间和效率上能够满足一般情况下作为访问控制系统的性能需求。

第六章 总结和展望

6.1 总结

SDN 作为一种全新的网络设计理念，与网络安全联系紧密。一方面，它的出现给传统的网络安全研究带来了很多新的思路和解决方式；另一方面，其具有的集中控制性和开放性也会产生很多新的安全问题，比如控制器、基础设施层、控制器与应用层之间以及控制器和转发设备之间的安全问题等。其中，北向接口的开放性给控制层和应用层之间的安全带来了极大的威胁。攻击者可以通过北向接口暴露出来的漏洞，直接向控制器发起攻击。恶意应用可以随意接入控制层，访问网络状态信息和操纵网络流量，破坏网络的正常状态，威胁网络安全。针对 SDN 应用可以随意接入网络，访问网络资源，给北向接口带来安全威胁的这个问题，本文设计了一种面向应用的 SDN 安全架构，实现了应用访问控制的安全功能，包括对接入网络的 SDN 应用进行身份认证，权限检查，基于属性的访问控制等，以增强北向接口的安全性。本文完成的主要工作有：

1. SDN 架构和 SDN 安全研究综述

本文第一章首先介绍了 SDN 的发展背景以及 SDN 架构的特点，SDN 存在的安全问题，包括应用层安全、北向接口安全、控制层安全、南向接口安全和数据转发层安全。然后针对这些安全问题，对目前的研究现状以及主要的解决思路进行了阐述。

2. SDN 安全架构、SDN 控制器和访问控制技术研究综述

本文第二章首先对目前已有的 SDN 安全框架进行了简单的阐述，包括 FRESKO、SDSA、PermOF 等。接着对目前 SDN 控制器的发展进行了总结，详细介绍了 NOX/POX, RYU, ONOS, Floodlight, OpenDaylight 这五种开源控制器，在进行对比后选取 Floodlight 作为本论文使用的 SDN 控制器。最后对目前的访问控制技术进行了介绍，包括 DAC, MAC, RBA-C, ABAC 等，并且对不同的访问控制技术进行了比较。

3. 面向应用的 SDN 安全架构的设计

针对北向接口的安全问题，本文提出了一种面向应用的 SDN 安全架构，实现了对应用进行访问控制的安全功能，包括对 SDN 应用进行身份认证，权限检查和基于属性的访问控制等。然后对面向应用的 SDN 安全架构中的应用访问控制系统进行了具体的框架和模块设计，将系统划分为前端视图层，逻辑控制层，数据存储层三层。接着对每一层的功能模块进行了详细的设计和介绍。最后重点描述了本访问控制系统所采用的应用访问控制决策算法。

4. SDN 应用访问控制系统的实现

根据之前的系统设计，对系统进行具体的实现。首先介绍了系统的总体实现过程，接着对系统各个功能模块的实现方式进行了详细的介绍。重点阐述了逻辑控制层主要模块的实现过程，包括 XACML 访问控制模块中访问控制决策算法的实现方式。最后通过 SDN 整体安全架构的测试，对本系统的功能进行了验证。证明了本文提出的 SDN 安全架构能够实现对 SDN 应用的访问控制。

6.2 展望

本文提出的面向应用的 SDN 架构能够对应用进行身份认证，权限检查以及基于属性的访问控制，完成了对接入网络的 SDN 应用的访问控制功能，增强了北向接口的安全性。但对 SDN 应用的访问控制是 SDN 安全研究的一个重要方向，目前的研究工作还处于初步发展阶段。随着 SDN 安全研究工作的深入，对面向应用的 SDN 安全架构的研究展望如下：

1. 在分布式的 SDN 网络上进行应用的访问控制。本安全架构是在单一的 SDN 控制器上进行应用的访问控制，然而随着 SDN 分布式网络的不断发展，如何在多控制器协同工作的分布式网络中进行应用的访问控制是未来的一个研究方向。

2. 对访问控制决策算法进行完善。本文采用的访问控制决策算法效率不高，针对应用的每次访问请求都会进行一次相应的访问控制判决。对于实时性要求较高的应用，时间开销大。下一步希望能够对算法进行优化，增强算法的有效性。

3. 北向接口的标准化。目前 SDN 北向接口安全问题的解决思路进展缓慢，很大的原因在于北向接口的未标准化。每类控制器在北向接口方面都不尽相同，差异很大，加大了北向接口的安全问题的解决难度。希望在未来，北向接口能达成一个统一的标准。

参考文献

- [1] 王蒙蒙, 刘建伟, 陈杰, 等. 软件定义网络: 安全模型、机制及研究进展[J]. 软件学报, 2016, 27(4):969-992.
- [2] Shin S, Porras P, Yegneswaran V, et al. FRESCO: Modular Composable Security Services for SoftwareDefined Networks[J]. Proceedings of Network & Distributed Security Symposium, 2013.
- [3] 刘文懋, 裘晓峰, 陈鹏程, 等. 面向 SDN 环境的软件定义安全架构[J]. 计算机科学与探索, 2015, 9(1):63-70.
- [4] Wen X, Chen Y, Hu C, et al. Towards a secure controller platform for openflow applications[C]. ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING. ACM, 2013:171-172.
- [5] 涂山山. 云计算环境中访问控制的机制和关键技术研究[D]. 北京邮电大学, 2014.
- [6] 叶春晓, 钟将, 冯永. 基于属性的访问控制策略描述语言[J]. 东南大学学报(英文版), 2008, 24(3):260-263.
- [7] 王淑玲, 李济汉, 张云勇, 房秉毅. SDN 架构及安全性研究[J]. 电信科学, 2013, (03):117-122.
- [8] Scotthayward S, Kane C, Sezer S. OperationCheckpoint: SDN Application Control[C]. IEEE, International Conference on Network Protocols. IEEE, 2014:618-623.
- [9] Bendiab M. A Novel Access Control Model for Securing Cloud API[C]. International Conference on NETWORKING and Advanced Systems. 2015.
- [10] Ferguson A D, Guha A, Liang C, et al. Participatory Networking: An API for Application Control of SDNs[J]. Computer Communication Review, 2013, 43(4):327-338.
- [11] Porras P, Cheung S, Fong M, et al. Securing the Software-Defined Network Control Layer[J]. 2015.
- [12] Shu Z, Wan J, Li D, et al. Security in Software-Defined Networking: Threats and Countermeasures[J]. Mobile Networks & Applications, 2016:1-13.
- [13] Klaedtke F, Karame G O, Bifulco R, et al. Access control for SDN controllers[C]. ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING. ACM, 2014:1325-1335.
- [14] Shin S, Song Y, Lee T, et al. Rosemary: A Robust, Secure, and High-performance Network Operating System[C]. ACM, 2014:78-89.
- [15] Yang Y, Chi Y, Li D, et al. Analysis of SDN security applications[M]. Multimedia, Communication and Computing Application. 2015.
- [16] 左青云, 陈鸣, 赵广松, 等. 基于 OpenFlow 的 SDN 技术研究[J]. 软件学报, 2013(5):1078-1097.
- [17] 王敢甫, 吴京京, 韩达. 基于 SDN 的 web 访问控制应用的实现[J]. 软件, 2016, 37(7):49-54.
- [18] 周环, 刘慧. 基于 Floodlight 的 SDN 控制器研究[J]. 计算机工程与应用, 2016, 52(24):137-147.
- [19] 周苏静. 浅析 SDN 安全需求和安全实现[C]. 中国通信学会信息通信网络技术委员会年会. 2013:113-116.

致 谢

在本篇论文的写作中，我的导师对我进行了细心的指导。本篇论文的完稿不仅凝结着我的心血，更是汇聚了我的导师，同学和各位师兄师姐的无私帮助。

首先，我要感谢我的导师孙岩老师。在开题初期，孙老师对我的选题及方向进行了细致的指导；在论文编写过程中，孙老师对我遇到的困难和疑惑进行了耐心的解答和帮助。孙老师的严谨的学术态度和高深的学术水平，让我在本次毕业论文写作中受益匪浅。感谢孙老师半年来的辛苦指导。

其次，我要感谢在论文写作中对我提供无私帮助的同学及各位师兄师姐。正是他们的智慧和耐心，才让我在遇到困惑之时努力解决了问题。

最后，我要感谢我的父母，是父母对我精神和物质上的支持，才让我完成了在北京邮电大学的本科学业，感谢他们的养育之恩。

在未来的研究生阶段，我将继续奋进努力，以此回报。