**目　　录**

[第一章　绪论 1](#_Toc453012471)

[1.1研究背景和研究意义 1](#_Toc453012472)

[1.2国内外研究和应用现状 1](#_Toc453012473)

[1.3论文研究内容 2](#_Toc453012474)

[1.4论文内容安排 2](#_Toc453012475)

[第二章　API管理和API访问控制技术研究 3](#_Toc453012476)

[2.1 API管理研究 3](#_Toc453012477)

[2.1.1 API与API管理现状 3](#_Toc453012478)

[2.1.2 API管理平台介绍 3](#_Toc453012479)

[2.1.3 本平台API管理功能性需求分析 5](#_Toc453012480)

[2.2访问控制技术研究 8](#_Toc453012481)

[2.2.1 访问控制技术 8](#_Toc453012482)

[2.2.2常见的访问控制模型 9](#_Toc453012483)

[2.3 API访问控制技术介绍 11](#_Toc453012484)

[第三章 API管理平台的设计与实现 13](#_Toc453012485)

[3.1 API管理平台的架构 13](#_Toc453012486)

[3.2 开源网关Kong 14](#_Toc453012487)

[3.2.1 Kong的工作流 14](#_Toc453012488)

[3.2.2 Kong的插件 14](#_Toc453012489)

[3.2.3 Kong的管理 15](#_Toc453012490)

[3.3 基于Django框架的API管理系统的实现 15](#_Toc453012491)

[3.3.1 Python-Kong-Client 16](#_Toc453012492)

[3.3.2 API管理系统管理员端的实现 18](#_Toc453012493)

[3.3.3 API管理系统用户端的实现 21](#_Toc453012494)

[第四章　API访问控制技术的设计与实现 24](#_Toc453012495)

[4.1 API访问控制决策算法的设计 24](#_Toc453012496)

[4.1.1 ABAC与API管理的结合 24](#_Toc453012497)

[4.1.2.DAC和MAC与API管理的结合 25](#_Toc453012498)

[4.1.3 RBAC与API管理的结合 25](#_Toc453012499)

[4.2基于ABAC的API访问控制的实现 26](#_Toc453012500)

[4.2.1 API网关访问控制决策流程 26](#_Toc453012501)

[4.2.2 Kong插件配置的实现 27](#_Toc453012502)

[第五章 系统测试 29](#_Toc453012503)

[5.1 功能测试 29](#_Toc453012504)

[5.1.1 用户注册功能测试 29](#_Toc453012505)

[5.1.2 用户发布API功能测试 29](#_Toc453012506)

[5.1.3 用户修改API功能测试 29](#_Toc453012507)

[5.1.4 用户删除API功能测试 29](#_Toc453012508)

[5.1.5 用户购买/使用API功能测试 30](#_Toc453012509)

[5.1.6 管理员管理API功能测试 30](#_Toc453012510)

[5.1.7 管理员管理用户功能测试 30](#_Toc453012511)

[5.2 性能测试 30](#_Toc453012512)

[第六章 总结与展望 34](#_Toc453012513)

[6.1 总结 34](#_Toc453012514)

[6.2 展望 34](#_Toc453012515)

[参考文献 36](#_Toc453012516)

[致　　谢 37](#_Toc453012517)

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. 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[11]和PANE控制器[12]。

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。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。

1.3 论文研究内容

本文的主要研究内容包括：

1. 学习和研究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. 设计一种基于SDN的应用访问控制系统，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，以增强北向接口的安全性。应用只有通过这三个过程的检查，才可以成功访问控制器资源。
3. 对SDN开源控制器进行调研分析，选取一种SDN控制器作为应用访问控制系统实现的基础。
4. 研究目前主流的访问控制策略和机制，选择合适的访问控制模型，设计一种合适的应用访问控制决策算法。
5. 应用访问控制系统和访问控制决策算法的实现。

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SDN技术的架构和特点，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开源框架。

第三章　SDN应用访问控制系统的设计，本章详细的介绍了应用访问控制系统的设计，包括系统整体架构，各模块设计，以及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SDN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，并根据测试结果对需求进行了验证。

第六章　总结与展望，对文章内容进行总结，对基于SDN的应用访问控制的研究前景和研究方向进行了展望

第二章　相关技术研究

2.1 SDN技术

SDN技术将控制逻辑与数据转发相分离，是一种新兴的网络体系架构，给网络架构的变革带来了新的思路。在传统的网络设备中，控制逻辑与数据转发是紧密集成的，比如交换机和路由器。SDN架构的核心思想是集中控制性和网络可编程性，使用软件控制器以可编程的方式对网络的运行状态进行操作和控制。通常，SDN架构主要分为三层：数据转发层、控制层和应用层。如图2-1所示。

**数据转发层**

**SDN交换机**

**SDN交换机**

**SDN交换机**

**控制层**

**SDN控制器**

**应用层**

**网络虚拟化**

**拓扑发现**

**流量监控**

**北向接口**

**南向接口**

**负载平衡**

图 2-1 SDN架构图

基础设施层由许多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。模式字段定义了流模式，流模式是包的头字段值的集合。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。值得注意的是，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。

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。

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。值得注意的是，目前北向接口方面还没有统一的标准协议。

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ActiveBroadba-ndNetworks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](https://www.baidu.com/link?url=bjPBwJBVBNWFxUSQ95-RrYUjJb9cgSNMZ0j6d59gAiWKJmLc7KJdXbndQBkKE9zKHKu4-xO6LSwrA-KnRnyuHWyplRFVYMxcqDgLqaoi00t5-b838_-RAi_qdhxKc7v4&wd=&eqid=b42c81b30001f1ce0000000359146b48)**。**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，OpenDayligh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[6]（Discretionary Access Control，DAC）、强制访问控制类型[7]（Mandatory Access Control，MAC）、基于角色的访问控制类型[8]（Role Based Access Control，RBAC）和基于属性的访问控制（Attribute Based Access Control，ABAC）和。

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，一张表中有着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)≥SC(o)。一般情况下DAC用来保护系统资源，MAC用来作为补强的的安全措施，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 开源框架介绍

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[24]。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2-2所示[24]，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 2-2 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器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2-3所示，Restlet框架由两个主要部分组成。首先，[Restlet API](https://restlet.com/technical-resources/restlet-framework/guide/2.3/core/overview)是一个支持REST和HTTP概念的中立API，便于处理客户端和服务器端应用程序的调用。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。

**Application**

**Restlet API**

**Restlet Engine**

**Extensions**

**Extensions**

图 2-3 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](http://www.oschina.net/p/slf4j),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 SDN访问控制系统的设计

3.1 需求分析

SDN网络应用程序和控制器之间的交互在整个北界接口(NBI)中进行。目前，各类控制器在北向接口(NBI)方面没有统一的规范，具有的安全防范措施较少。通过北向接口(NBI)暴露出来的漏洞，攻击者可以向控制器发起攻击，从而使得对来自应用层的安全威胁的防护工作变得尤为重要。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，通过对控制器上应用程序的访问权限分析，设计一种细粒度的SDN应用访问控制系统。本系统需要对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 总体设计

**3.2.1 系统总体架构设计**

依据上一小节的需求分析，提出了系统的整体框架图，如图3-1所示：

**SDN应用访问权限管理系统**

**前端页面层**

登录页面

权限管理页面

网络视图页面

应用身份信息管理页面

访问控制策略创建页面

**逻辑控制层**

信息交互模块

应用身份信息管理模块

权限管理模块

XACML访问

控制模块

身份认证模块

权限检查模块

日志记录模块

**数据存储层**

内存数据库

图3- 1 系统总体架构图

由图3-1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。

* + 1. **模块设计**
       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。

1. 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。
2. 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。
3. 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。
4. 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。
5. 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。

**3.2.2.2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访问控制模块的本层的核心模块，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 | EXPDATE | 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：Wen 等人在论文《Towards a Secure Controller Platform for Open Flow Applications》中分析的控制器上应用程序的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  addListener:ListenerDispatcher.java |
| flow\_removed\_event |
| 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3-2所示：

日志记录模块

应用程序请求访问控制器

获得应用的APPID和APPKEY

是否通过认证

N

Y

权限检查模块

图3- 2 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3-3所示：

日志记录模块

得到应用试图使用的权PER0

在数据存储层中查询该APPID对应的权限列表PERLIST

PERLIST是否含有PER0？

N

Y

XACML访问控制模块

图3- 3 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3-4所示：

从信息交互模块获得相关信息

日志记录模块

策略创建子模块

访问控制判决子模块

判决结果

为通过？

N

Y

XACML访问控制模块

图3- 4 XACML访问控制模块流程图

由图3-4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，这种算法将在下一小节进行详细描述。

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><app-id>< exception-event><denied-permission>

date和time：表示非法访问请求发生的时间。

app-id：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。

**3.2.2.3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](http://baike.baidu.com/item/Key-Value))存储[数据库](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)，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[11]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3-5所示：算法的输入为REQ和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poli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atti可能来自于REQ，也有可能来自外部环境。算法的输出Decision是布尔型，表示对于请求的判决结果。

|  |
| --- |
| 算法　基于属性的应用访问控制算法 |
| 输入：  REQ<APP, ENV>  POL = {pol1,pol2,…polm}  ATT = {att1,att2,…attm} |
| 输出:  Decision : Boolean |
| BEGIN  For all(poli ∈POL) do  IF ATT\not\rightarrowpoli THEN  Decision ← Deny  END IF  END For  Decision ← Permit  END. |

图3- 5 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Response。若评估结果Response为Permit，则允许访问，若是Deny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。

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。

入口**main()**函数

启动**FloodlightModuleLoader**

**FloodlightModuleLoader**查看配置文件，寻找需要加载的模块

关联模块与对应的服务

生成模块加载列表

**FloodlightModuleLoader**对模块进行初始化并启动

前端视图层

**SystemClientModule**

逻辑控制层

**AppInfoManageModule**

**AppPermissionManageModule**

**AppAuthenticationModule**

**AppPermissionCheckModule**

**XacmlCtrModule**

**LogManagementModule**

**InformationInteractionModule**

数据存储层

**DataInfoStorgeModule**

系统的功能模块列表

图 4- 1 模块的加载流程图

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,InformationInteractionModu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。

application-Finder()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\_flows}

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app-Key完全正确，则身份认证成功，返回认证结果。

AppPermissionCheckModule实现对应用的权限检查功能。主要方法有appPerm-issionSetFinder(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。

**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，主要方法包括creatPolicy()，creatRequest()，creatPDP()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三个步骤，首先creatPolicy()创建访问控制策略Policy并交给策略管理点PAP，然后creatRequest()把应用的访问请求转化为Request, creatPDP()创建策略决策点PDP，PDP读入Request并在PAP中查找相应的策略，最后通过getResult()得到最后的判决结果。

**写入日志**

**从信息交互模块**

**获得属性的相关信息**

**创建访问控制策略Policy**

Deny

**策略执行点PEP将应用的访问求转化为Request**

**将Policy交给策略管理点PAP**

**PEP通过上下处理器将Request转交给策略决策点PDP**

**PDP通过PAP查找相关的Policy**

**PDP生成评估结果**

**应用成功访问控制器资源**

Permit

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、Permiss-ionListResource、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。

R

HTTP客户端

SystemClientModule

REST应用

PermissionListResource

Router

AppListResource

PolicyCreatResource

REST应用

VirtualHost

HTTP服务器

Restlet组件

图 4- 3 InformationInteractionModule模块的实现框架

**4.3.2 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。

**4.3.2.1 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如图所示：

**4.3.2.2 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()，$("#registityButton").click(function(),$("#update-Button").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—是实现后的界面图：

**4.3.2.2 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())， $("#addButton").click(function()和$("#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。$("#addButton").click(func-tion())和$("#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—是实现后的界面图：

**4.3.2.3 访问控制策略创建页面policyCreatePage**

policyCreatePage把允许访问的时间段[time-quantum](https://www.baidu.com/link?url=G6Rst-BmrDbZAe-BzxhW1HjhRG8ZCd5GrwnSQRlbXUdEWjK89jmWdl_w5mB7ZgjMp1k2qH59mwVyH4OwTHE39n1hvCbVMJhzupPqdE8Dkn0g9qltxuFwiBkQ9ohqfL91&wd=&eqid=e13b35fb00017cfa000000035912be48)和允许的应用注册商registrary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所示：

4.3.2.4 网络视图页面networkPage

networkPage通过对floodlight的RestApi调用，得到有关控制器，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所示：

**4.3.3 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Da-taInfoStorgeModule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, AppIdentit-yCertificate 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。

系统测试环境如下：

操作系统：ubuntu Kylin 14.04 Lts

浏览器：Google Chrome 51.0.2704.106m

JAVA环境：jdk-1.7.0\_91

IDE：eclipse neon.2

SDN控制器：floodlight-0.91V

Mininet版本：2.1.0

测试前的准备工作：

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192.168.1.100，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=192.168.1.100,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。

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\_topolog，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fdlPolicy.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fdlPolicy.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按钮后能否实现网络拓扑、相关网络及节点的的查看操作。
2. 测试方案：网路管理员点击网络视图按钮，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5-1所示。
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=192.168.1.100:8080 –typr 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5-2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5-3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，并且功能正确。

**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5-1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=192.168.1.100:8080 –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5-4所示。
3. 在权限管理页面给应用circuitpusher.py初始化read\_topolog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5-5和5-6所示。
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5-7所示。
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5-1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=192.168.1.100:8080 –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5-8所示。
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=192.168.1.100:8080 –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5-8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=192.168.1.100:8080 –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5-7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。

5.2 性能测试

Dd