**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研究背景和研究意义**

随

## **1.2国内外研究现状**

值。

## **1.3论文研究内容**

本

## **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，本章首先介绍本文的研究背景和研究意义，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　API管理和API访问控制技术研究，本章首先介绍了目前国内外使用API管理的现状和主流方法，接着介绍了主流控制访问技术，最后对API访问控制技术的背景和研究现状进行了介绍。

第三章　API管理平台的设计与实现，本章详细的介绍了设计的API管理平台的基本架构、功能模块设计以及具体的程序实现。

第四章　API访问控制技术的设计与实现，本章根据第二章中介绍的常见的访问控制决策算法，结合实践中可能遇到的需要情形，设计了三种访问控制决策算法。本章的后半部分介绍了其中的一种访问控制决策算法的在API管理平台中的实现。

第五章　系统测试，本章模拟实际情况对API管理系统进行了功能测试和性能测试，对API网关进行了性能测试，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，展望了API管理和API访问控制技术的研究的前景和研究方向。

第二章　相关技术综述

## **2.1 SDN技术**

伴

## **2.2 SDN控制器调研分析与选型**

**2.2.1 访问控制技术**

访问控制是信息安全研究的一个重点，不论是传统的IT领域还是新兴的互联网领域，访问控制在其中都扮演着重要的角色，对于保证服务器软硬件、资源和用户数据的安全性起着举足轻重的作用。

访问控制是指，针对服务器端客体资源获取能力和范围进行控制的机制，当资源受到未经授权的访问时，可以依据设定好的安全策略和算法对访问进行认证和限制，以提供合理的手段对资源进行保护。可以说，访问控制存在的主要目的和意义就是保证资源的安全性，对访问进行区分和隔离，以保证正常用户的访问，而阻止非正常用户的访问。访问控制对于一个完整系统而言，是不可或缺的，对于系统安全至关重要。

访问控制包括三个要素，即主体、客体和访问控制策略，其中第三个要素，即访问控制策略是访问控制的关键技内容。如图2-4是ISO访问控制通用框架：



图2-4 ISO访问控制通用框架

主体(Subject)是指一个提出请求或要求的实体，是动作的发起者，但不一定是动作的执行者，可以是用户，也可以是任何主动发出访问请求的智能体，包括程序、进程服务等。

客体(Object)是需要接受其他主体访问的被动实体，包括所有受访问控制机制所 保护下的系统资源，在不同应用场景下可以有着不同的具体定义。比如在操作系统中 可以是一段内存空间，磁盘上面的某个文件，在数据库里可以是一个表中的某些记录，在Web上可以是一个特定的页面，网络结构中的某个广义上的数据包结构。

访问控制策略(Access Control Policy)是主体对客体的操作行为的约束条件集。简单的讲，访问控制策略是主体对客体的访问规则集，它直接定义了主体对客体可以实施的具体的作用行为和客体对主体的访问行为所做的条件约束。访问控制策略在某种程度上体现了一种授权行为，也就是客体对主体访问时，所具有的操作权限的允许。主体进行访问动作的方式取决于客体的类型，一般是对客体的一种操作，比如请求内存空间，文件的操作问题，修改数据库表中记录，以及浏览陌生服务器中的某些页面等。

在API访问控制问题中，客体是目标API，主体是意欲使用目标API的用户，对于用户发送的请求，我们应该合理的设置访问决策，根据之前设定的访问控制策略和规则库对用户的请求进行判定，从而达到对API进行访问控制的作用。

**2.2.2常见的访问控制模型**

国内外研究者己经在数据与服务的访问控制领域进行了大量研究，也产生了很多的访问控制相关理论和模型，如Discretionary Access Control(DAC)自主访问控制、Mandatory Access Control(MAC)强制访问控制、Role Based Access Control(RBAC)基于角色的访问控制、Attribute Based Access Control(ABAC)基于属性的访问控制等，并在应用、管理平台等方面得到了大量的应用，在这些系统中发挥着重要的作用，为系统安全做出了重要的贡献[15]。

1、自主访问控制

Discretionary Access Control（DAC，自主访问控制），其基本思想是系统中的主体（用户或者资源的所有者）可以根据自己的意愿，将其拥有的对客体（资源或者服务）的权限授予给其它主体。自主访问控制是一种有效的访问控制机制，能有效保护客体不被非法或者越权访问。

虽然自主访问控制是一种有效的访问控制机制，也已经在不少的系统中实现了，但是该机制也有自己的弱点。首先，自主访问控制的客体访问权是掌握在客体的所有者手中的，这样使得访问权非常的分散，不利于集中管理。其次，访问权所有者可能不是专业人员，对访问控制不了解，因而不能有效的对访问权进行管理。最后，也是最重要的一点，自主访问控制中，访问权可以传递，即将自己的访问权授予给以后，同样可以将从那里获得的访问权授予给其他人。若访问权被传递出去，管理就会非常困难，并且会带来严重的安全问题。

2、强制访问控制

Mandatory Access Control（MAC，强制访问控制），其基本思想是系统中的主体和客体都会被管理员事先分配一个固定的安全属性，而该属性是不能被除了管理员以外的人修改的，根据安全属性判断主体是否有权访问客体。文献[12]中提到，强制访问控制最先使用于军事应用中，并且常与自主访问控制相结合使用。

强制访问控制相比于自主访问控制，添加了对客体属性的划分，并通过访问之前对访问权限和安全属性进行匹配，以判断是否允许访问。某种程度上讲，强制访问控制拥有更高的安全性，可以防止非法访问。但是这种访问控制过于严格，过于复杂。由于每个主体对每个客体的安全属性都会有一个匹配值，会产生N\*N的一个对应关系，造成管理上的不便，影响系统的灵活性。

3、基于角色的访问控制

Role Based Access Control（RBAC，基于角色的访问控制），其基本思想是，建立一个角色集合，并赋予角色相应权限，在用户与资源之间加入了角色层，通过为用户分配角色，使得用户拥有角色相关的权限。该机制使得新建用户时，只需分配相应角色，而不需要赋予多种访问权限，减少了不必要的操作。并且，仅修改角色就可以有效降低管理成本和开销，有利于简化授权管理[17]。

基于角色的访问控制是一种普遍适用、更为有效的访问控制方式，也是访问控制领域的研究热点，其基于访问者身份进行授权，既符合常理，又方便快捷。文献[13]在RBAC的基础上，加入了属性，使得能够较好的进行更细致的访问控制，但将其仍有一些不足之处，不能支持开放环境的访问控制。例如：若非法访问者，获取了合法的身份信息，便可获得对应的权限，造成系统信息泄漏；若非法访问者获取了某个角色，就可以随意使用相关权限，使得权限被滥用，对系统造成损害，这种方式也违背了最小权限原则。总之，RBAC不太适合用于开放的环境，也难以做到细粒度的访问控制。

4、基于属性的访问控制

Attribute Based Access Control（ABAC，基于属性的访问控制），其主体思想是利用主体、客体以及所处环境等相关的属性来进行访问控制和授权，通过属性来对服务访问进行限制。正是由于不是直接利用主客体之间的关系，使其能适应开放的网络环境。与传统的访问控制不同，基于属性的访问控制是基于请求者和资源的属性来进行授权，因而其具备令人印象深刻的灵活性、可扩展性的特点。能够有效支持匿名访问。文献[14]提出了提升ABAC安全和性能的方法，使得的效率和安全性有了进一步的提升。目前，对于使用ABAC对API进行访问控制已经有较多的文献和实现[11]，第四章中将对API管理中使用ABAC进行访问控制具体介绍。当然，ABAC也有其缺点，因其没有细粒度的划分数据，难以对数据进行动态、细粒度的访问控制。

5、访问控制模型总结

表2-1 常见访问控制模型的优缺点对比

|  |  |  |
| --- | --- | --- |
| 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 |

表2-1针对上述的访问控制模型的优缺点进行总结，以期选择有价值的、合适的和易于实现的API访问模型，为API的访问控制技术的设计和实现打下基础。

## **2.3 访问控制技术**

目前，对于API访问控制技术，各方尚处于摸索阶段，行业内各厂商拥有和实现的方式也不尽相同，总结而言，主要的API访问控制技术有以下：

1. 身份认证

对于每个使用API的用户，验证其是否具有使用API的权限具有十分重要的意义。目前主要使用的身份认证的方法有：

* 基础HTTP认证

其方法就是使在请求API时将用户注册时的用户名和密码放入URL的参数或者Header中作为其身份验证，其优点是验证简单方便。但是其也有非常明显的缺点，因为HTTP协议采用明文传输，所有的信息都会暴露在外面，所以直接将用户名和密码在网络中传输十分的危险。为了保证安全，最好能够使用HTTPS协议来代替HTTP协议。目前只使用基础验证作为访问API时身份认证的已经非常少了。

* Token/Key认证

其方法就是为每个用户名单独生成一个Token/Key，每次访问API时需要将Token/Key带入URL参数中或者Header中，相比基础HTTP认证，因为Token/Key为独立生成，因此可以大大提高安全性。这种方法仍然存在一定的问题，由于HTTP协议明文传输，所以一旦别人获取到这个Token/Key，就可以伪装身份使用该API。目前这种认证方式使用较多，但是对于安全要求更高的API，应该考虑使用更为安全认证方式如oAuth认证或者多种认证方式结合使用[2]。

* oAuth认证

oAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是oAuth的授权不会使第三方触及到用户的帐号信息（如用户名与密码）[3]。这种方式最为安全，但也会增加延迟时间和网络资源消耗。

1. 用户权限管理

用户权限管理是企业使用API管理时非常重视和关注的地方，强大的用户权限设置功能可以满足企业用户管理API时遇到的各种问题，下面将介绍几种云服务商提供的用户权限管理的方式：

* RAM (Resource Access Management)资源访问管理

RAM是阿里云使用的用户权限管理方式。通过RAM，管理员可以集中管理使用的用户（比如员工、系统或应用程序），以及控制用户可以访问您名下哪些资源的权限。从权限角度看，管理员与RAM用户是一种root与user的关系，管理员对资源拥有一切操作控制权限，而RAM中的用户只能拥有管理员授予管理的权限。

* IAM（Identity and Access Management）。

作为云服务商的标杆，AWS使用IAM使用作为其用户权限管理的方式。IAM具有非常大的灵活性，不仅可以为用户定制群组和角色，还可以手动为每个群组、角色和用户设定访问控制策略。

* ACL(Access Control List)访问控制列表

与路由器中的ACL功能相似，在一些平台中，管理员用户可以通过配置ACL中的白名单和黑名单从而达到对用户限制使用和访问的功能。目前如APIcloud等使用了该方法。

1. 其他功能

除了身份验证和用户权限管理两项基本的访问API控制功能，还有一些平台为用户提供了更多的API访问控制功能，如IP访问限制、速率限制等。

## **2.4 web开发技术**

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。

1. 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。

1. 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。

1. 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。

1. 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。

1. 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。

1. 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 开发框架介绍

**4.1 开发框架介绍**

4.2 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

4.3 主要功能模块实现

**4.3.1 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。

4.3.1.1 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。

4.3.1.2 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。

4.3.1.3 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。

4.3.1.4 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

4.3.1.5 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。

4.3.1.6 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() $("#registityButton").click(function(),$("#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—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。$("#addButton").click(function()和$("#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—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