基于Floodlight的SDN控制器研究

周 环[1,2],刘 慧[1]
ZHOU Huan[1,2], LIU Hui[1]

1.中国科学院大学 计算机与控制工程学院,北京 100190
2.中国科学院 信息工程研究所,北京 100093
1.School of Computer and Control Engineering, University of Chinese Academy of Sciences(UCAS), Beijing 100190, China
2.Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

ZHOU Huan, LIU Hui. Research on SDN controller based on Floodlight. Computer Engineering and Applications, 2016, 52(24):137-147.

Abstract: The network architecture has been used for nearly 30 years. In order to make the whole network run smoothly, switches/routers have to run within more than 6000 protocols, which means if one network point increases a protocol the other points also have to make changes. SDN(Software Defined Network) makes the network programmable, so that it enables the network to be more flexible to fit needs of network users. In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. As a result, enterprises and carriers gain unprecedented programmability, automation, and network control, enabling them to build highly scalable flexible networks that readily adapt to changing business needs. SDN controller plays a very important role in SDN because not only does it communicate with the underlying devices but also provides APIs for the application layer. This paper analyzes the background and principle of SDN as well as its development at first; then it does some researches on an open source project——Floodlight; at last this paper summarizes and analyses the features of SDN controller.

Key words: Software Defined Network(SDN); SDN controller; network architecture

摘 要: 目前正在使用的网络架构已有30年的历史。在此架构下,交换机/路由器需要在超过6 000个分布式协议中使整个网络正常运行。这意味着只要有一个网元增加一种新的协议,其他网元都必须在结构上做出变更。SDN(Software Defined Network,软件定义网络)则打破了这种桎梏,它使得网络可编程,从而让网络在满足用户需求方面更具灵活性。SDN架构将控制和转发解耦,将控制功能集中到逻辑独立的控制环境之中,同时为应用层提供底层网络的抽象视图。结果就是SDN可以为用户提供可编程性极强的网络、网络自动化管理以及网络控制等功能,从而满足日益变化与丰富的网络需求。SDN控制器在SDN架构中的作用至关重要,它既要与基础设施层交互也需要与应用层经由API交互。首先分析了SDN架构的产生背景、原理和其发展现状;随后研究并分析了一个SDN控制器的开源项目Floodlight;最后通过对当前7种控制器的实验以及SDN相关原理对SDN控制器的特性进行了总结与分析。

关键词: 软件定义网络;SDN控制器;网络架构
文献标志码: A **中图分类号:** TP393 doi:10.3778/j.issn.1002-8331.1603-0330

1 引言

与传统网络架构相比较而言,SDN[1]是一种革命性的变革。在传统架构中,数据面的报文转发和控制面的路由决定是整合在一起的。随着移动互联网、云计算等一些新技术的发展,加速了数据中心的变革,而日益丰富的网络业务、攀升的网络带宽需求以及服务交付的个

基金项目: 国家自然科学基金-青年科学基金项目(No.61402432);中国科学院大学校长基金项目。
作者简介: 周环(1992—),男,硕士研究生,研究领域为网络安全;刘慧(1980—),女,博士,讲师,研究领域为计算机网络,网络安全,E-mail:hliu@ucas.ac.cn。
收稿日期: 2016-03-24 **修回日期:** 2016-06-16 **文章编号:** 1002-8331(2016)24-0137-11
CNKI**网络优先出版:** 2016-08-10, http://www.cnki.net/kcms/detail/11.2127.TP.20160810.1057.064.html

性化需要等都给新一代数据中心提出了更严格的要求。所以传统的紧耦合大型主机的局限性也日益凸显，通过增加RFC数量的方式修补网络也造成了交换机或者路由器设备控制功能的极度复杂。为了适应日后互联网日益丰富的业务需求，业内形成了“现在是创新思考互联网体系结构，应该采用新的设计理念的时候”的主流意见[2]。同时也对未来网络体系架构所应该具备的性质和功能提出了要求。

SDN很好地满足了新的需求，它让网络可编程化从而使研究人员在网络中基于真实生产流量进行大规模实验成为可能。同时SDN提供开放的API，进而可实现对网络设备的自动化配置和对网络流量的实时监控。

软件定义网络这种思想是由Stanford University在研究Clean State[3]项目时提出的，设计的目的是在研究如何提升网络可靠性、能效、速度和安全性问题时能利用网络进行大规模真实流量和丰富应用的实验。SDN技术也于2009年入选了美国MIT主办的《技术评论》杂志十大新型技术[4]。尽管SDN定义了一种新的网络架构，属于下一代网络新技术研究课题，但是SDN并不革新原有的IP分层网络的报文转发行为，它只是简化了报文转发规则产生的复杂性[5]。

SDN的核心是将控制功能从网络中分离出来，将其转移到逻辑独立的控制环境——SDN控制器中。SDN控制器可以在通用的服务器上运行，使用者可以进行控制功能编程。也正是因为此，控制功能不再局限于路由器或者交换机之中，也不再只能由设备生产厂商进行编程和定义。

Floodlight是一个开源的SDN控制器项目，它是用JAVA开发的基于apache协议的SDN控制器。本文主要将通过对此项目的分析对SDN控制器进行研究，同时利用cbench和hcprobe对当前7种SDN控制器的性能、可靠性、可扩展性等进行实验，最后通过以上分析和实验对控制器特性进行了总结和分析，以期对SDN控制器做出有益的探索。

2 SDN研究现状及技术原理

2.1 SDN/OpenFlow的发展历程

美国GENI项目资助的斯坦福大学Clean Slate项目组于2006年首次提出了SDN思想，以斯坦福大学Nick Mckeown教授为首的研究团队在校园网络的试验创新中提出OpenFlow[6]的概念，正是基于OpenFlow使网络具备可编程的特性SDN的概念应运而生。Martin Casado和他的团队成员于2007年提出了Ethane架构[7]，作为Clean Slate项目的子项，Ethane架构试图通过一个集中式的控制器，使研究人员方便地定义基于网络流的控制策略，并把这些安全策略运用到网络设备中，最终灵活地实现对整个网络的安全控制。

在Ethane和其前续项目Sane[8]的启发下，Nick Mckeown教授等人于2008年正式提出OpenFlow的概念，并于当年发表了题为《OpenFlow：Enabling Innovation in Campus Networks》的论文，首次详述了OpenFlow的工作原理和其几大应用场景。基于OpenFlow的可编程特性，Nick Mckeown教授以及他的团队成员便进一步提出了SDN（Software Defined Network）这一概念。而SDN概念也于2009年入围了Technology Review年度十大前沿技术，并自此取得了学术界和工业界的认可和支持。OpenFlow规范目前已经经历了1.1、1.2以及1.3等版本[9]。谷歌和微软也已经将SDN架构应用到其数据中心[10-11]。

2.2 SDN体系架构与特性

SDN架构将控制和转发相分离，并且可直接编程。它的核心思想是将传统的紧耦合网络架构解耦为转发、控制、应用分离的体系架构从而形成数据平面层、控制平面层和应用层三层，如图1所示。

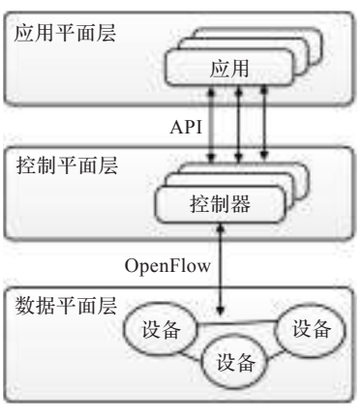

图1 SDN技术架构

在这一架构下，控制平面层中的控制器与数据平面层由控制数据面接口（南向接口）交互，而控制平面层与应用层之间则经由控制器提供的API（北向接口）交互；数据平面中的网络基础设施充当原交换机/路由器设计中的转发者角色，这也被称为OpenFlow交换。

南向接口可以屏蔽网络基础设施资源在支持的协议、种类等方面存在的异构性，从而使得数据平面层的网络基础设施能无障碍地接收来自控制器的转发策略，承担起网络中的转发业务；北向接口为上层的应用层提供开放的API，使用户可通过软件随时直接地进行控制功能的编程。

软件定义网络是一种动态的、可管理的、性价比高并且适应能力极强的新兴网络架构。它非常适合如今高带宽、动态性更强的应用程序。这种架构解耦网络控制和转发功能使网络变得更直接和可编程，并且使底层基础设备对应用程序和网络服务抽象。

3　Floodlight相关分析与研究

3.1　Floodlight项目介绍

Floodlight是一个开源的、企业级的、apache许可的、基于Java的OpenFlow控制器，由开发人员社区支持维护[12]。Floodlight被设计出来的目的旨在完成对数量日益庞大交换机、路由器、虚拟交换机和支持标准OpenFlow协议的接入点的灵活控制。而其开源的特性也使得其质量更可靠，更具透明性。Floodlight不仅仅是一个SDN控制器，它也包含一系列模块化应用，而这些应用可以向上提供REST API，从而帮助应用层的应用更好地管控整个网络。由于Floodlight是用Java开发的，基于Java跨平台的特性，Floodlight也可以运行在多种操作系统之上，其最主要的运行环境是Ubuntu和Mac OS X。而Floodlight在整个SDN架构中的位置如图2所示。

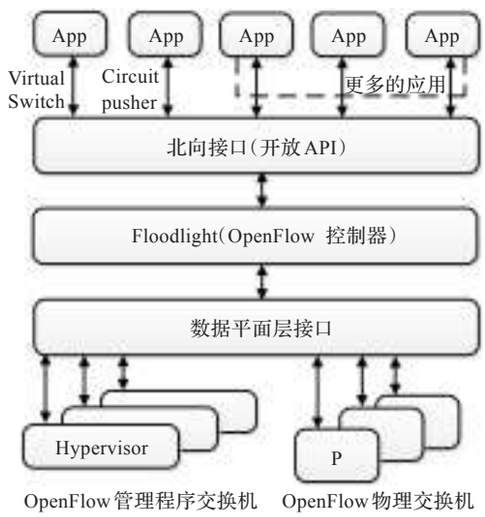

图2　Floodlight在整个SDN架构中的位置

3.2　Floodlight项目分析与研究

Mininet是一个轻量级的SDN网络研发、仿真以及测试平台。可以用它来虚拟OpenFlow交换机以及主机节点，控制器可以使用已经集成了的NOX，同时也可以使用其他支持OpenFlow协议的外部控制器。此外，BigSwitch公司还提供了集成Floodlight控制器的iso文件。

Mininet使用CLI的方式创建以及配置底层网络，只需要几行命令行就可以完成网络的创建以及对网络的测试。Mininet也提供Python API，可以方便地自定义拓扑。

本部分基于Floodlight+Mininet实验环境，重点对Floodlight控制器的架构、开发效率/语言、模块化实现、OpenFlow消息的读写算法以及底层的拓扑发现机制进行了详细深入的分析和研究。

(1)Floodlight特性

Floodlight提供了一个模块加载系统，使开发者通过IOFMessageListenner和IFloodlightModule这两个接口可以方便地扩展和增加其功能。它在很弱的依赖关系下就能完成设置。不仅如此，Floodlight支持多种虚拟和物理OpenFlow交换机，比如Open vSwitch(OVS)、Arista7050、Juniper(MX，EX)等。Floodlight可以处理混合在一起的OpenFlow和非OpenFlow网络，它能实现对多种OpenFlow交换设备组成的网络的管理。同时，FloodLight是BigSwitch[13]项目中控制器的核心，所以足以说明它在性能方面的优异性。

(2)Floodlight架构研究

Floodlight不仅仅只是一个SDN控制器，它是SDN控制器和一系列模块化的Floodlight的应用的集合。它通过实现一系列常用功能来控制和查询一个OpenFlow网络，与此同时在其之上的应用程序通过实现不同的功能特性来解决用户对于网络不同的需求，用户通过这些应用可以完成对整个OpenFlow网络的抽象化和虚拟化，获取网络的拓扑结构，并且可以通过这些应用程序完成对网络流量的管理和控制以及完成对网络QoS相关参数的配置。Floodlight控制器、以Java模块形式构建的应用和在Floodlight向上提供的REST API基础上建立的应用程序的关系图如图3所示。

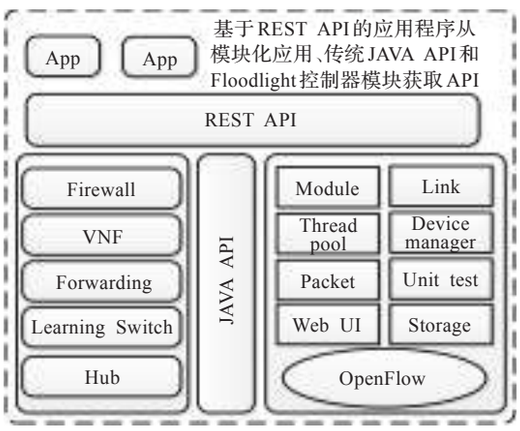

图3　Floodlight项目架构

(3)编程语言分析

在Floodlight之前，一些SDN控制器采用的编程语言是C和C++，虽然它们可以用来生产性能优异的应用程序，但使用这些语言也显著地增加了开发人员的负担。一些常见的问题包括：大段时间的完整编译(10 min)、混淆产生编译错误的真正原因、手动内存管理造成的错误、内存溢出等等[14]。当然，开发人员也采用了一些方法力求解决这些问题：比如采用外围组件增量编译以减少编译时间，严格控制使用诸如智能指针类的技术以改善内存错误，但是即便如此，这些方法仍是不完善的，因为这些方法的使用仅仅局限在少数开发人员之中。

选择C/C++在一些环境下是正确的，但是针对一个在商品化硬件上运行的SDN控制器来说，能否有一种语言能轻易地实现CPU和内存的延展也成了一个值得探讨的问题。基于以上思考，要满足要求，所采用的编

程语言必须满足3点特性:内存自动管理、跨平台以及性能优异。

内存自动管理(也称垃圾回收)可以消除大多数编程相关的问题。拥有这一特点的编程语言的编译时间通常很短或者基本可以忽略不计,从而消除了因为等待程序编译而浪费的时间。这类语言同时也提供错误报告并指出具体在哪一行发生编译/运行错误,而满足以上要求的语言有:C#、JAVA和Python等。如表1所示,有三种语言的特性比较。

表1 三种程序语言特性比较

语言	内存自动管理	跨平台	性能优异
C#	√	×	√
JAVA	√	√	√
Python	√	√	×

Floodlight理想的运行平台是Linux,但是如果能在没有显著移植工作量的情况下使其能够运行在Mac OS X和Windows上对于开发者和使用者来说都是极为方便的。若是在跨平台移植中涉及到太多的工作量,则会阻碍最初的控制器移植到一些非Linux系统之上。

C#、JAVA和Python都具备一些跨平台运行的能力。然而官方对C#提供的解释器——CLR(Common Language Runtime,通用语言运行平台)却缺少对非Windows之外的操作系统的支持,所以Floodlight并未采用C#这种语言。

高性能是一个很主观的术语,在这种情况下,它的衡量标准之一就是对处理器内核的扩展能力,在官方的解释器中缺乏真正的多线程也使Python无法成为Floodlight的编程语言。

而JAVA则可以很好地满足以上所说的3个条件:首先由于JVM的存在,JAVA几乎可以跨越大多数操作系统平台使用,而不需要太多的移植操作;同时JAVA很好地支持了内存动态管理,在编程中没有指针这一概念,所以很好地杜绝了很多编程中可能出现的错误;最后,完美地支持多线程也使JAVA在性能上拥有很优异的表现,而用JAVA编写的诸如Hadoop和Tomcat都展现出了很高的性能。所以,最终Floodlight成为了一个基于JAVA的SDN控制器。

(4)Floodlight启动过程

Floodlight采用模块加载系统以决定运行哪些模块,采用此系统的目标是:通过修改配置文件来决定哪些模块会被加载;实现一个模块但不需要修改它所依赖的那些模块;构建一个定义良好的平台和开放API以扩展Floodlight;实现对代码的模块化。

①主要部件

模块加载系统的主要部件有:模块、模块加载器、服务、配置文件以及一个在Jar文件中包含了可用模块列表的文件。

模块被定义为一个实现了IFloodlightModule接口的类,而一个模块可以包含一个或者多个服务,服务则被定义为一个继承了IFloodlightService接口的接口。配置文件中规定了哪些模块可以被加载,并采用键值对的形式来表示。在模块列表中键是floodlight.modules,而对应的值则是以逗号分隔的模块列表,下面是一个Floodlight默认的配置文件:

```
Floodlight.modules=net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\
net.floodlightcontroller.forwarding.Forwarding,\
net.floodlightcontroller.jython.JythonDebugInterface
```

②模块启动步骤

模块发现:

所有在类路径里的模块,也就是实现了IFloodlightModule的类,均会被发现并建立三种映射。第一种是服务映射,它会建立服务和提供该服务的模块之间的映射关系;第二种是模块-服务映射,它会建立模块和它所提供的所有服务之间的映射关系;最后一种是模块-名称映射,也就是建立模块类和模块类名称之间的映射关系。

寻找最小模块集合:

使用深度优先遍历算法(Depth First Search,DFS)找出所需要加载模块的最小集合,所有在配置文件中定义的模块均会被添加到队列之中,而出队的模块将会被添加到启动模块列表之中。当一个模块被添加到启动模块列表之中后,如果该模块的依赖模块还未被添加到启动列表,就会在该模块上调用getModuleDependencies()方法。

会有两种情况引起getModuleDependencies异常,第一种是找不到配置文件之中定义的模块或模块依赖,另一种是两个模块都提供某种服务,但却没有指定具体使用哪一个。

初始化模块:

用迭代的方式加载模块集合,每个模块都会调用init方法,而模块需要做的事情则有两个。其一是调用getServiceImpl()方法将它的依赖关系写到一起;另一个就是初始化自己内部的数据结构。

启动模块:

每个模块上调用init()方法后会接着调用startUp()方法,这个方法中模块会调用它所依赖的其他模块,并最终完成所需模块的启动。

(5)OpenFlow消息的读取和写入分析

SDN控制器的性能通常由两个方面衡量:每秒内控制器能处理和响应的PacketIn事件数量;控制器对每个事件的平均处理时间。

在IFloodlightProvider服务下注册的应用通过实现IOFMessageListener接口接受来自OpenFLow交换机特定种类的消息。注册的监听者对每个种类的OpenFlow

消息形成一条串行处理管道，管道内部监听者的顺序是可以配置的，并且监听者可以选择是否继续传播相应消息。接下来将从OpenFlow消息的读取和写入两方面对SDN控制器Floodlight做出相应分析。

①读OpenFlow消息

为了在性能方面有优异的表现，Floodlight和基于它的应用都是多线程的，而针对多线程这一特性，本文也提出了两种处理策略。

共享队列：

图4显示了共享队列设计，其中包含两组线程集。第一组线程集是I/O线程集，它从交换机中读取并反序列化OpenFlow消息。然后将读取到的消息放入共享队列之中。每个交换机被分配给一个I/O线程，并且多个交换机可能被分配给同一个I/O线程，当然，I/O线程也能将消息传送到相应的交换机之中。第二组线程则是管道线程集，通过IOFMessageListener处理从共享队列中出队的消息。这对于管道线程来说是效率很高的一种方式，每当产生出队消息时，管道相应线程不会处于空闲等待状态。

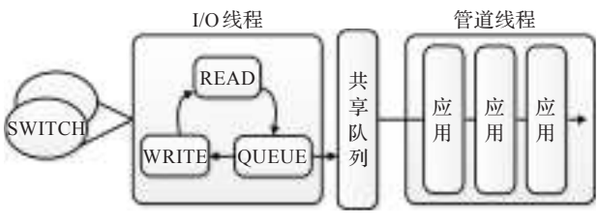

图4　共享队列形式

然而这种设计则需要共享队列锁，所以两组线程集之间会产生对锁的争用。当然这种设计也可以做出相应优化，如为每个交换机设计一个或者多个线程，每个线程对对应不同的优先级。这种优化可以通过队列循环服务来提升交换机之间消息处理的公平性。

运行直到完成：

图5显示了运行直到完成的设计，一个相对于共享队列更为简化的版本，它只有一个单一的I/O线程池。其中每一个线程类似于在共享队列设计中的I/O线程，然而与管道线程等待处理出队消息不同，这种设计能够直接处理出队消息。它在读取路径上并不需要锁，因为反序列化消息的进程同时也会通过IOFMessageListener处理消息。这种设计也进一步对IOFMessageListener提出了要求：对于每个交换机来说，一次只有一个线程处理该交换机的消息，也就是说每个线程一次只对应一个交换机。这种设计克服了繁忙交换机可能会被静态分配给线程集子集而因此造成空闲线程等待的设计缺陷。

Floodlight使用的是读OpenFlow设计是运行到完成的设计，并配以可配置数目的I/O线程。OpenFlow交换设备以一个循环方式分配到I/O线程，并且保持静态分配状态直至它们断开。

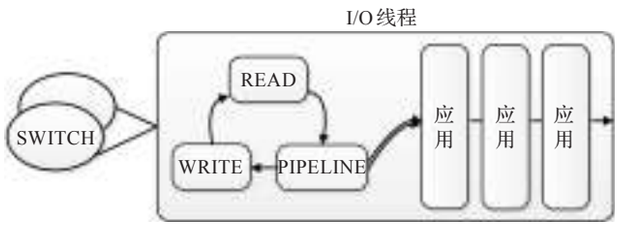

图5　运行直到完成形式

②写OpenFlow消息

消息写到交换机的方法同样影响到控制器的性能。Floodlight是多线程的写操作可能同时来自多个线程，并且为了防止竞争的产生写方法必须同步。算法1的设计方案是直接写，采用此方法来将消息发送到OpenFlow交换机。应用程序调用写方法，它可以序列化消息并将其添加到交换机特定的缓冲区之中，然后将其再写入socket。这种设计的速度很慢，因为没有中间批处理的情况下，JAVA虚拟机会对每个socket写操作发出一个对应的系统内核写操作。在繁忙系统中，用户态和内核态的转变会花费很长的时间。

算法1

```
1: function WRITE(OFMessagemsg)
2:     buf.append(msg)
3:     flush()
4: end function
5: function FLUSH
6:     socket.write(buf)
7: end function
```

一个修改后的批处理方案解决了这个问题，这种设计的第一部分在算法2中显示，一个修改后的flush方法包含两个布尔值：written和readSelect。当写socket发送时，写标志被设置为true，以防止后续的写操作直到标志被设置为false。当在socket写之后仍有未被写入的数据时，needSelect标志被设定，以标明传出TCP缓冲的状态是满的，并且如果缓冲区空间可用时会将数据调入缓冲区。

算法2

```
1: function FLUSH
2: if !written && !needSelect then
3:         socket.write(buf)
4:         written=true
5:     if buf.remaining()>0 then
6:         needSelect=true
7:     end if
8: end if
9: end function
```

算法3显示了这种设计的第二部分，它涉及到I/O环路的修改。第3到9行确保了每个交换机会在传出缓冲不满时对传出数据进行写操作，第15到18行则确保只有在有可用传出缓冲区检测使用系统调用选择函数

时写操作才会发生。对于负载较大的系统自然批处理发生在I/O环之间,以减少写调用和用户内核开销。

算法 3

```
Algorithm 3 Revised I/O Loop
1: function IOLOOP
2: while true do
3:    for all Switch sw : switches do
4:       sw.written=false
5:       sw.flush()
6:       if sw.needSelect then
7:          sw.selectKey.addOp(WRITE)
8:       end if
9:    end for
10:   readySwitches=select(switches)
11:   for all Switch sw:readySwitches do
12:      if sw.selectKey.readable then
13:         readAndProcessMessages(sw)
14:      end if
15:      if sw.selectKey.writeable then
16:      sw.needSelect=false
17:      sw.flush()
18:      end if
19:   end for
20: end while
21: end function
```

(6)Floodlight拓扑发现原理分析

Floodlight同时采用LLDP和BBDP实现链路发现服务。本节首先会介绍传统网络通过LLDP进行链路发现的原理;然后会分析在控制平面层和数据平面层分离的SDN网络中,控制器和OpenFlow交换机之间采用OF协议进行链路发现的流程。

①LLDP分析

LLDP实现原理:

LLDP[15]架构如图6所示,在每个交换机上会维护两个管理信息库MIB(Management Information Base)—LLDP local system MIB和LLDP remote system MIB。这两个MIB[16]主要用于管理,此外,还有四个MIB库,分别对应着实体MIB、物理拓扑MIB、接口MIB以及其他MIB。

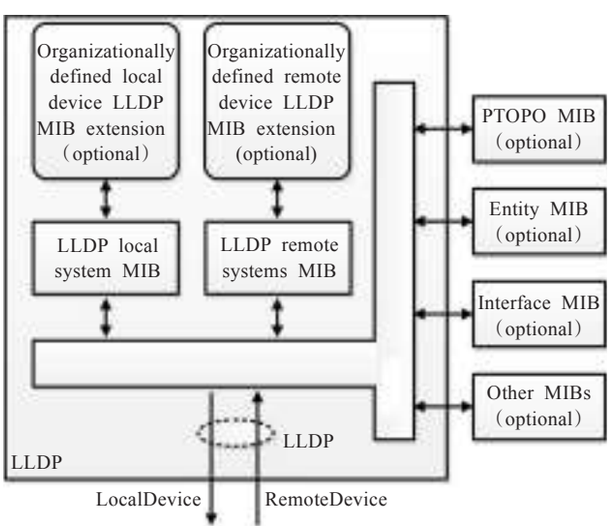

图6　LLDP架构

LLDP agent通过与上述四个MIB库的交互,可以更新自身的LLDP local system MIB库,然后再通过LLDP帧,将自身的一些信息通过连接到远端设备上的接口发送到远端设备。

同时,交换机也会接收远端设备传送过来的LLDP帧,从而更新LLDP remote system MIB。通过这种方式,设备就可以通过所有连接到相邻设备上的接口来更新并维护自己的LLDP remote system库。而相邻设备信息主要有:连接的接口、所连接接口的MAC地址等。整个LLDP模块收发信息均通过MIB组织。每个设备则会周期性地或者当设备状态发生变化时将LLDP帧发送给相邻设备。

当Local System MIB或Remote System MIB中信息发送变化时,交换设备就会向网管设备汇报。

LLDP帧格式:

LLDP报文的格式如图7所示。

DA	SA	LLDP Ethertype	Data+pad	
LLDP Multicast address	MAC address	88-CC	LLDPDU	FCS
6 Byte	6 Byte	2 Byte	1 500 Byte	4 Byte

图7　LLDP报文格式

其中各字段的含义如下所示。

LLDP报文的目的地址是一个组播地址,它的值为:01-80-C2-00-00-0E。源MAC地址是本地设备的以太口地址。帧类型(LLDP Ethertype)的取值是0x88CC,设备正是通过此字段来判断该帧为LLDP帧,并将其交给LLDP模块来处理。LLDPDU(LLDP Data Unit,LLDP数据单元)是信息交换的主体,基本的信息单元用TLV(Tag-Length-Value)的形式表示,其中有4个类型是默认的:End of LLDPDU、Chassis ID、Port ID和TTL。FCS为帧校验位。

LLDP的发送过程:

LLDP MIB Manager从本地MIB信息库之中获取本地设备的信息,并将其封装到相应的TLV之中,然后将TLV封装成LLDPDU的格式发送出去。LLDPDU分成两类,其中一类是普通的LLDPDU,被用来描述设备的相关信息;另外一类是“shutdownLLDPDU”,它被用来告知对端设备,其远端的MIB库之中的信息将要老化,该信息应该被删除。

LLDP的接收过程:

LLDP帧的接收过程也就是对LLDPDU进行拆分

和检查的过程，它可以分为3个过程：帧确定、帧有效性检查、更新LLDP remote system MIB。帧确定主要是检查帧的目的地址和帧类型字段是否正确，如果不正确则丢弃该数据包；有效性的检查主要涉及到LLDPDU内容的检查，一般会先检查LLDPDU的头三个TLV是否正确；最后一步是更新MIB库，因为Port ID和Chassis ID可以唯一确定一个邻居设备，所以根据解析出来的Port ID和Chassis ID来查找远端MIB库，如果找到匹配，便更新相应的信息，否则建立一个新的MIB库。

②Floodlight中的拓扑发现机制

在Floodlight中，同时采用LLDP和广播包（BDDP）进行链路探测。LLDP的目标地址是01：80：c2：00：00：0e，而BDDP的目标MAC地址是ff：ff：ff：ff：ff。两种数据帧类型值分别是0x88cc和0x8999。Floodlight中定义了两种链路，它们分别是直接链路和广播链路。直接链路通过LLDP数据包发现，广播链路通过广播包发现。

Floodlight控制的SDN网络中，网络拓扑发现的主要流程如图8所示。

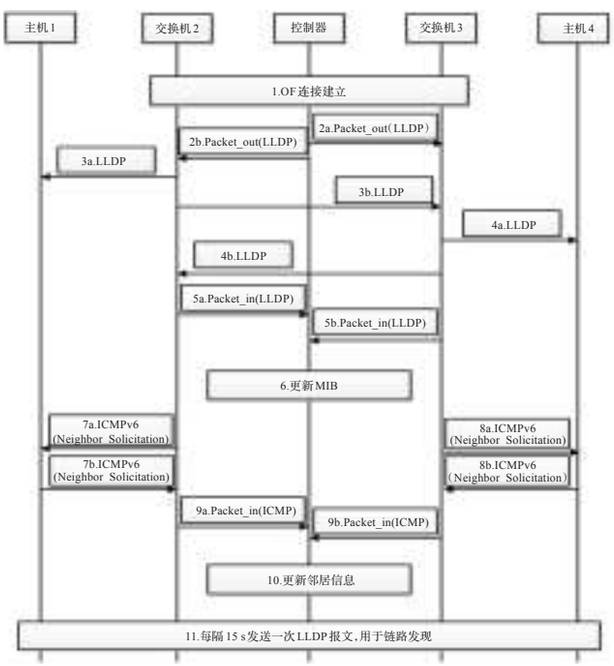

图8　网络拓扑发现主要流程

（a）首先控制器与交换机之间建立OF协议通道，控制器获取交换机的MAC地址、端口号、设备标志等信息，并用其来维护本地的LLDP MIB。

（b）控制器从本地的LLDP MIB设备信息库之中获取本地设备的信息，并将信息封装到TLV之中，然后再将TLV封装成LLDPDU格式的LLDP报文。得到上述报文后，再通过OF协议将LLDP报文封装成packet_out报文，并将其发送到交换机，交换机会根据交换机上的端口数量来构造相应数量的LLDP报文。

（c）交换机2接收到packet_out报文之后，会根据其中的actions将其中的LLDP报文从相应的端口发送出去。

（d）同样地，交换机3收到packet_out消息后，会执行与交换机2相同的操作。若还有其他交换机，步骤同上。

（e）交换机收到对端端口的LLDP报文之后，会构造packet_in消息反馈给控制器，告诉交换机，LLDP报文已经收到。

（f）控制器根据接收到的LLDP报文，分别更新交换机2与交换机3的LLDP remote MIB，从而完成链路探测。

（g）在网络启动伊始，根据IPv6中的NDP（Neighbor Detection Protocol）协议，交换机2向各个端口发送Neighbor Solicitation报文，以用于邻居探测请求，当主机1收到NDP协议包后同样构造Neighbor Solicitation消息，并将其发送到交换机2。

（h）同样地，交换机3向各个端口发送Neighbor Solicitation报文，以用于邻居探测请求，当主机4收到NDP协议包后同样构造Neighbor Solicitation消息，并将其发送到交换机3。

（i）交换机将接收到的Neighbor Solicitation消息封装为packet_in包发送到控制器。

（j）控制器收到之后更新邻居信息。

（k）此后每间隔15 s控制器与交换机之间就会重复进行步骤2到6，周期性地进行链路的更新。

在更多情况下，不同的OF域之间还会通过传统的L2交换机相连，则步骤2到步骤6还会发送BDDP报文用于广播链路的发现，其过程与LLDP基本相同。

4　SDN控制器性能评估与比较

为了对控制器的特性和性能做更进一步的分析和研究，本文采用cbench和hcprobe这两个测试工具对目前已有的7种开源OpenFlow控制器的性能进行了测试和评估。

4.1　实验平台

实验平台由两台服务器构成如图9所示。每台服务器都有两个Intel Xeon E5645六核处理器，2.4 GHz的主频以及48 GB的内存。两台服务器都运行Ubuntu 12.04.1 LTS（GNU/Linux 3.2.0-23-generic x86 64）操作系统。其中一台服务器用来部署控制器，为了消除处理器之间通行造成的影响，将所有控制器都绑定到服务器上的一个处理器核心。另外一台服务器则用来根据特定测试用例来产生流量。

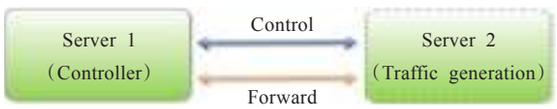

图9　实验平台

4.2　测试用控制器

本文测试用SDN控制器有如下7种。

NOX为一个多线程的基于C++的控制器,POX为一个单线程的基于Python的控制器,它被广泛用于网络应用快速原型研究,Beacon为一个多线程的基于Java的控制器,这个控制器依赖于OSGi和Spring框架,Floodlight为一个多线程的基于Java的控制器,MUL为一个多线程的基于C的控制器,Maestro为一个多线程的基于Java的控制器,Ryu为一个单线程的基于Python的控制器。

4.3　实验方法

测试的实验方法包括网络拓扑虚拟化、SDN控制器性能和可扩展性的评估以及可靠性的分析。

4.3.1　虚拟化

实验中通过cbench来虚拟化底层网络拓扑,自定义网络中交换机以及主机的数量,抽象并整合底层网络资源,从而实现对各种不同SDN控制器的测试和评估。

4.3.2　性能/可扩展性

SDN/OpenFlow控制器的性能由两方面决定:吞吐量(flows/s)和延迟(响应时间,ms)。实验通过控制网络中交换机数量、主机数量以及服务器的CPU核心数这三个变量对7种控制器的性能进行测试和评估,同时体现了控制器的可扩展性。

(1)吞吐量

本文主要分析了以下参数与控制器吞吐量之间的关系:

①连接到控制器的交换机数量(1,4,8,16,64,256),每台交换机连接的主机数量相同。

②交换机连接的主机数量(10^3,10^4,10^5,10^6,10^7),控制器连接的交换机数量均为32台。

③可用的CPU内核数量。

(2)延迟

延迟是通过一台交换机向控制器发送请求到接收到来自控制器的回复所用时间的平均值,本文实验测量5次往返时间的平均值。

4.3.3　可靠性评估

为了评估可靠性,测试了在长时间给定负荷下各个控制器出错的次数。负载文件通过hcprobe生成。

在测试中,使用5台交换机以每秒2 000到18 000次请求的频率向控制器发送OpenFlowPacketIn消息。测试时间一共持续48 h,记录了在48 h内每种控制器的出错次数。

4.4　实验结果

(1)吞吐量

图10显示了在不同交换机数量下,各个控制器的吞吐量变化,由于POX和Ryu都是单线程的控制器,所以图10显示交换机的数量的变化并不影响其吞吐量。对于多线程的控制器,由于交换机数量的增加可以增加CPU核心的利用率,所以图10显示在交换机数量增加时,多线程控制器的吞吐量普遍增加,但交换机数量增加到与CPU核心数量基本相同时,控制器吞吐量就基本不变了。同时如图10所示,对于Maestro这种控制器来说,在交换机数量较小时,其性能较为优异,但当交换机数量增加到较大时,其吞吐量反而会降低。

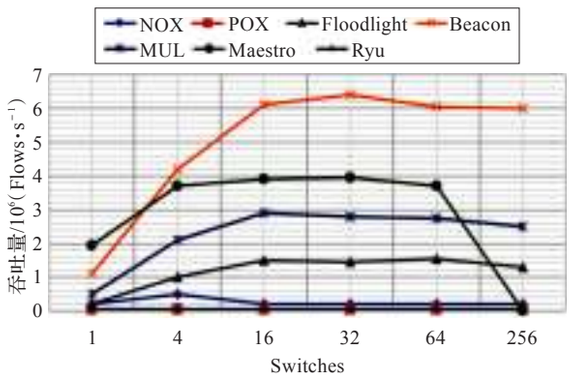

图10　交换机数量对吞吐量的影响

图11显示了在不同主机情况下,各个控制器的吞吐量变化。如图11所示交换机所连接的主机数量对大多数控制器的吞吐量并没有实质性的影响。但对于Beacon这种控制器来说,当主机数量提升到10^7时,其吞吐量有了一定的下降;而对于Maestro来说,当主机数量提升到百万数量级时,其吞吐量显著下降。

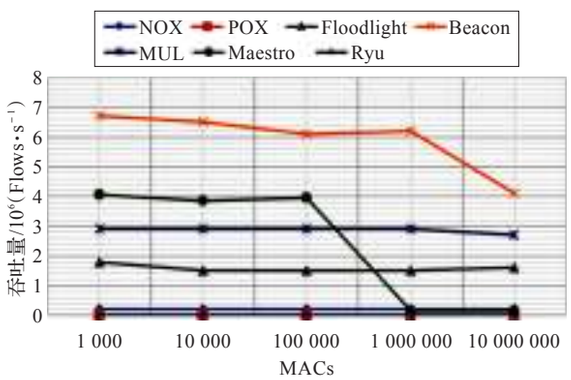

图11　主机数量对吞吐量的影响

图12显示了在CPU核心数量变化时,各个控制器的吞吐量变化。由于Pox和Ryu这两种控制器是单线程控制器,所以当CPU核心数量增加时,其性能并没有

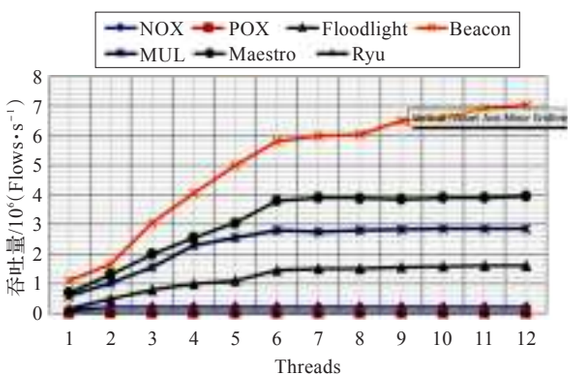

图12　CPU核心数量对吞吐量的影响

变化。Maestro的吞吐量在核心数量达到8个之后就基本保持稳定，可以看出Maestro最多可以利用8个CPU核心。而对于Floodlight来说其吞吐量随着CPU核心数的增加也呈现稳定增加。Beacon则表现出最强的可扩展性，其吞吐量随着CPU核心数量的增加也显著提升。

对吞吐量的实验说明SDN控制器能很好地实现对底层网络拓扑的虚拟化，完成对底层网络资源的抽象和整合，同时大多数控制器具备较强的可扩展性，能实现在横向扩展模型上添加物理网络功能并且控制器能将网络抽象为单个设备进行管理。

(2)延迟

各个控制器的平均响应时间如表2所示，实验显示各个交换机的平均延迟时间(每flow10^{-6} s)。此实验中一台交换机与10^5台主机相连，如表2所示，Mul和Beacon这两种控制器的平均延迟时间最短，而平均延迟时间最长的则是基于Python的两种控制器：POX和Ryu。

表2　平均延迟时间

控制器	平均延迟时间(每flow10^{-6} s)
NOX	91 531
POX	323 443
Floodlight	75 484
Beacon	57 205
MuL	50 082
Maestro	129 321
Ryu	200 021

(3)可靠性

实验结果显示测试用控制器在给定的负荷下均正常运行，没有常出现关闭连接和丢包的情况。可靠性实验说明测试用控制器在测试条件下都具备较高的可靠性。可靠性测试如表3所示。

表3　可靠性测试

控制器	关闭连接数	丢包数
NOX	0	0
POX	0	0
Floodlight	0	0
Beacon	0	0
MuL	0	0
Maestro	0	0
Ryu	0	0

5　SDN控制器特性总结

传统的网络由数据面、控制面和管理面构成，其中的控制面实现OSI模型[17]中3到7层的各类报文转发控制功能。比如支持IP路由协议(如RIP/BGP/OSPF)更新路由表，支持第二层物理主机学习并更新MAC(Media Access Control，介质访问控制)地址，根据过滤功能或用户配置来生成ACL(Access Control List)转发控制表[18]。数据面则根据控制面传来转发信息表进行查表、报文解析、过滤匹配和端口映射等与报文转发相关的功能。而管理面则支持用户对设备的配置和管理，比如实现简单网络管理协议(Simple Network Management Function，SNMP)和命令行接口(Command Line Interface，CLI)功能。虽然交换设备从功能逻辑上做了较好的划分，但控制面需要支持的各种各样协议规范，比如BGP、OSPF、RIP、组播、流量工程、防火墙、区分服务、网络地址转化、多协议标签交换、虚拟局域网等已使路由器的设计与实现都变得极其复杂。

而在SDN网络中，与原来的网络体系不同，交换机只起到转发的作用，而SDN控制平面层则将原路由器/交换机设备中的控制功能分离出来形成SDN控制器。SDN控制器制定转发策略，并将这些转发策略传递给使用OpenFlow协议网络上的设备以完成数据转发，从而完成控制器对数据平面层网络设备的灵活管控。不仅如此，SDN控制器向上提供各种开放API，使上层应用可以调用并完成对网络设备信息、流量信息、网络QoS参数等的控制。

本文基于以上对Floodlight控制器的详细分析以及对7种开源控制器的性能比较，对SDN控制器的特性进行了分析和总结。

(1)支持OpenFlow

OpenFlow协议是SDN控制器南向接口的核心协议，也是SDN控制器实现对底层网络资源抽象和整合的基础。当一个数据包到达OpenFlow交换机时，其头字段会和流表的记录比较。如果找到匹配项，则根据流表中记录的操作将数据包转发或者丢弃数据包。当OpenFlow交换机接收到数据包与流表中的记录不匹配时，OpenFlow交换机会将数据包封装并将其传给控制器。控制器决定数据包应该如何处理并且通知交换机丢弃数据包或者在其流表中增加一条记录以支持新的数据流。

(2)网络虚拟化

SDN最大的优势之一就是实现对网络的虚拟化。显然类似于VLAN(Virtual Local Area Networks，虚拟局域网)和VRF(Virtual Routing and Forwarding，虚拟路由和转发)，这些网络虚拟技术实用性很强，但是它们不管在范围还是价值上其实都是受到限制的。为了达到价值的最大化，网络虚拟化必须是端到端的，并且它必须以一种类似于服务器虚拟化抽象和联合计算资源的方式来完成对网络资源的抽象和整合。这些功能促使面向特定租户的虚拟网络的创建，这些虚拟网络的拓扑结构源自于底层的物理网络，并且它也使得IT机构能够动态地创建基于策略的虚拟网络以满足大范围的需求。而SDN控制器则能向应用层提供网络虚拟化的接口(北向接口)，将虚拟网络从物理网络解耦。上述实验也正是使用虚拟化的方式完成了对各种控制器的性

能测试和比较。

(3)网络功能化

在安全性的要求下,云设备供应商通常希望将租户彼此之间隔离起来。虽然这种需求通常是相对于公共云服务提供商来说的,但实现用户隔离也是大多数行业和IT组织的需求。为了有效地满足这个需求,SDN控制器必须能使虚拟网络之间相互隔离,同时能将虚拟网络集中配置并自动执行配置内容。

在报文头部有12个元组可以用来匹配流表中的项。为了能让流在被传送时有最大的灵活性,所以SDN控制器在做路由决定时基于多种头字段是很重要的。同样重要的还有控制器能在一个流接一个流的基础上定义QoS参数。

相对于路由流量来说,SDN控制器一个更重要的特征是它能在流的源和目的之间发现多条路径的能力,以及在多个链路之间对给定流的分流能力。这种能力消除了生成树协议的局限性并且增加了解决方案的有效性和可扩展性。这种能力也消除了通过如TRILL(Transparent Interconnection of Lots of Links)或者SPB(Shortest Path Bridging)等协议增加网络复杂性的需求。

Floodlight的模块化启动方式以及底层的拓扑发现机制就很好地体现了这一特性。

(4)可扩展性

传统的局域网被部署在多层架构之中,在此架构中第3层的路由功能是连接多个第2层网络。而支持东西向数据传送时,这些传统的局域网的扩展性并不是很好,因为至少有一台(更多情况下是多台)第3层设备在端到端的路径之中。

SDN允许IT组织使用一个可扩展的网络模型。在此网络模型下,IT组织能在它们需要时增加网络功能,并且SDN控制器允许这些IT组织与机构对所有的这些网络功能进行统一管理。一个与SDN可扩展性至关重要的因素是SDN控制器能支持交换机的数量。在当前环境下,IT组织机构一般期望它们所获得的控制器至少能够支持100台交换设备,但是它们也应该意识到一个控制器可支持的交换机数量将取决于正在被支持的SDN用例。

上述多种SDN控制器性能比较实验说明了SDN控制器具备在横向扩展模型上添加物理网络功能并且控制器能将网络抽象为单个设备进行管理。

(5)可靠性

SDN控制器能提供设计验证并且这种验证可以消除常见错误,从而增加网络的可用性。即使如此,SDN仍然受到一种质疑,那就是SDN控制器是一个单独的节点,所以如果SDN控制器出现故障,整个网络的可用性都会受到影响。为了应对这种质疑,SDN控制器的设计采用了一些措施,其中之一就是在源和目的之间寻找多条路径,这样网络的可用性就不会受到单一链路故障的影响。如果SDN控制器在源和目的之间只设置一条路径,当面对链路故障时,控制器必须将流重定向到一个可用链接之上,而这则需要控制器持续地监控整个网络拓扑结构。相对于外部链接的可用性,更重要的是SDN控制器能支持技术和设计上的可选性,从而增加网络的可靠性。

在控制器自身的可用性方面而言,至关重要的是控制器自身使用支持冗余设计的硬件和软件进行构造,因为支持冗余设计,任意单点故障都不会影响整个系统运行,甚至是关键节点。与此同样重要的是控制器能够支持集群。比如两个SDN控制器的集群处于活跃热待机模式可以增加网络的可靠性;3个集群的控制器,其中一个处于热待机模式,会增加SDN控制器的可用性、可扩展性和整体性能。然而为了使集群能够提供非常快速的故障切换时间,处于活跃状态的控制器与处于等待状态的控制器能维持存储器的同步显得至关重要[19]。

可靠性最主要的特性可以概括为以下三点:

①支持从源到目的的多样化路径;

②利用支持冗余设计硬件和软件来构建控制器;

③形成控制器集群。

(6)网络安全性

为了能保证网络的安全,一个SDN控制器必须能支持企业级身份验证和网络管理员授权。此外网络管理员必须能够控制对SDN控制流的访问。SDN控制器安全性策略之一就是采用复杂的过滤器来过滤数据包。另外一种策略就是租户之间必须完全隔离。

除此之外,因为SDN控制器很容易成为攻击的目标,SDN控制器需要能限制控制通信的速率并且在可疑攻击来临时能提醒网络管理员。

6 结束语

SDN是一种全新的网络架构,从提出到现在,它已经迅速地从实验室走出,引起了各界的广泛关注。它的提出旨在对现有复杂网络的虚拟化和模块化,将控制与转发分离,使网络面向应用可编程。SDN控制器可以说是SDN网络的核心所在,它向上提供开放应用接口,向下控制数据转发。本文对SDN的发展和技术架构进行了分析总结。并通过Floodlight这个开源项目对SDN控制器做了具体的分析和研究,最后通过对目前已有的7种控制器的性能实验以及SDN相关原理对SDN控制器的特性进行了分析和总结,以期为SDN控制器的研究与发展做出有益探索。

参考文献:

[1] ONF Market Education Committee.Software-defined net-

working：the new norm for networks[EB/OL].[2016-01-05]. http://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf.
[2] 王淑玲，李济汉，张云勇，等.SDN架构及安全性研究[J].电信科学，2013(3)：117-122.
[3] McKeown N，Anderson T，Balakrishnan H，et al.OpenFLow：enabling innovation in campus networks[J].ACM SIGCOMM Computer Communication Review，2008，38(2)：69-75.
[4] IONA Technologies.Technology review[EB/OL].[2016-01-05]. http://www.technologyreview.com/article/412194/tr10-software-defined-networking/.
[5] 张顺淼，邹富民.软件定义网络研究综述[J].计算机应用研究，2013，30(8)：2246-2251.
[6] OpenFlow.OpenFlow configuration and management protocol OF-CONFIG 1.0[EB/OL].[2016-01-05].https://www.opennetworking.org/images/stories/downloads/of-config1dot0-final.pdf.
[7] Open Network Foundation(ONF).OpenFlow configuration and management protocol v1.1[EB/OL].[2016-01-05].https://www.opennetworking.org/images/stories/downloads/of-config-1.1.pdf.
[8] Casado M，Freedman M J，Pettit J，et al.Ethane：taking control of the enterprise[J].ACM SIGCOM Computer Communication Review，2007，37(4)：1-12.
[9] Open Network Foundation(ONF).OpenFlow switch specification v1.3.0[EB/OL].[2016-01-05].http://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf.
[10] Jain S，Kumar A，Mandal S.B4：experience with a globally-deployed software defined wan[C]//Proceedings of ACM SIGCOMM Conference，2013.
[11] Hong C，Kandula S，Mahajan R.Achieving highutilization with software-driven WAN[C]//Proceedings of ACM SIGCOMM Conference，2013.
[12] Floodlight[EB/OL].[2016-01-05].http://www.projectfloodlight.org/floodlight/.
[13] BigSwitch[EB/OL].[2016-01-05].http://www.bigswitch.com/products/SDN-Controller.
[14] Erickson D.The Beacon OpenFlow controller[C/OL]//HotSDN'13，2013.http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p13.pdf.
[15] Attar V Z，Chandwadkar P.Network discovery protocol LLDP and LLDP-MED[J].International Journal of Computer Applications，2010，1(9)：93-97.
[16] Ericsson A B.Simple network management protocol[EB/OL].[2016-01-05].http://www.erlang.org/doc/apps/snmp/snmp.pdf.
[17] Balchunas A.OSI Reference model[EB/OL].[2016-01-05]. http://www.routeralley.com/guides/osi.pdf.
[18] Balchunas A.Access control lists[EB/OL].[2016-01-05]. http://www.routeralley.com/guides/access_lists.pdf.
[19] Cai Z，Cox A L，Ng T S E.Maestro：a system for scalable OpenFlow control，technical report TR10-08[R/OL].Rice University，2010：1-10.http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.185.968&rep=rep1&type=pdf.

(上接90页)

[8] Ahmad S G，Munir E U，Nisar W.PEGA：a performance effective genetic algorithm for task scheduling in heterogeneous systems[C]//2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems(HPCC-ICESS)，2012：1082-1087.
[9] Neto R F T，Filho M G.Literature review regarding ant colony optimization applied to scheduling problems：guidelines for implementation and directions for future research[J].Engineering Applications of Artificial Intelligence，2013，26(26)：150-161.
[10] Poli R，Kennedy J，Blackwell T.Defining a standard for particle swarm optimization[C]//Swarm Intelligence Symposium，2007：120-127.
[11] 郑光勇，李肯立，潘果，等.化学反应优化算法求解最小顶点覆盖问题[J].小型微型计算机系统，2015，36(2)：301-305.
[12] 陈国良，王煦法，庄镇泉，等.遗传算法及其应用[M].北京：人民邮电出版社，2001.
[13] 赵国亮，李云飞，王川.异构多核系统任务调度算法研究[J].计算机工程与设计，2014，35(9)：3099-3106.
[14] Xu Y，Li K，Hu J，et al.A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues[J].Information Sciences，2014，270(6)：255-287.
[15] 程子安，童鹰，申丽娟，等.双种群混合遗传算法求解柔性作业车间调度问题[J].计算机工程与设计，2016，37(6)：1636-1642.
[16] 袁礼海，宋建社，毕义明，等.混合遗传算法及与标准遗传算法对比研究[J].计算机工程与应用，2003，39(12)：124-125.