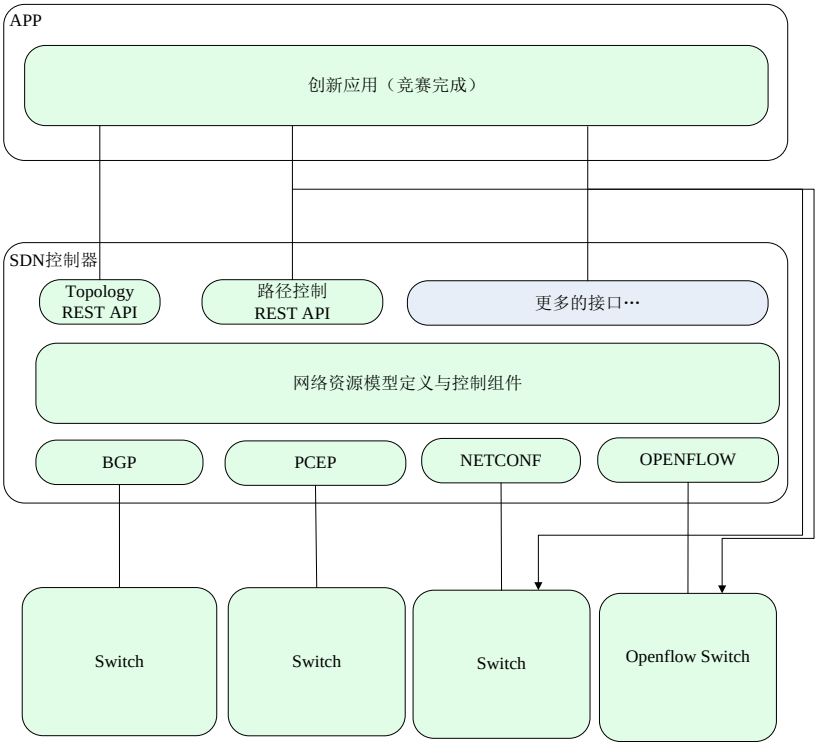
第三题：设计题（SDN 应用方案设计）

方案五：预设基础架构的创新应用开发



背景：实际生产中厂商提出的架构和实现 SDN 的方式多种多样，在不同厂商给出的基础架构上实现应用的创新与开发是有必要的。

要求：基于厂商给出的基础架构和控制器，组件 SDN 网络，通过可以通过控制器获取网络拓扑并控制其转发路径。根据提供的不同语言的开放接口写出创新应用程序，解决现有网络中的问题。

目录

一．设计题实现思路

综合考虑五道题后，我们决定选择第五题设计题，利用 SDN 架构的控制器实现基于用户的绿色通信网络休眠调度。我们做的算法添加为 SDN 架构北向接口的创新应用，通过添加控制器转发调度模块（创新 APP 应用）的方式实现，如图 1 所示。

在我们的研究过程中，首先基于 OpenFlow 协议，使用 Floodlight Controller + Mininet 的环境搭建虚拟 SDN 网络并添加我们的创新应用模块来实现。接下来再考虑将添加的算法调度模块移植到 H3C 厂商提供的 VCF Controller 上，在硬件上实现绿色通信网络的休眠调度机制。

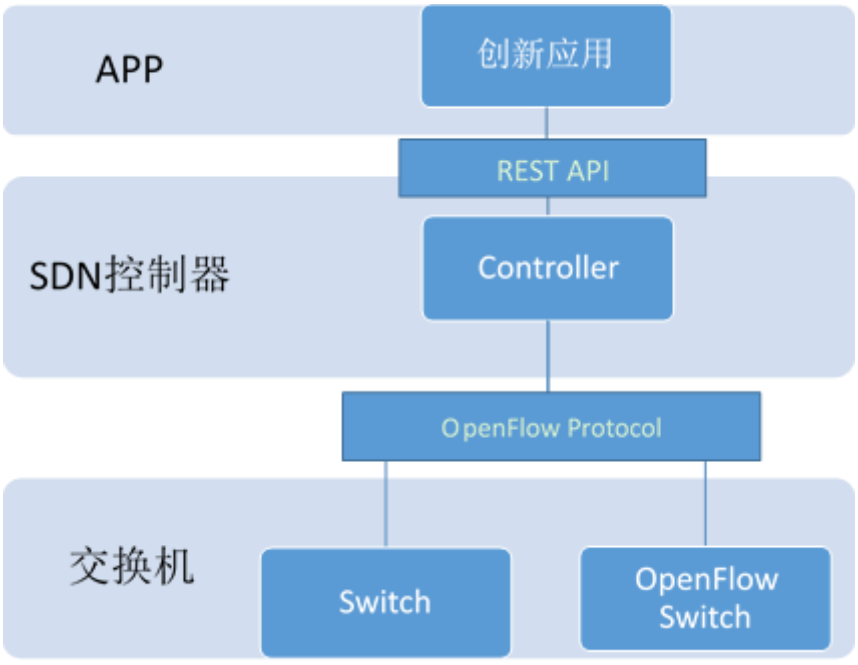


图 1　SDN 架构及北向创新

考虑到目前的交换机上没有休眠调度的功能，所以前期在虚拟环境下模拟网络算法实现时，只要实现将数据包聚合到部分交换机上进行转发、传输、丢弃等操作，将其余的交换机和链路闲置，这样在实际操作时添加一个交换机闲置时休眠的功能模块即可实现。

二．选题背景介绍

2.1 研究背景

据全球权威机构调查，通信信息行业每年的能耗约占全球的 3%，每年的碳排放量约占全球的 2%，超过整个航空业的碳排量，并具有持续增长势头[1,2]；而我国政府每年 800 亿元能源消耗中有 50%来自 IT 产品[3]。随着通信信息产业的迅猛发展，通信网络中的能耗成为一个不可忽视的问题。特别是近年来，云计算和云服务的发展，智能终端的出现，带来了大量的网络流量；通信网、传感网、因特网和物联网等基础设施的建设，增加了网络的规模和能耗。到 2020 年，全球需减排温室气体 15%-30%才可以避免全球温度上升 2 度[1]。我国在“十二五”规划中已经将节能减排列为重大专项，国家工业与信息化部也制订了通信业节能减排政策。

为此，有必要研究如何提高网络的能效，降低网络的整体能耗，实现绿色通信网络。事实上，当前网络中能量的利用率仍然非常低[4,5]。这由两个方面的原因造成。第一，网络设计的冗余性。目前的网络基础设施都是针对流量高峰期或者最坏情况而设计的，而在普通情况下不能被有效地利用。比如，在凌晨 3:00-7:00，网络设备基本处在闲置状态，而这种情况下的设备耗电量约占满载时候的 90%[6,7]。再如，在无线通信中，设备空闲监听消耗了接收过程能耗的 50%以上。实验测量结果显示，手持设备的空闲、接收、传输能耗比为 1:1.05:1.4[8]；WiFi 无线局域网的能耗比为 1:2:2.5[9]。大量的闲置网络设备浪费了数量可观的电能 第二，用户业务的动态性。网络设施总是以用户业务峰值来设计，但由于人类的活动规律（如作息规律）、用户的移动性（如移动互联网）以及业务的动态性等，网络业务在时间和空间上呈现分布的随机性，在某个时间或地区，可能出现轻负载或无负载。因此，能使闲置状态的设备休眠的技术受到广泛关注,即当网络流量较低时，把网络设备关闭或者置于睡眠状态；当流量增大的时候再把它开启或者唤醒，这样可以使设备不必全天候处在高功率运转状态,以大大降低空闲设备造成的能量浪费。

2.2 国内外研究现状

通信网络的能耗问题网络的能耗问题已经引起了一些国际组织、机构和研究者的关注。2010 年 1 月，由贝尔实验室牵头的 Green Touch 成立，致力于未来 5 年通信网络有效性提高千倍的研究。国际电子与电气工程师协会(IEEE)于 2011 年组织了三期绿色通信专刊，并认为绿色通信技术将是未来 10 年通信行业的重中之重[12]。国际著名网络期刊(Computer Networks)于 2012 年也组织了一期绿色网络技术方面的特辑。上述所提到的专刊特辑均有我国学者参与发起，如浙江大学的张宏纲教授和清华大学的牛志升教授等。绿色通信已是当前学术界所关注的研究热点之一。

路由器是网络的主要设备，高性能、高速率路由器等设备的持续全功率全时段运行存在着大量的能量浪费；移动互联网，云计算数据中心等应用的涌现，使得网络的规模扩大和用户流量的激增，并消耗了大量的能量。针对这些问题，有学者提出了“绿色互联网”的概念[5]，通过基于全网的协调控制和调度，构建绿色高能效的互联网。实现这一目标的第一步是建立网络设备的能耗模型。

由海量网络业务催生的云计算数据中心往往由成千上万的服务器构成。数据中心的能耗问题受到了广泛的关注。为解决数据中心的能耗产生的散热问题，Google 等大型提供云服务公司甚至把数据中心建设到北极或者深埋地底。云计算数据中心网络将是休眠技术的应用热点[13]。

随着组播，区分服务，流量工程，MPLS 等功能的布置，当前的互联网结构体系架构已经变得越来越复杂，在新技术和新应用的开展方面缺乏灵活性。

软件定义网络针对这种变化对网络架构提出的创新，当前的网络架构，协议，算法等均没有考虑到设备的休眠行为，对基于用户的网络休眠机制的深入研究，有助于构架新型的绿色通信网络架构和理论。SDN 创新架构可以提供灵活的创新平台，并已经在数据中心得到应用。由于当前连接 WAN 网的链路带宽利用率很低，谷歌公司选择使用 SDN 来改造数据中心之间互联的 WAN 网（即 G-scale Network），因为这个网络相对简单，设备类型以及功能比较单一，而且 WAN 网链路成本高昂（比如很多海底光缆），所以对 WAN 网的改造无论建设成本、运营成本收益都非常显著。他们把这个网络称为 B4 网络。改造完成后，链路带宽利

用率提高了 3 倍以上，接近 100%[15]。

三．设计方案和实现方法

3.1 问题分析

将空闲的设备休眠是最一种直觉的节能方法。接入设备（如蜂窝网的基站，无线局域网的接入点等）是网络能耗最大的设备，约占 50%以上。因此，如果能让接入设备监听业务负载，在没有用户业务时，就可以休眠或关闭接入点[10]。同样，网络的中继设备也可以应用休眠技术[11]。设备是否休眠取决于网络中业务的分布。

我们把网络设备的休眠看做一个动态选路和设备调节的过程，休眠的前提是设备的空闲。由于用户业务的动态性，设备的休眠需要适应于用户的业务量和网络的状态。休眠技术的应用使得网络设备处于频繁的休眠/唤醒操作过程中，以适应动态变化的业务。当业务流量增加时，需要唤醒更多的设备，以支持新增的网络业务。传统的网络优化一般把休眠节能问题看成一个静态的网络效用最大化问题，这不符合动态的网络业务和休眠操作的特点。因此网络的休眠应该是一个设备休眠和网络业务动态平衡的过程。

3.2 研究目标

在网络中，用户有自己的业务需求。传统网络中的用户会选择自己的路由传递信息。在 SDN 架构的网络下，控制器可以运行算法为用户选路，通过控制器的选路，我们将有业务流量的路由器将处于活跃状态，处于空闲状态的路由器进入

休眠节能模式，从而达到节能的效果。

3.3 研究方法

结合我们的设计，充分调研和分析国内外相关方面的理论和实践，跟踪相关技术的最新发展动态；深入分析理解中央调控下的休眠机制，从理论上建立绿色网络休眠机制的数学模型，设计高效的休眠机制；通过理论分析和基于 Floodlight Controller +Mininet 的虚拟 SDN 网络构建以及休眠调度模块的添加来验证模型和算法的可行性，评估其性能；搭建基于 H3C 的 VCF Controller 的实验平台，通过实验验证网络休眠机制的可行性和效果。

四. 理论分析和算法实现

4.1 算法的系统模型建立

考虑一个通信网络 G(*V,E*)，其中 *V* 是网络节点（如路由器）集合，*E* 是连路集合。假设链路处于活跃时的能耗为C_e,休眠时没有能耗。由[6,7,15]可知，由于频道的闲置监听，网络设备处在闲置状态的设备耗电量约占满载时候的 90%，因此我们可以假定C_e为常数。我们可以进一步考虑更加实际的能耗模型，但本项目采用的分析框架和方法同样适用。网络中的用户数为 *k*，其中每个用户 *i* 有源节点S_i和目的节点d_i。为了通过网络传递业务，控制器为用户选择它们之间的一条路径。路径选择策略决定了用户的能耗。SDN 控制器为每个用户 i 选择 P_i,所有用户的路径选择就构建了网络的拓扑，即$\bigcup P_i$。

我们使用$\mathbf{k}_e$表示同时使用链路 e 的用户数，这样每个用户平均分享了$\mathbf{C}_e/\boldsymbol{k}_e$部分的链路能耗。花费共享机制对用户来说是公平合理的，它可以吸引用户共享

链路，从而降低网络能耗。图 2 描述了这种公平的共享机制。在源节点 s 和目的节点 d 之间存在多条路径，圆中的数字表示使用该条路径的用户数，链路边的数字表示每个用户的花费。

链路选择决定了用户的消费，假设控制器控制为用户 i 选择路径$\mathbf{P}_i$，则对于选择了 P_i 路径的用户 i，其总的能量消耗为

$$Cost_i(P_i) = \Sigma_{e \in P_i} c_e / k_e. \qquad (1)$$

我们研究了网络中轻负载情况下的的睡眠调度机制，这是休眠机制可行的证明。在这种网络中，用户的链路需求通常少于链路的容量，因此我们可以假设我

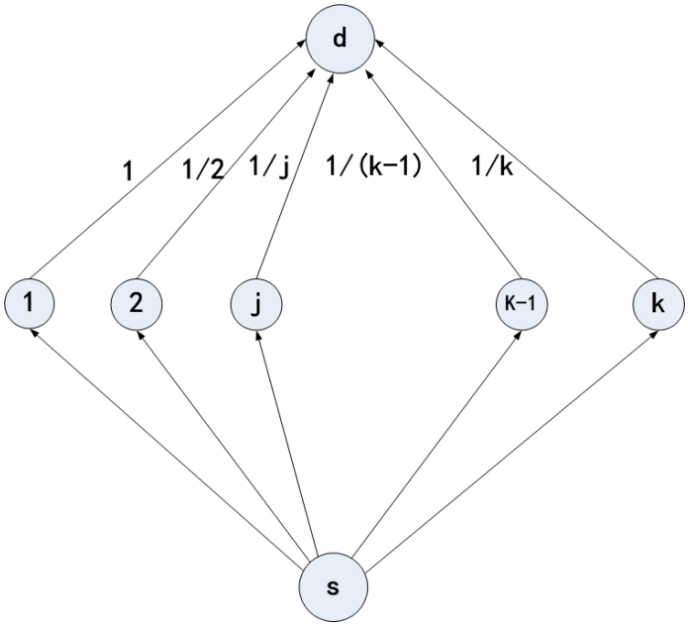


图 2 公平共享机制

们研究的模型中，链路没有负载上限。将控制器为用户选择的链路表示为策略向量$\mathrm{S} = \{P_1, P_2, \ldots, P_K\}$，S 是网络中一系列可行的策略空间。

我们研究的拓扑构成问题和互联网中的路由问题不同，互联网中路由问题的用户通常致力于寻找链路实现更高的吞吐量。为达到这个目标，每个用户会优先考虑避开网络阻塞，选择闲置链路。在绿色拓扑构成问题中，我们期望控制器为用户优先选择共享用户数目多的链路已达到降低用户花费的效果。如图 3 中所示，有箭头的实线表示绿色网络拓扑构成机制下用户选路的情况，虚线表示路由问题中的用户的选择。在绿色拓扑构成机制中，只有粗实线的几条链路需要开启，细实线的链路可以进入休眠状态。而在路由问题中，用户 1 和 2 各自选择了不会

冲突的路径实线高吞吐量，但是也增加了能耗。

在 SDN 架构下，网络设备将请求发给控制器，由控制器来解决拓扑形成问题。我们提出的绿色拓扑构成机制如下工作：

- 控制器基于公平的共享机制由我们设计得算法为用户选择路由链路
- 主干网络中的路由器基于目前提供的服务量决定切换状态，若路由器闲置则可以进入休眠模式或关闭

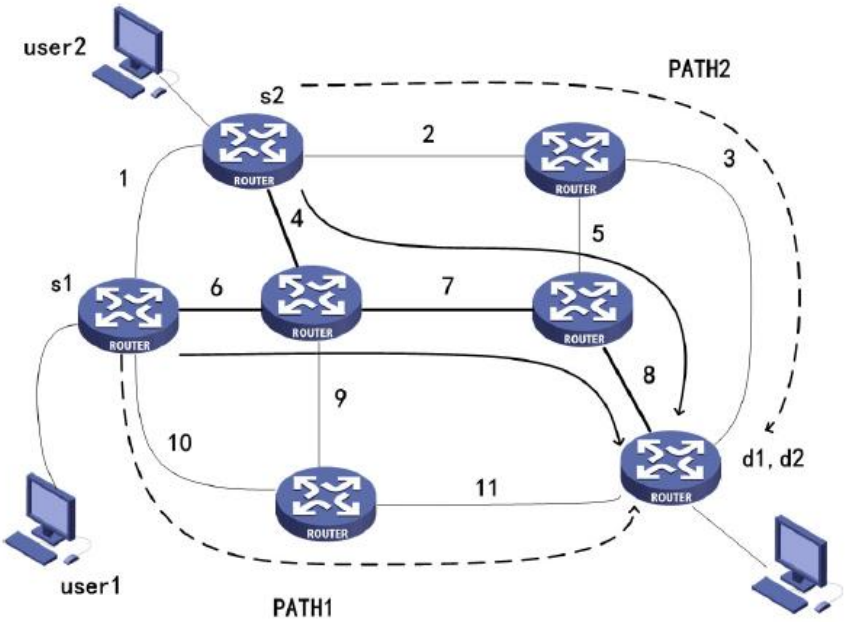


图 3 绿色拓扑网络机制下的路由选路

我们使用图 3 来说明链路共享和拓扑构成。用户 1 的源节点和目的节点为S_1和d_1，用户 2 的源节点和目的节点为S_2和d_2。控制器在收到用户请求后，为了达到能耗最小，控制器通过绿色拓扑网络算法为用户 1 和用户 2 选择共享链路 7 和 8，从而使其他链路闲置。选路完成的状态我们称之为均衡状态，这里用 NE 来表示你。选路之后，网络可以控制空闲的路由器关闭或休眠。

4.2 算法的优点

我们致力于提出新的路由算法从而达到实现绿色网络，提高网络能源利用率，

大幅度节约能源的目的。由于目前使用的通信网络在设计时并没有考虑能耗问题，能量的利用率实际上非常低。目前的网络容量设计是考虑到流量高峰期或者满负荷情况而设计的，具有冗余性的特点。在通常情况下，这会导致网络很难达到满载情况。因此，在通常情况下，网络容量会有大量的浪费情况存在[2]。

从这个角度出发，我们讨论了基于休眠调度的拓扑构成问题，并提出了一种基于用户激励的拓扑构成机制。基于我们提出的拓扑构成机制，可以减少处于开启状态的网络设备，提高网络利用率并节能。

4.3 算法的实现

为实线 SDN 控制器控制下的拓扑形成机制，我们需要引入最佳动态回复 BRD（Best Response Dynamics）算法。该算法核心思想如下

Algorithm BRD

1: state is note a NE
2: **while** state is not a NE **do**
3: **for** $i \in I$ **do**
4: **for** $e \in E$ **do**
5: choose P_i
6: **end for**
7: change $S = \{P_i\}$
8: **end for**
9: **if** S haven't changed
10: **then** state is a NE
11: **end if**

12: **end while**

在第五部分中，会设计具体如何将算法添加入 SDN 虚拟网络的 Floodlight Controller 控制器中，实现绿色网络休眠机制算法。

五．虚拟网络环境的搭建和算法模块实现

首先，我们使用 Floodlight Controller + Mininet 构建虚拟的 SDN 网络，然后将理论分析出的绿色网络休眠机制算法添加为 Floodlight Controller 的一个模块，整合之后通过分析，观察处于闲置状态的交换机数量以检测算法的优劣。

5.1 Floodlight+Mininet 的虚拟网络环境搭建

在官网上分别下载 Floodlight（http://www.projectfloodlight.org/floodlight/）和 Mininet（http://mininet.org/）的最新版本，在 Vmware 中导入 Mininet。

与前两道大题目搭建的虚拟 SDN 架构网络环境稍微有所区别的是，这里采用 Eclipse 导入 Floodlight Controller 运行，方便添加、修改模块从而实现我们需要的功能。

导入 Floodlight Controller 的步骤如下

（1）首先打开命令行终端，输入命令

```
cd floodlight
ant eclipse
```

（2）打开 eclipse，选择 import，导入已有 JAVA 项目，并选择 floodlight 所在

的目录即可

（3）运行程序的选项中选择添加新的运行方式：

Name 一栏填写 FloodlightLaunch, Project 一栏填写 Floodlight, Main 一栏填写 net.floodlightcontroller.core.Main

5.2 需要使用的 floodlight 模块

图 4 为 Floodlight 控制器的模块示意图，在这里我们主要用到控制模块中的

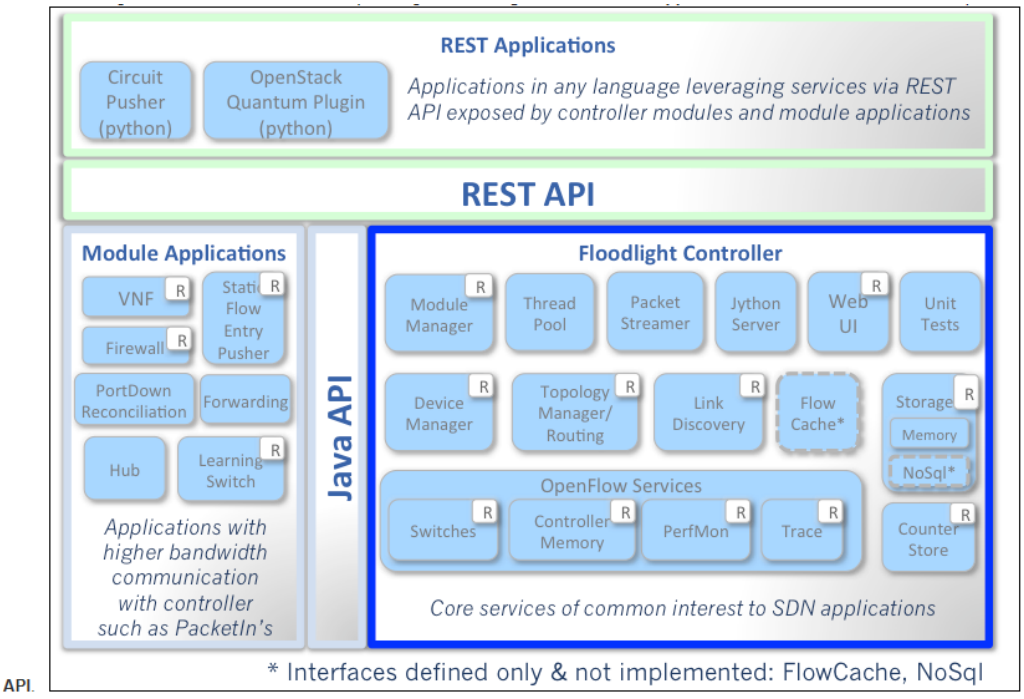


图 4 Floodlight 模块示意图

Device Manager 和 Packet Streamer。这两个模块内部又分为几个小模块，各自功能介绍如下：

（1） LinkDiscoveryManager

属于Device Manager，用于发现和维护OpenFlow网络的状态，对应模型位于net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager

（2） TopologyService

属于 Device Manager，用于维护控制器的拓扑信息，发现网络中

的路由器接入，对应模型位于 net.floodlightcontroller.topology.TopologyManager.

（3） RestApiServer

属于 Device Manager，通过 HTTP 协议开放了各个模块的 REST API（上图中附有 R 标志的模块）。

（4） PacketStreamer

该模块提供数据包流服务，用于为任意的交换机、控制器和观察者选择他们感兴趣的 OpenFlow 的数据包。该模块由两部分功能接口构成：1.基于 REST 的接口，定义了 OpenFlow 消息特性，称之为过滤器 2.基于 Thrift 的数据包过滤器

5.3 需要禁用和修改的模块

Floodlight 自带了转发模块 Forwarding。

Forwarding 模块位于 net.floodlightcontroller.forwarding.Forwarding。打开 Eclipse 观察 Forwarding 模块可见，Forwarding 模块调用 IDeviceService 类来确定源设备和目标设备，实现两个设备之间的数据包转发。Forwarding 模块并未提供路由功能，也没有 VLAN 的封装和解封功能，并由于 OpenFlow1.0 版本的局限性没有关于 cookie 匹配的说明。Forwarding 模块的转发方法使用了最短路径算法来完成找路，该路由算法不能满足我们的需要。

为了实现我们的休眠调度算法，转发模块 Forwarding 必须禁用。作为代替，将添加我们的绿色休眠调度算法作为新的转发模块，我们称之为 Scheduling 模块。在下一节给出具体实现方法。

禁用模块只需要在 src/main/resources/floodlightdefault.properties 找到 net.floodlightcontroller.forwarding.Forwarding 并注销此行即可。

进一步观察 Forwarding 模块的转发算法实现，发现具体算法实现在抽象类 ForwardingBase 文件内，如下图 5 所示。在我们的 Scheduling 模块中需要实现此抽象类，用于 4.3 节提出的 BRD 算法中的用户找路。

```
 2  *    Copyright 2011, Big Switch Networks, Inc.
17
18 package net.floodlightcontroller.routing;
19
20 import java.io.IOException;
59
60 /**
61  * Abstract base class for implementing a forwarding module.  Forwarding is
62  * responsible for programming flows to a switch in response to a policy
63  * decision.
64  */
65 @LogMessageCategory("Flow Programming")
66 public abstract class ForwardingBase
```

图 5 抽象类 ForwardingBase

5.4 算法模块实现

Scheduling 模块用于代替 Forwarding 模块，使用 4.3 节中的 BRD 算法以绿色网络休眠算法实现数据包的转发和交换机、链路的选择，需要以下功能保证正常执行算法同时不影响 Floodlight Controller 的其他功能运行。

(1) Scheduling能够调用LinkDiscoveryManager和TopologyService检测整个SDN网络的拓扑结构

(2) Scheduling模块执行优先级应该与Forwarding模块一致，从而保证不影响数据转发交换的逻辑顺序及其他模块的正常工作

(3) Scheduling模块可以监听数据包进入，分析数据包的转发请求

(4) Scheduling模块对收到的数据包进行分析，决定丢包还是转发或者报错

(5) Scheduling在收到数据包转发请求后能够使用绿色网络休眠机制算法计算每个数据包的转发路径，在为所有数据包做出选择后发出指令

(6) Scheduling能够发送log报告自己转发结果

另外，Scheduling 模块还应该实现发出指令让处于闲置状态的交换机进入休眠状态。由于目前厂商的交换机没有支持休眠机制，所以此处先写出方法，不写出方法的函数和具体的解，留作和厂商合作进行实际交换机实现的时候移植用。

本节下来分开详细介绍模块各功能的实现方法。

5.4.1 模块执行顺序和优先级设置

Floodlight 中模块通过 IOFMessageListener 接口监听 OpenFlow 消息，在 Eclipse

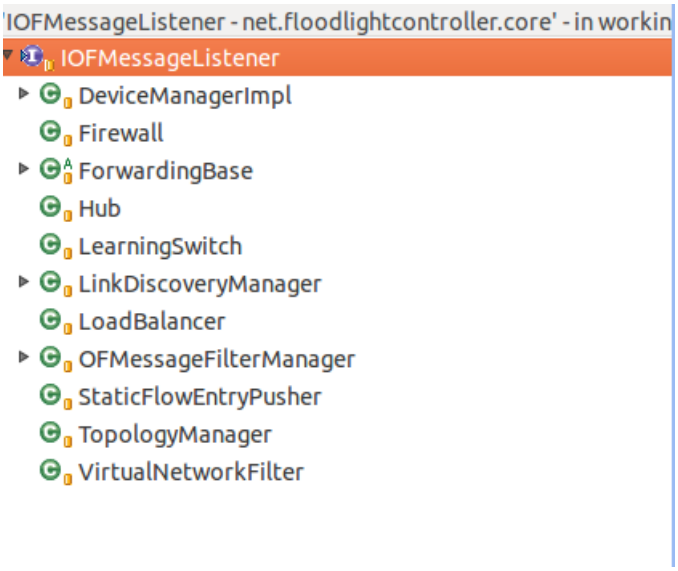


图 6 监听 IOFMessageListener 的模块

中打开 IOFMessageListener，通过 Type Hierechy 窗口观察，可得到图 6。从图上可以看到监听数据包进入的模块。

观察数据包的进入，即 PacketIn 函数。在 Floodlight 中，一个数据包进入被监听系统检测到并进行操作的流程如图 7 所示。

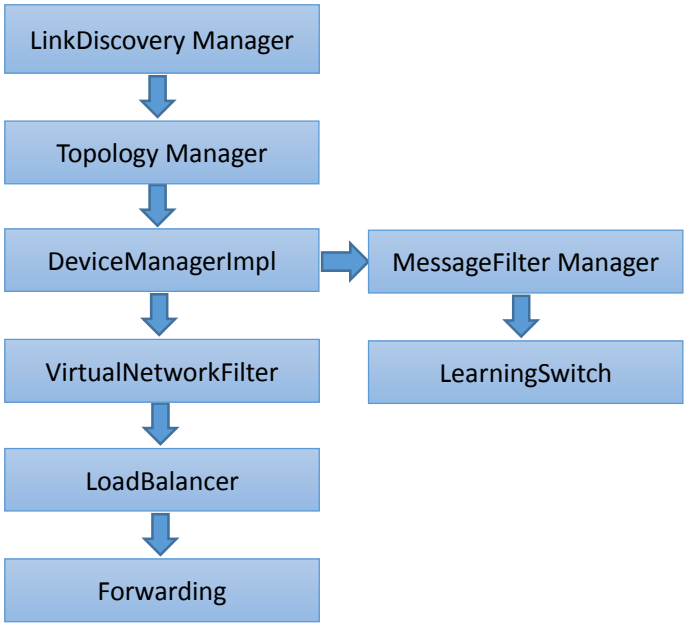


图 7 数据包进入的监听检测流程

我们添加的算法模块应用于代替 Forwarding 模块，所以在设计 PacketIn 的模块进入顺序时需要将添加的模块和 Forwarding 模块的优先级设置为一致，这样才能够保证不影响其他模块工作的情况下 Floodlight Controller 控制器正常运行。

5.4.2 模块创建

（1） 新建模块

添加模块步骤如下

1. 打开eclipse文件夹，找到src/main/java文件夹
2. 右键点击该文件夹，选择New/Class
3. 在Package中填入net.floodlightcontroller.scheduling
4. 在Name一栏填入Scheduling
5. 添加interface

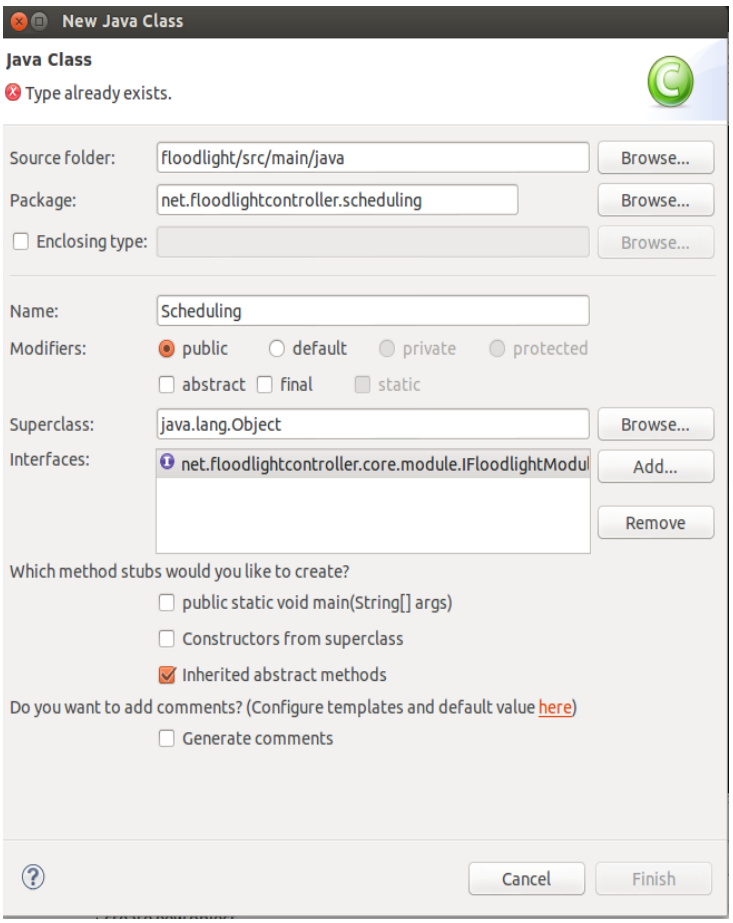


图 8 添加新的 JAVA 类

6. 选择IOFMessageListener和IFloodlightModule
7. 完成模块添加

如图 8 所示，在 src/main/java 中添加新的 JAVA 类，添加成功后，会得到如图 9 所示的新的 JAVA 类的 Scheduling 文件。

```java
package net.floodlightcontroller.scheduling;

import java.io.IOException;

@SuppressWarnings("unused")
public class Scheduling extends ForwardingBase implements IFloodlightModule {

    @Override
    public Collection<Class<? extends IFloodlightService>> getModuleServices() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Map<Class<? extends IFloodlightService>, IFloodlightService> getServiceImpls() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void init(FloodlightModuleContext context)
            throws FloodlightModuleException {
        // TODO Auto-generated method stub

    }
```

图 9 生成的 Scheduling 文件

（2）定义 log 变量记录该模块的运行情况，如图 10 所示

```java
protected static Logger log = LoggerFactory.getLogger(Forwarding.class);
```

图 10 log 变量的引入

（3） 模块依赖关系申明及初始化

如 5.2 中所述，我们需要使用 Device Manager 和 Packet Streamer 模块，其主要接口有：

```java
@Override
public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
    Collection<Class<? extends IFloodlightService>> l =
            new ArrayList<Class<? extends IFloodlightService>>();
    l.add(IFloodlightProviderService.class);
    l.add(IDeviceService.class);
    l.add(IRoutingService.class);
    l.add(ITopologyService.class);
    l.add(ICounterStoreService.class);
    return l;
}
```

图 11 依赖关系申明

IFloodlightService, IFloodlightProviderService, IDeviceService

IRoutingService, ITopologyService, ICouterStoreService

所以需如图 11 所示进行模块依赖性申明

在写完依赖性申明后，需要如图 12 所示继续填写模块装载 loading System 和初始化申明 Init。

```
public void init(FloodlightModuleContext context) throws FloodlightModuleException {
    super.init();
    this.floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
    this.deviceManager = context.getServiceImpl(IDeviceService.class);
    this.routingEngine = context.getServiceImpl(IRoutingService.class);
    this.topology = context.getServiceImpl(ITopologyService.class);
    this.counterStore = context.getServiceImpl(ICounterStoreService.class);
}

@Override
public void startUp(FloodlightModuleContext context) {
    super.startUp();
}
```

图 12 初始化申明

5.4.3 数据包监听模块

```
public Command receive(IOFSwitch sw, OFMessage msg,
                       FloodlightContext cntx) {
    switch (msg.getType()) {
        case PACKET_IN:
            IRoutingDecision decision = null;
            if (cntx != null)
                decision =
                        IRoutingDecision.rtStore.get(cntx,
                                                     IRoutingDecision.CONTEXT_DECISION);

            return this.processPacketInMessage(sw,
                                               (OFPacketIn) msg,
                                               decision,
                                               cntx);
        default:
            break;
    }
    return Command.CONTINUE;
}
```

图 13 继承监听模块后的具体实现

在我们添加算法模块时，需要使用 public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx)函数，如图 13 监听数据包的进入（ProcessPacketInMessage）。

5.4.4 数据包操作判断

首先，对数据包的操作进行判断，转发或者丢弃或者广播。此处可以使用 case 语句进行时说明，每一种行为在 case 语句之后单独说明。转发行为即我们的休眠调度算法。具体代码如图 14 所示。

```
        switch(decision.getRoutingAction()) {
        case NONE:
            // don't do anything
            return Command.CONTINUE;
        case FORWARD_OR_FLOOD:
        case FORWARD:
            doScheduler(sw, pi, cntx, false);
            return Command.CONTINUE;
        case MULTICAST:
            // treat as broadcast
            doFlood(sw, pi, cntx);
            return Command.CONTINUE;
        case DROP:
            doDropFlow(sw, pi, decision, cntx);
            return Command.CONTINUE;
        default:
            log.error("Unexpected decision made for this packet-in={}",
                    pi, decision.getRoutingAction());
            return Command.CONTINUE;
        }
    } else {
        if (log.isTraceEnabled()) {
            log.trace("No decision was made for PacketIn={}, forwarding",
                    pi);
        }

        if (eth.isBroadcast() || eth.isMulticast()) {
            // For now we treat multicast as broadcast
            doFlood(sw, pi, cntx);
        } else {
            doScheduler(sw, pi, cntx, false);
        }
    }
```

图 14 数据包操作判断

5.4.5 转发方法设计

我们的设计重点在于使用 4.3 节提出的最佳动态回复 BRD 算法达到绿色网络的休眠调度的目的，所以此处仅对 doScheduler 方法进行分析，丢包和广播等详细实现方法如 doFlood, doForwardFlow, doDropFlow 的代码见附录 1。

（1） 首先，我们需要申明doScheduler方法，并导入数据包的内容，如图

```
i=0;
OFPacketIn[] Pi = null;
Pi[i]=pi;
i++;
```

图 15 数据包的导入和存储

15所示。图中引入了private int整形变量i，用于计数数据包数量，并存于数组Pi中。

（2） 得到数据包之后，需要将源、目的地址等信息读取出来，通过forwardingBase模块继承的函数和IDeviceManager模块导入的函数来完成，之后将数据存储在数组中，并对各个数据包的数据进行排序，代码如下图16所示。

```
IDevice dstDevice =
        IDeviceService.fcStore.
        get(cntx, IDeviceService.CONTEXT_DST_DEVICE);
IDevice srcDevice =
        IDeviceService.fcStore.
        get(cntx, IDeviceService.CONTEXT_SRC_DEVICE);

SwitchPort[] srcDaps = srcDevice.getAttachmentPoints();
Arrays.sort(srcDaps, clusterIdComparator);
SwitchPort[] dstDaps = dstDevice.getAttachmentPoints();
Arrays.sort(dstDaps, clusterIdComparator);
```

图 16 数据包信息导入

（3） 导入完成之后，就是用第四章中4.3节给出的最佳动态回复BRD算法对数据包进行找路调度处理，算法部分的JAVA方法实现如下图17所示。

```
int NE=0;
Route[] route = null;

while(NE==0){
    Route[] routeAhead = route;
    for (int tmp=0;tmp<i;tmp++){
        SwitchPort srcDap = srcDaps[tmp];
        SwitchPort dstDap = dstDaps[tmp];


        route[tmp] =
                routingEngine.getRoute(srcDap.getSwitchDPID(),
                        (short)srcDap.getPort(),
                        dstDap.getSwitchDPID(),
                        (short)dstDap.getPort(), 0);

    }
    if (routeAhead==route)
        NE=1;
}

//After choosing paths for all packets, we forward them
for (int tmp=0;tmp<=i;tmp++){
    doFlood(sw, pi, cntx);
```

图 17 BRD 算法实现

（4） 当所有数据包的转发路径确定后，即达到了纳什均衡状态，此时使用log写日志的方式来反馈信息，表明此模块执行顺利，如图18所示。

```
if (log.isTraceEnabled()) {
    log.trace("Find the Nash Equilibrium State");
}
```

图 18 反馈运行报告

（5） 确定转发路径之后，利用底层模块提供的转发函数完成所有数据包的转发工作，如下图19所示。

```
Integer wildcard_hints = null;
IRoutingDecision decision = null;

//After choosing paths for all packets, we forward them
for (int tmp=0;tmp<=i;tmp++){

    wildcard_hints = decision.getWildcards();
    decision = IRoutingDecision.rtStore
            .get(cntx,
            IRoutingDecision.CONTEXT_DECISION);
    long cookie =
            AppCookie.makeCookie(FORWARDING_APP_ID, 0);

    pushRoute(route[tmp], match, wildcard_hints, Pi[tmp], sw.getId(), cookie,
            cntx, requestFlowRemovedNotifn, false,
            OFFlowMod.OFPFC_ADD);
}
```

图 19 数据包转发

5.5 模块注册和初始化

完成上述步骤后，Scheduling 模块基本已经完成了，最后需要告诉 Floodlight 在启动的时候装在此模块。首先，我们需要告诉 Loader 该模块存在。为此，我们需要在 src/main/resources/META-INF/services/net.floodlight.core.module.IFloodlightModule 中添加以下代码

net.floodlightcontroller.scheduling.Scheduling

然后，我们需要修改 Floodlight 整合文件，即 floodlightdefault.properties 在其中添

加如下代码

floodlight.modules = <leave the default list of modules in place>,

net.floodlightcontroller.mactracker.MACTracker

至此，本模块添加完毕。

六．总结

本设计题通过添加创新应用的形式进行了相应拓展，实现了理论分析得出的绿色网络的休眠调度算法，旨在当前的通信网络中大幅度节约能源，提高能源利用率。

通过本应用可以实现汇聚数据流，使得更多的交换机处于闲置状态的效果，在此基础上，将闲置状态的交换机调整至休眠状态或者关闭即可以节省大量的能耗。在下来的工作中，我们希望能够通过 H3C 厂家的设备，将我们的算法移植到实际的控制器和交换机上，并和 H3C 的技术人员一起实现交换机的休眠模块的编写，在实际设备上测试我们的理论模型的可行性。

七．参考文献

[1] Webb M. Smart 2020: Enabling the low carbon economy in the information age[J]. The Climate Group, London, 2008.

[2] Bianzino A, Chaudet C, Rossi D, et al. A Survey of Green Networking Research[J]. IEEE Communications Surveys & Tutorials, 2010(99): 1–18.

[3] 《中华工商时报》, 2008年12月14.

[4] M.Gupta, S.Singh. Greening of the Internet[A]. In Proc. ACM SIGCOMM 2003, Germany, 2003.[C]: 19–26.

[5] J. Baliga et al., Energy consumption in access networks, OFC, San Diego, 2008

[6] Garcia M, Garcia D, Garcia V, et al. Analysis and modeling of traffic on a hybrid fiber-coax network[J]. IEEE Journal on Selected Areas in Communications, 2004, 22(9): 1718–1730.

[7] Heller B, Seetharaman S, Mahadevan P, et al. ElasticTree: Saving energy in data center networks[A]. In: Proc. 7th USENIX Conf. Networked systems design and implementation[C], 2010.

[8] Mark Stemm and Randy H Katz, “Measuring and reducing energy consumption of network interfaces in hand-held devices,” IEICE Transactions on Communications, vol. E80-B, no. 8, pp. 1125–1131, Aug. 1997.

[9] OliverKasten,EnergyConsumption,http://www.inf.ethz.ch/~kasten/research/bathtub/energy_consumption.html, EldgenossischeTechnischeHochschule Zurich.

[10] W. Ye, J. Heidemann, and D. Estrin, “An energy-efficient MAC protocol for wireless sensor networks,” in Proc. IEEE INFOCOM 2002, pp. 1567–1576, 2002.

[11] A.P. Jardosh, K. Papagiannaki, E.M. Belding, K.C. Almeroth, G. Iannaccone and B. Vinnakota, Green WLANs: On-demand WLAN Infrastructures, Springer Mobile Networks and Applications, vol. 14, no. 6, pp. 798-814, 2009.

[12] IEEE Communication Magazine, special issues on Green Communication， November 2010, June 2011 and August 2011.

[13] M. Lin, A. Wierman, L. Andrew and E. Thereska, “Dynamic right-sizing for power-proportional data centers,” in Proc. IEEE INFOCOM, 2011.

[14] Mike Fratto, “Application Defined Networking: The Next Wave,” Network Computing, Aug. 23, 2012.

[15] Jain S, Kumar A, Mandal S, et al. B4: Experience with a globally-deployed software defined WAN[C]//Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM. ACM, 2013: 3-14.

[16]www.sdnap.com

[17]《深度解析 SDN》 张卫峰 电子工业出版社

[18]Software-Defined Networking: The New Norm for Networks

[19] http://www.projectfloodlight.org/floodlight/