分类号 密级

UDC

学 位 论 文

SDN中动态负载均衡与节能机制的设计与仿真实现

|  |  |  |  |
| --- | --- | --- | --- |
| 作 者 姓 名 ： | 胡书培 | | |
| 指 导 教 师 ： | 王兴伟 教授 | | |
|  | 东北大学计算机科学与工程学院 | | |
| 申请学位级别： | 硕士 | 学 科 类 别 ： | 工学 |
| 学科专业名称： | 计算机应用技术 | | |
| 论文提交日期： | 2017年12月 | 论文答辩日期： | 2017年12月 |
| 学位授予日期： | 2018年01月 | 答辩委员会主席： | 姚羽 |
| 评 阅 人 ： | 尹朝万 贾杰 |  | |

东 北 大 学

2017年12月

**A Thesis in Computer Application Technology**

**Design and Simulated Implementation of**

**Dynamic Load Balancing and Energy-Saving Mechanism in SDN**

By Hu Shupei

Supervisor: Professor Wang Xingwei

**Northeastern University**

**December 2017**

独创性声明

本人声明，所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：

日 期：

学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人同意东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

作者和导师同意网上交流的时间为作者获得学位后：

半年□ 一年□ 一年半□ 两年

学位论文作者签名： 导师签名：

签字日期： 签字日期：

# 摘 要

作为一种新型的网络范式，软件定义网络（Software-Defined Networking, SDN）实现了控制平面与数据平面相分离，解决了网络协议臃肿、网络创新困难等问题，提高了网络的灵活性和可扩展性。然而，随着SDN技术研究的深入，人们发现SDN仍然面临诸多挑战如负载均衡和节能问题。本文在对SDN中的负载均衡与节能技术进行调研之后，设计一种动态机制用以解决SDN面临的负载均衡与节能的问题。主要工作如下：

首先，本文提出SDN负载均衡与节能机制的框架。整个框架分为流量监测机制、路由和流调度机制、OpenFlow协议部分和基础设施部分。其中，流量监测机制负责监控网络实时的状态，实现数据流级别的流量测量；路由和流调度机制负责在对即将休眠或可能出现拥塞的链路进行流调度；OpenFlow协议负责控制平面和数据平面之间的交互；基础设施部分在转发数据的同时负责数据流路径快速地安装和更新。其次，本文设计一种基于网络整体流量和数据流速率变化的动态轮询算法，实现了以较小的开销获得流级别的流量测量。然后，本文提出一种基于链路偏好的随机路由算法和两种流调度算法以实现SDN的动态负载均衡与节能。在路由算法中，本文综合负载均衡与节能两个因素，将链路利用率映射为链路的偏好，根据链路的偏好对链路上的流量进行调控，进而使得数据流在路由阶段就能实现负载均衡与节能。这种映射关系随着网络的流量以及链路的利用率动态变化，体现路由机制良好的动态性。此外，本文分别设计非迭代式和迭代式的流调度算法，对网络中的拥塞链路和空闲链路进行流调度，进而达到均衡网络负载和节能的目的。针对局部链路出现的“假拥塞”和“链路状态频繁切换”问题，本文设计一种弹性缓冲机制。最后，针对数据流路径的安装和更新，设计不同的策略，以降低控制器的负载。

本文基于Ryu控制器和Mininet平台实现SDN中的动态负载均衡与节能机制，并进行仿真实验。实验结果表明，在网络能耗、链路平均利用率和链路利用率的标准差等多个指标上，本文设计的负载均衡与节能机制比对比算法更加有效可行。

**关键词：** 软件定义网络；负载均衡；节能；流调度

# Abstract

As a new networking paradigm, Software-Defined Networking (SDN) decouples the control plane from the data plane of network, decreasing the complexity of network, improving the power of network innovation, and enhancing the network flexibility and scalability. However, with the development of SDN, it still has many challenges especially the load balance and energy-saving. Therefore, a dynamic mechanism to solve the problem of load balancing and energy-saving in SDN proposed in this thesis. The main contents of this thesis are summarized as follows:

Firstly, the framework of load balancing and energy saving mechanism for SDN is proposed. The framework includes many function modules, including traffic monitoring, routing and flow scheduling, OpenFlow protocol and infrastructure. Among them, traffic monitoring aims at monitoring the real-time status, and achieving traffic measurement at the level of flow; routing and flow scheduling aim at scheduling the flows for the links that may enter into sleep mode or encounter congestion; OpenFlow protocol improvement aims at guaranteeing the communication between the control plane and the data plane; infrastructure provides the basis for data transmission, and at the same time aims at quickly installing and updating the data flow paths. Secondly, a dynamic polling algorithm based on the change of the whole network traffic and flow rate is designed for traffic monitoring. It could measure the traffic at the level of data flow with a low cost. Then, the thesis proposes a random routing algorithm based on link preference and two flow scheduling algorithms to achieve dynamic load balancing and energy saving for SDN. In the routing algorithm, with the consideration of both load balancing and energy saving, the link utilization is converted to link preference, according to which the traffic on link is controlled, so that load balancing and energy saving can be achieved at the stage of routing. This relation between link utilization and preferences can be adjusted dynamically to match network traffic or link utilization, reflecting the dynamic advantage of this mechanism. In addition, the iterative and non-iterative flow scheduling algorithms are designed. An elastic mechanism is also designed, aiming at fake congestion and frequent switch of link status appearing on local link. Finally, to reduce the controller load, different strategies are put format for data flow path installation and updating.

The simulation environment is constructed based on Ryu controller and Mininet. Then, the mechanism of dynamic load balancing and energy-saving is implemented. Experimental results show that the proposed load balancing and energy-saving mechanism is more feasible and effective compared with the benchmark mechanisms, in terms of network energy consumption, average link utilization and standard deviation of link utilization.

**Key words:** Software-Defined Networking; Load balancing; Energy-saving; Flow scheduling

目 录

[独创性声明 I](#_Toc501478287)

[摘 要 II](#_Toc501478288)

[Abstract III](#_Toc501478289)

[第1章 绪 论 1](#_Toc501478290)

[1.1 研究背景 1](#_Toc501478291)

[1.1.1 SDN概述 1](#_Toc501478292)

[1.1.2 SDN负载均衡机制 5](#_Toc501478293)

[1.1.3 SDN节能机制 6](#_Toc501478294)

[1.2 研究现状 7](#_Toc501478295)

[1.2.1 SDN负载均衡机制的现状分析 7](#_Toc501478296)

[1.2.2 SDN节能机制的现状分析 8](#_Toc501478297)

[1.3 课题来源 9](#_Toc501478298)

[1.4 本文主要内容及组织结构 10](#_Toc501478299)

[第2章 相关技术基础 11](#_Toc501478300)

[2.1 OpenFlow协议 11](#_Toc501478301)

[2.1.1 OpenFlow简介 11](#_Toc501478302)

[2.1.2 OpenFlow流表 13](#_Toc501478303)

[2.1.3 OpenFlow消息 15](#_Toc501478304)

[2.2 SDN中流量测量 16](#_Toc501478305)

[2.2.1 主动式测量 17](#_Toc501478306)

[2.2.2 被动式测量 18](#_Toc501478307)

[2.3 SDN中流调度算法 19](#_Toc501478308)

[2.3.1 支持负载均衡的流调度算法 19](#_Toc501478309)

[2.3.2 支持节能的流调度算法 20](#_Toc501478310)

[2.4 本章小结 20](#_Toc501478311)

[第3章 SDN中动态负载均衡与节能机制的设计 21](#_Toc501478312)

[3.1 总体设计 21](#_Toc501478313)

[3.1.1 系统框架 21](#_Toc501478314)

[3.1.2 网络模型设计 22](#_Toc501478315)

[3.2 流量监测机制设计 26](#_Toc501478316)

[3.2.1 动态时间间隔监测策略 26](#_Toc501478317)

[3.2.2 流量矩阵构建 27](#_Toc501478318)

[3.2.3 算法描述 28](#_Toc501478319)

[3.3 路由和流调度机制设计 31](#_Toc501478320)

[3.3.1 基于链路偏好的随机路由算法 31](#_Toc501478321)

[3.3.2 支持负载均衡与节能的流调度算法 34](#_Toc501478322)

[3.4 数据平面路径安装与更新策略设计 39](#_Toc501478323)

[3.4.1 路径流表安装策略 39](#_Toc501478324)

[3.4.2 路径流表更新策略 40](#_Toc501478325)

[3.5 本章小结 41](#_Toc501478326)

[第4章 仿真实现与性能评价 43](#_Toc501478327)

[4.1 仿真平台搭建 43](#_Toc501478328)

[4.1.1 实验环境 43](#_Toc501478329)

[4.1.2 实验拓扑 44](#_Toc501478330)

[4.2 仿真实现 46](#_Toc501478331)

[4.2.1 主要数据结构及函数实现 46](#_Toc501478332)

[4.2.2 实验设置 49](#_Toc501478333)

[4.2.3 参数确定 50](#_Toc501478334)

[4.3 性能评价 51](#_Toc501478335)

[4.3.1 评价指标 51](#_Toc501478336)

[4.3.2 对比机制 52](#_Toc501478337)

[4.3.3 性能测试与分析 52](#_Toc501478338)

[4.4 本章小结 60](#_Toc501478339)

[第5章 结束语 61](#_Toc501478340)

[参考文献 63](#_Toc501478341)

[致 谢 67](#_Toc501478342)

[攻读硕士学位期间发表的论文 69](#_Toc501478343)

# 第1章 绪 论

## 1.1 研究背景

本节将分别对SDN及SDN中的负载均衡和节能机制进行背景介绍。首先，在SDN概述中将介绍SDN历史背景、基本原理和特点，然后介绍SDN中负载均衡和节能机制的研究目的和意义。

### 1.1.1 SDN概述

需求驱动着技术的发展，当人类有了信息共享的需求时，计算机网络就诞生了，并且自计算机网络出现的那一刻起，整个世界都变得不一样了。无论当时的计算机科学家是否意识到这点，但是从那一刻起世界开始“加速”起来。尽管当时只有寥寥几个节点，但是它们构成的网络像一条条“高速公路”使各地的人们更加便捷地共享信息，这便是最初的计算机网络。

随着时代的变迁和科技的发展，越来越多的国家、企业和个人加入这个网络，网络的规模越来越大，功能已经从单纯的信息共享演变为购物和娱乐等，并且使用的人员也从高校、科研人士扩展至普通民众，但是唯一不变的却是20世纪70年代确立的TCP/IP体系结构。时代在发展，需求也在不断变化，虽然TCP/IP架构在当时非常完美，但是近年来随着新技术移动互联网、大数据、云计算的发展和应用，TCP/IP架构面临的问题越来越突出，例如：（1）协议越来越臃肿，网络难以管理维护；（2）设备制造厂商的技术封闭，新协议难以部署，网络演进困难；（3）网络安全问题日益严重，黑客事件频繁发生；（4）移动性问题越来越突出等。究其根本原因在于当前的网络仍是建立在TCP/IP架构之上，并且应对新问题的方式是在TCP/IP架构基础上进行“修修补补”，导致网络协议臃肿复杂，无法应对新的挑战。软件定义网络（Software Defined Networking, SDN）就是在这样的背景下诞生于“Clean-Slate Design for the Internet”项目。该项目的主旨为“另起炉灶”重新设计网络体系结构替代TCP/IP架构，避免渐进叠加“打补丁”式迭代更新，从根上去解决问题。

虽然SDN技术看似出现的很突然，但是SDN的理念——“可编程的网络”已经有20多年的历史。总的来说“可编程的网络”的发展历史主要分为3个阶段：1）90年代中期至2000年前后，网络中引进了可编程的功能；2）2001年至2007年，实现了控制平面与数据平面相分离，使得数据平面专注于转发；3）2007年至今，OpenFlow API和网络操作系统，通过开放的网络接口和开放的方式使得控制平面和数据平面相分离的可伸缩性和实践性广泛被接受[1]。具体地，自2008年Nick McKeon的论文“OpenFlow: Enabling Innovation in Campus Networks”发表以来，OpenFlow便逐渐引起了人们的广泛关注。作为控制平面和数据平面的交互协议，OpenFlow使得网络更加灵活，同时它也因此入选“信息领域未来十大技术之一”。2011年在谷歌、微软和FaceBook等公司倡导下开放网络基金组织（Open Networking Foundation, ONF）的成立，这标志着SDN标准化进程的开启，SDN的概念也首次被确立。2013年4月传统的设备芯片厂商Cisco、Juniper和IBM等成立了OpenDayLight开源项目，这代表着传统封闭的设备厂商对于开源技术的认可，它对SDN的发展具有里程碑式意义[2]。同年8月Google公司的B4[3]项目成功部署，引起了巨大的轰动。学术界也不甘落后，2014年12月斯坦福大学和加州伯克利分校共同创立了ON.Lab，并且推出了面向运营商业务场景的SDN开源操作系统ONOS，至此形成OpenDayLight和ONOS两大阵营[2]。此外，ONF成立至今，相继出台了一系列OpenFlow协议规范（目前最新版本为OpenFlow v1.6），使得协议内容逐渐丰富，功能越来越多样。随着OpenFlow协议的不断完善，ONF的成立和Google B4项目的成功应用，SDN作为一种新的网络范式，被越来越多的研究机构看好。

关于SDN，ONF给出的定义如下：“在SDN的体系结构中，数据平面和控制平面相分离，网络的智能和状态是逻辑集中式的，底层网络设备被抽象出来为上层提供服务[4-5]。”这个定义反应了SDN的四大特点：

（1）数据平面和控制平面相解耦。区别于传统的网络数据转发和控制集中在路由器等设备上，SDN将网络的控制功能从底层设备中分离出来放置于控制器中，而底层设备专注于转发，从而提高了数据转发效率。

（2）转发决策是基于流而不是目的。在TCP/IP架构的网络中数据包的转发是基于目的IP地址，目的IP一旦确定数据转发路径就确定了，通过IP协议数据包能够逐跳转发至目的主机。然而SDN具有对整个网络的控制功能，通过可编程接口，能轻易实现不同的路径计算算法，接着将路径以流表的形式下发至底层设备，最后底层设备通过流表项的匹配使得数据包可以通过不同的路径到达目的主机。

（3）逻辑集中式的控制。通过拓扑发现模块，控制器能够与管控范围内相连的网络设备进行交互，进而获取全局的网络视图。同时底层的网络设备作为一种资源受到控制器的统一管理分配。

（4）控制平面通过对底层网络设备的抽象赋予网络可编程的能力。底层的网络设备通过OpenFlow协议被抽象为SDN Datapath，并且作为一种资源为上层提供服务。控制器通过实现OpenFlow能够与底层设备进行通信，同时在OpenFlow的基础上将网络的控制交互功能封装为应用程序接口（Application Programming Interface, API）提供给应用层使用。应用层通过调用控制器提供的API使网络变得可编程。

ONF给出的SDN的基本架构的如图1.1所示。整体来看，分为三层两接口：应用层、控制层、基础设施层、应用层与控制层之间的北向接口、控制层与基础设施层之间的南向接口。北向接口负责应用层与控制层间的通信，目前没有统一标准。南向接口负责控制层与基础设施层间的通信，标准协议为OpenFlow。



图1.1 SDN体系结构[2]

Fig. 1.1 SDN architecture [2]

（1）基础设施层

基础设施层位于SDN架构的底层，主要包含各种网络设备，例如Open vSwitch、ofsoftswitch13、Indigo、OpenFaucet和Pantou。它将物理设备抽象为一个个逻辑设备，并且将它们作为一种资源提供给控制层使用。基础设施层的设备主要负责信息采集和数据转发，不具有控制功能。

（2）控制层

控制层包含一个或多个逻辑集中式的控制器，通过实现OpenFlow协议与基础设施层进行交互。同时，控制层需要借助于底层提供资源的抽象，对底层设备进行管理，实现对网络的控制功能（拓扑发现、路径计算、流表下发等）。在大型的SDN网络中，通常需要多个控制器，控制器之间通过东西向接口进行通信。除此以外，控制层将网络的控制功能封装为API提供给应用层，赋予网络可编程的能力，使得网络更加灵活。现有的开源控制器包含NOX/POX、Ryu、FloodLight和OpenDayLight等。

（3）应用层

应用层位于SDN架构最上层，它直接与SDN控制器交互而不用关心基础设施层设备的工作状态。通过对控制层提供的接口的组合调用，可以轻易实现多种应用，例如负载均衡、流量监测、访问控制以及能效网络等。

具有单控制器的SDN实际部署的例子如图1.2所示。控制器通过安全通道连接多个OpenFlow交换机，并通过OpenFlow协议对其进行管理。在SDN中，当一个数据流的数据包到达OpenFlow交换机时，交换机执行流表匹配。如果匹配成功则执行相应的动作，从对应的接口转发出去；否则，OpenFlow交换机提取数据包头部信息，并通过PacketIn消息发送到控制器，由控制器根据头部信息为其计算路径，并通过FlowMod消息下发流表。交换机根据收到FlowMod消息后安装相应的流表项，为后续的数据包匹配做准备。



图1.2 SDN网络拓扑

Fig. 1.2 SDN network topology

尽管SDN有着很多的优良的特性，但是它并不是万能的。随着研究的深入，人们发现它也面临着新的问题和挑战。概括如下：1）单点失效，这是由于其集中式的控制逻辑造成的；2）安全性，SDN中的安全性问题至关重要，北向接口、南向接口和控制器特别容易成为攻击者的目标；3）北向接口协议问题，北向接口协议标准化刚起步，没有标准的协议；4）可伸缩性，随着网络规模的增大，需要多个控制器协同地工作，如何保证多控制器的协调一致具有很大挑战性；5）与现有网络的兼容问题。凭借着逻辑集中式的控制和网络可编程的特点，SDN吸引了大批的研究者，当前的研究主要集中在流量工程、移动和无线、测量和监测、数据中心网络以及安全和可靠性等方面[6]。总之，目前SDN技术还处于发展阶段，也只有谷歌、微软等少数大公司有能力在数据中心部署，离真正的普及SDN还有很长的路要走。

### 1.1.2 SDN负载均衡机制

随着网络用户的增多及移动社交网络、云计算和大数据等技术的发展，网络中的流量呈爆炸式增长。这些增长的流量中，对时间敏感的视频流量占比高达86%[7]。从趋势上看，流量将持续增加并且似乎没有上限，例如在中国每年的购物狂欢节11月11日这天，中国最大的电商网站——天猫网将会承受来自全中国乃至全球每秒十几亿的点击流量。然而网络链路的带宽和设备的处理能力是有限的，部署新设备的代价是运营商或服务提供商不愿承担的。如何充分利用当前的设备，在资源有限的条件下满足用户的需求？如何降低数据流在网络中的传输时延、提高网络的吞吐率？这不仅是传统网络面临的问题更是SDN面临的问题。

负载均衡属于流量工程的一部分，它的主要目的是通过提高资源的利用率、网络的吞吐量以及降低传输时延等来优化网络性能。关于负载均衡的研究由来已久，例如在传统网络中ECMP（Equal-Cost Multi-Path）技术和多协议标签交换（Multi- Protocol Label Switching，MPLS）等技术已趋于成熟，但是它们不能直接应用于新的网络范式。首先，网络体系结构已经发生了改变，它们并没有利用SDN架构的优点，例如逻辑集中式控制、网络可编程等。其次，SDN中的设备已经与传统网络有了很大的区别，传统的网络设备控制与转发功能是紧耦合的关系，SDN将网络设备控制功能分离出来，使得底层只专注于数据转发。最后，相比于传统网络，基于IP地址和下一跳的路由，SDN的路径是可编程的。因此以SDN的视角来研究负载均衡是很有必要的。借助于以往技术的积累和SDN平台的特点，我们期望在SDN负载均衡的研究上取得更好的效果。

### 1.1.3 SDN节能机制

近年来，信息通信技术（Information and Communication Technology, ICT）产业的能耗问题日益突出，逐渐引起了科研人员的关注。据European Network of Excellence in Internet Science（EINS）的一项调查报告指出：2012年ICT产业全年消耗了全世界4.7%的电能[8-9]。另一份关于能耗的报告指出，ICT产业能耗到2020年底预计将占世界能耗的14%[9-11]。甚至连世界知名的物理学家霍金也在2017年腾讯WE大会上预言2600年电力消耗将让地球变成炽热的火球。ICT产业耗能巨大其根本原因在于设备的过量供应，导致设备利用率低下，网络的潜能没有得到有效发挥。此外，这也与运营商保守的思维模式相关，面对新技术，他们会保持观望的态度。当前的网络是以用户为中心，设计人员考虑更多的是“更低的时延”、“更高的吞吐量”，极少涉及到节能。例如，假设有两种设计方案都能达到效果，一种是通过增加网络设备，另一种是通过修改网络的架构设计更高效的算法。运营商可能更愿意去选择前者，因为它已经被证明过可行并且似乎更容易实现，这对设备和电能来说都是巨大的浪费。此外，网络中的流量具有很强的规律性，例如在白天（8:00-18:00）这段时间处于上班时间，人们忙于工作，所以网络流量适中。然而，在傍晚（18:00-22:00）这段时间属于娱乐时间，网络流量可能逐渐增高达到全天的峰值。最后在夜间到凌晨（22:00-8:00）这段时间人们多已经休息，所以网络流量达到全天最低点。因此在含有冗余路径的网络特别是数据中心网络中存在着大量非必要开启的设备。考虑到流量的规律性，如果夜间网络中的空闲链路一直处于工作状态，则可能造成资源的浪费。由于ICT产业能耗问题的凸显，使得研究设计节能路由、绿色网络变得十分必要。

在SDN中，借助其逻辑集中式的控制，降低了网络计算的复杂性，使网络设备的管理变得更加便捷。此外，利用其可编程的特性研发能耗感知或者绿色路由的网络应用变得更加简单。SDN控制器可以根据当前网络的负载对流量进行调度，优化网络资源配置。对于没有流量经过的节点可以使其进入休眠模式，同时流量较低的节点可以将数据流调度到其他活跃的链路，以此实现节能和优化网络资源配置[12]。最后，当前工业界和学术界分别提供了不同的开源平台OpenDayLight和ONOS，在此基础之上研究新的节能应用将极大地降低企业的部署成本。

## 1.2 研究现状

在对SDN负载均衡与节能相关文献调研的基础上，本文对当前有关SDN负载均衡与节能的研究现状进行了归纳和整理。

### 1.2.1 SDN负载均衡机制的现状分析

近年来关于SDN负载均衡机制已经开展了一些研究，通常的做法是对网络中的数据流进行调度。数据流调度算法可以说是负载均衡的核心，通过合理地调度网络中的数据流，从而优化网络的资源配置、提升网络性能，实现低时延、高吞吐量和避免网络拥塞的目的。从网络设备的角度考虑，SDN中负载均衡技术可以分为针对链路、服务器和SDN控制器的负载均衡；从流量调度的角度进行划分可以分为满足QoS流量调度、基于elephant流检测的流量调度和其他流量调度，具体如图1.3所示。



图1.3 基于SDN流量调度分类方案[13]

Fig. 1.3 Classification of traffic scheduling schemes based on SDN[13]

由于SDN的工作方式是当流表匹配失败，交换机发送PacketIn消息到控制器，因此控制器如果被攻击或者出现故障整个网络都将瘫痪，这使得控制器成为新的网络性能的瓶颈。在SDN中部署多个控制器能有效提高网络计算的效率，同时也能够有效防止单点失效问题，增加网络的健壮性，所以无论是出于性能考虑还是安全考虑，有关多控制器的部署和负载均衡研究都有重要意义。并且，OpenFlow v1.3及后续协议版已经增加了对多控制器的支持。文献[14]给出了当前SDN控制器负载均衡相关方案HyperFlow[15]、DIFANE[16]和BalanceFlow[17]的对比。

在传统网络中服务器的负载均衡主要是通过高性能代理来实现，在SDN中我们可以使用控制器来充当代理的角色。斯坦福大学提出的Anster\*X模型[18]就是基于Web服务器的负载均衡应用。在Anster\*X模型中控制器通过监测与服务器相连的交换机的状态确定每个服务器的响应时间，每次选择响应时间最短的服务器为新到达的请求提供服务。

在网络中，由于负载不均衡经常会导致局部链路拥塞，因此有关链路的负载均衡的研究，通常是寻找缓解局部链路拥塞的解决方案。传统的网络中基于数据流级别调度的ECMP是经典的解决方案。首先，它计算出多条备选路径，然后通过将数据流按照数据包的包头进行映射到不同的路径。然而当多个elephant流映射到相同的路径上仍然不可避免出现局部拥塞。另一种替代方案是实现数据包级别的调度，对一个数据流进行切分，到达目的主机在进行排序合并，但是这种方案不利于TCP流传输，因为无序到达的数据包将会导致TCP发送窗口的减小进而导致传输时延增加，并且引入的排序开销也不能忽视。在SDN中链路的负载均衡有如下几种解决方案。Hedera[19]是一种基于elephant流检测和调度解决方案。Mahout[20]同样是基于elephant流的检测，此外它充分利用了流表匹配优先级对elephant流进行调度。文献[21]提出的动态负载均衡路由算法是一种基于负载方差的最大流和次大流的调度，保证链路的负载均衡。LABERIO通过离线计算保存交换机之间的可能路径，使其在后期寻找替代路径时非常高效。

### 1.2.2 SDN节能机制的现状分析

关于SDN中的节能机制的研究目前多集中于数据中心的应用场景。对于数据中心网络来说能耗主要来自于以下两方面：1）网络设备，包括服务器、交换机、路由器和链路等；2）其他服务性设备，例如空调制冷系统、灯光系统等，其中网络设备占系统耗能的1/3[22]。节能机制的实现主要依据设备的负载以及流量周期性变化的特点使网络中的冗余设备休眠。本文的研究主要集中于可编程的OpenFlow交换机和链路设备。对于一个交换机，一个端口的屏蔽将大约有2-4w的节能效果。交换机从空载到满载大约有8%的能耗增加。表1.1展示了48个端口的交换机在不同配置下的能耗指标。其中，Switch A、B、C分别代表Cisco、ProCurve和Brocade不同类型的交换机[22-23]。

表1.1 交换机能耗指标[23]

Table 1.1 Switch energy consumption indicators [23]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ports Enabled | Ports Traffic | Switch A | Switch B | Switch C |
| None | 0 | 151w | 133w | 76w |
| All | 0 | 184w | 170w | 97w |
| All | 1Gbps | 195w | 175w | 102w |

SDN控制器拥有全局的网络视图，南向接口使其对低层次设备的管理更加灵活，这极大地降低了优化网络资源实现节能应用的复杂性。Elastic-Tree[23]专门设计了一个优化模块获得满足QoS需求的网络子集，从而将不使用的设备链路关闭达到节能效果，结果显示能耗降低了50%。多层流量工程（Multi Layer Traffic Engineering, MLTE）和GreCO是一种和Elastic-Tree相似的方法，也获得了很好的节能效果。EXR（Exclusive routing algorithm）展示了一种基于时间对数据流进行调度的路由算法，相比于其他节能算法EXR更加高效[24]。文献[25]描述了一种新的节能路由协议，它将网络流量路由到最合适的路径上满足用户的需求，其结果显示在夜晚可以节能45%。文献[22]针对时延敏感和不敏感的应用场景提出了两种路由算法降低数据中心能耗30-40%。相比于其他算法文献[26]的节能策略最为简单，控制器通过OpenFlow协议收集交换机信息，对负载低于阈值的交换机直接进行流调度，然后管理交换机。

根据1.2.1节和1.2.2节分析讨论，可以得出这样的结论：对于SDN链路级别的负载均衡和网络设备的节能机制，两者的共同点是都需要对数据流进行调度，不同点在于前者通常将负载最大链路上的最大流调到负载最小的链路上，而后者相反。由此可以看出在某种程度上这是两个互相冲突的目标。当前对于SDN中负载均衡与节能的研究现状主要是集中在其中某一方面，或者是两者同时研究但是在流调度上分两个阶段调度，前一个阶段优先满足其一，后一个阶段再满足另一个。但是这样可能会使第二阶段调度再次违反第一阶段的约束。本文的创新点基于其共同的特点，在最开始的路由计算阶段和后期的数据流调度阶段同时考虑负载均衡和节能两个目标。此外与前人研究的不同之处在于本文的拓扑不再局限于数据中心网络，可以广泛应用于任何含有冗余设备的网络中。

## 1.3 课题来源

本论文研究得到国家自然科学基金项目（61572123）和教育部——中国移动科研基金项目（MCM20160201）资助。

## 1.4 本文主要内容及组织结构

本文共分为五章，各章节内容如下：

第1章为绪论，主要讲述了SDN的简要概述，包括SDN的起源、发展、架构特点和面临的挑战。此外介绍了SDN中的负载均衡与节能的研究背景及现状分析。在此基础之上介绍了本文研究与前人的不同之处。

第2章为相关技术基础部分，主要介绍在本文的负载均衡与节能设计过程中用到的技术，主要包括：1）OpenFlow协议的基本原理；2）SDN流量测量技术的原理及分类；3）常用的SDN负载均衡调度算法；4）常用SDN节能机制调度算法。

第3章为本文的主要内容，主要介绍SDN中动态负载均衡与节能机制的整体设计及详细设计，主要包括网络监测模块、路由及调度模块、路径安装与更新模块。其中，网络监测模块负责监测全网的设备状态，构建全网的流量矩阵。路由模块在流量矩阵的基础上对新到达的数据流计算路径，调度模块通过对已存在的数据流进行调度优化对网络资源配置。路径安装与更新模块需要在路由与调度模块的指导下将控制器计算的路径安装或者更新到OpenFlow交换机中。

第4章为仿真实现与性能评价，主要介绍实验环境的搭建、参数设置及对实验结果的分析与评价。其中，实验环境搭建部分包含对控制器、网络拓扑及交换机的配置。此外还将介绍如何使用Mininet自动化模拟大量数据流。

第5章对本文工作进行了总结，并且对下一步的研究进行了展望。

# 第2章 相关技术基础

本章将对SDN中动态负载均衡与节能机制中用到的相关技术做简要介绍，其中，包括OpenFlow协议、SDN流量测量技术和SDN中流调度技术。了解相关技术将有助于加深对本文所设计的算法机制的理解。

## 2.1 OpenFlow协议

作为SDN南向接口的标准协议，OpenFlow有着极为重要的作用。本节将对OpenFlow协议做一个简略介绍。首先是对OpenFlow协议的起源发展、不同版本协议特性对比和处理流程的介绍。然后，介绍OpenFlow协议最为关键的流表组件，并详细对流表的各字段进行解释。最后介绍OpenFlow协议中的主要消息。

### 2.1.1 OpenFlow简介

OpenFlow最初起源于校园网络。起初McKeown发现在现有的硬件基础上在网络中创新对科研人员来说很困难。这主要是由于网络设备中的代码对研究人员不可见，无法动态改变设备的行为，造成了网络的僵化。于是McKeown等人将交换机的一些公共特性进行抽象作为一种与交换机交互的协议，这样使得设备厂商在不暴露他们代码的基础上能够通过软件灵活的管理和控制网络。

OpenFlow通过对底层资源的抽象赋予了控制器对资源进行管理和调度的能力，解决了网络僵化的问题，使得科研人员能在网络上灵活创新，也使得工业界对SDN架构和OpenFlow满怀期待。一方面ONF制定了众多OpenFlow协议版本规范等（目前最新的版本是v1.6），为SDN的标准化工作做出了巨大的贡献。另一方面，SDN的发展也逐渐引起了设备厂商的注意，并且ONF向传统的设备厂商敞开怀抱，2014年ONF已经有包括各主要厂商在内的95家以上企业[27]。随着OpenFlow协议的完善其支持的功能越来越多，在早期各主要版本对例如表2.1所示。其中，Per flow meter是为了实现简单的QoS操作，Bundles用于将一组OpenFlow消息打包作为一个处理，Eviction and Vacancy events 用于预警和改善交换机流表空间不足的问题，Egress确保所有的操作在出端口之前完成，Flow Entry Statistics Trigger使得控制器不再通过轮询获得流信息，它通过自动触发机制将流表项统计信息发送给控制器。

表2.1 主要的OpenFlow协议规范版本对比

Fig 2.1 Comparison among different OpenFlow specification versions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Specification | 1.0.0 | 1.1.0 | 1.2.0 | 1.3.0 | 1.4.0 | 1.5.0 |
| Multiple flow table | NO | YES | YES | YES | YES | YES |
| Group table | NO | YES | YES | YES | YES | YES |
| MPLS matching | NO | YES | YES | YES | YES | YES |
| IPv6 | NO | NO | YES | YES | YES | YES |
| Multiple controller | NO | NO | YES | YES | YES | YES |
| Per flow meter | NO | NO | NO | YES | YES | YES |
| Bundles | NO | NO | NO | NO | YES | YES |
| Eviction and Vacancy events | NO | NO | NO | NO | YES | YES |
| Flow monitoring | NO | NO | NO | NO | YES | YES |
| Egress Tables | NO | NO | NO | NO | NO | YES |
| Flow Entry Statistics Trigger | NO | NO | NO | NO | NO | YES |



图2.1 OpenFlow交换机主要组件[28]

Fig. 2.1 Main components of OpenFlow switch[28]

本文的设计和实验的仿真主要基于OpenFlow v1.3版本，因此接下介绍的OpenFlow特性也主要基于1.3版本，根据1.3版本规范，交换机的主要组件包含：安全通道、流表和组表，如图2.1所示。控制器和各个交换机通过安全通道相连，在连接建立时交换机和控制器通过交换私钥互相认证。认证后的双方通过OpenFlow协议定义的消息格式进行交互。流表主要用于数据流信息的匹配和处理。组表是一组流表项集合，用于对多个数据流统一处理。

在SDN中一个数据流的处理包括：数据流第一个数据包的匹配、控制器计算、路径安装和数据流后续分组的匹配和转发。第一个数据包到达OpenFlow交换机时，交换机首先查找流表进行匹配，如果匹配成功则执行相应的指令，否则按照table miss规定动作进行处理，通常是发送PacketIn消息到控制器，如果table miss不存在则丢弃。控制器收到PacketIn消息后根据数据包头部信息为数据流计算转发路径，并通过FlowMod消息和PacketOut消息安装路径即添加流表项。同一数据流后续的数据包到达该交换机时，交换机执行匹配，按照相应动作指令处理而不再发送PacketIn消息。

### 2.1.2 OpenFlow流表

通过OpenFlow对交换机特性的抽象，SDN控制器拥有了对底层交换机的灵活的控制能力，其中流表作为OpenFlow协议的主要组件，在数据流的处理中占据着非常重要的作用。每个OpenFlow流表包含着多个流表项，每个流表项的结构如表2.2所示。

表2.2 流表项结构[28]

Table 2.2 The structure of flow entry[28]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Match Field | Priority | Counters | Instructions | Timeouts | Cookie |

Match Field：匹配域包含入端口和数据包头部信息，常用的包括源端口/目的端口、源IP/目的IP和源MAC/目的MAC等。其中1.3版本支持的必要匹配项如表2.3所示。

Priority：优先级域是一个整数，默认值为0，值越大优先级越高，table-miss优先级最低，table-miss的默认动作为将数据包packet in到控制器。

Counters：计数器字段存储数据流的一些统计信息，如字节数和数据包个数。

Instructions：指令字段包含更新一些数据执行一些动作，例如数据包从哪个端口出去，进入哪个出端口队列等。

Timeouts：超时字段有两个重要的概念，硬超时（hard-timeout）和软超时（idle-timeout），默认值均为0，表示流表项永远不会发生超时事件。其中，硬超时是流表项的最大生命周期，如果一个流表项的存在时间大于等于硬超时的值，则该流表项就会被删除。软超时又称为空闲超时，如果数据流的流表项在连续空闲（未匹配）时间超过软超时的值则流表项将会被删除。

Cookie：默认为0，其主要作为数据流的流表项的标识，如果使用该字段则需要在流表项安装时设置为唯一值，同时需要设置cookie\_mask字段。在进行匹配时使用cookie\_mask字段与cookie字段进行按位与运算确定相应的流，特别是在更改和删除流表项时会使用此字段。

表2.3 OpenFlow v1.3支持的必要匹配项[28]

Table 2.3 Required match fields for OpenFlow v1.3[28]

|  |  |
| --- | --- |
| Field | Description |
| OXM\_OF\_IN\_PORT | Ingress port |
| OXM\_OF\_ETH\_DST | Ethernet source address |
| OXM\_OF\_ETH\_SRC | Ethernet destination address |
| OXM\_OF\_ETH\_TYPE | Ethernet type of the OpenFlow packet payload, after VLAN tags |
| OXM\_OF\_IP\_PROTO | IPv4 or IPv6 protocol number |
| OXM\_OF\_IPV4\_SRC | IPv4 source address |
| OXM\_OF\_IPV4\_DST | IPv4 destination address |
| OXM\_OF\_IPV6\_SRC | IPv6 source address |
| OXM\_OF\_IPV6\_DST | IPv6 destination address |
| OXM\_OF\_TCP\_SRC | TCP source port |
| OXM\_OF\_TCP\_DST | TCP destination port |
| OXM\_OF\_UDP\_SRC | UDP source port |
| OXM\_OF\_UDP\_DST | UDP destination port |



图2.2 OpenFlow匹配过程[28]

Fig. 2.2 The matching process of OpenFlow[28]

OpenFlow流表的匹配处理流程如图2.2所示。当数据包进入交换机后首先从table 0开始匹配，如果匹配成功，则更新计数器执行指令，直到所有的flow table都处理完成，最后执行相应动作集。否则，查看是否存在table miss流表项，如果存在则将数据包或者其头部信息通过PacketIn消息发送给SDN控制器，如果不存在则丢弃该数据包。

### 2.1.3 OpenFlow消息

在现有的OpenFlow协议中支持3类消息：控制器到交换机消息（Controller-to-switch）、异步消息（Asynchronous）和对称消息（Symmetric）[28]。其中控制器到交换机消息由控制器发起，主要是为了控制管理底层的交换机和查询设备状态。控制器根据消息不同可能需要交换机的响应如Feature request消息，也可能不需要响应如FlowMod消息；异步消息由交换机发起，用于通知交换机网络事件或者交换机状态的改变，特别是当一个数据流结束或者流表项发生超时事件时，在OpenFlow交换机中将会触发FlowRemove消息；对称消息可以由控制器或者交换机发起，主要用在建立连接时的握手和保持连接，如Echo消息还可以用来确认交换机和控制器的连接是否仍然有效。

下面详细介绍一些常用的OpenFlow消息：

Hello：对称消息，在连接建立阶段，控制器和交换机必须互相发送Hello握手消息，并进行互相认证，Hello消息头部将包含发送方支持的最高版本的OpenFlow协议的版本号，接受方在收到对方的Hello消息之后，将对比双方支持的协议版本号，然后选择低版本的协议进行交互。例如，控制器支持的最高版本OpenFlow协议为v1.5，交换机支持的最高版本为v1.3，那么双方协商将使用v1.3作为通讯协议。

Feature request：该消息为控制器到交换机消息，主要用于控制器向交换机询问交换机支持的特性或能力，交换机需要将自己的特性例如支持几个流表等信息通过Feature response消息告知控制器。

FlowMod：控制器到交换机消息，作为控制和管理OpenFlow交换机的一种手段，FlowMod可以说是SDN中最为重要的一种消息。控制器通过FlowMod消息增加、修改和删除流表项，可以更改交换机的状态，进而控制交换机的匹配和转发行为。FlowMod消息报文格式如图2.3所示。其中，cmd字段可以设置为增加、删除或是修改流表项，如果cmd为修改或删除时，则需要同时设置Cookie和Cookie\_mask字段以确定修改或删除的具体流表项。此外，flag字段用以确定数据流结束时是否需要发送FlowRemove消息到控制器，它可以降低控制器开销。

PacketIn：异步消息，交换机会在两种情况下向控制器发送PacketIn消息，一种是Packet in到控制器作为流表项的一种明确行为，一种是匹配了table miss。

PacketOut：控制器到交换机消息，用于控制器将原始数据发送到到交换机，或者告知交换机可以释放指定的缓冲区。

FlowRemove：异步消息，当交换机流表中的流表项发生超期删除时触发FlowRemove消息，它不仅标志着某一个数据流的结束，而且还将数据流的统计信息发送给控制器。



图2.3 FlowMod消息结构[29]

Fig. 2.3 The structure of FlowMod message[29]

## 2.2 SDN中流量测量

在SDN中，流量测量能够实现对网络的近似实时监控，是一项常用且基础的技术。很多SDN应用例如安全、负载均衡、节能等通常需要实现对网络的实时监测。测量的精准性和测量开销是衡量流量测量技术的评价指标。由于网络的状态会随时间发生变化，所以时效性非常重要。越精准的测量通常意味着实时性越好，开销越大。因此寻找一种开销更小和更精准的流量测量技术至关重要。OpenFlow协议提供了一个数据平面流级别的信息收集机制，当前众多SDN流量测量技术都用到此机制提供的抽象接口。依据控制器获取信息的方式划分为主动式和被动式。主动式通过控制器向交换机注入额外的流量来探测流或者端口的统计信息；被动式则通过交换机主动向控制器发送统计信息。此外，OpenFlow协议的制定者也在不断寻求更好的测量技术，几乎在每一个主要版本的Release notes中都能找到关于流量统计、数据流监测等相关方面的更新。

当前关于流量测量技术，文献[13-14]已经做了细致精炼的总结工作，特别是文献[13]在文献[14]的基础上引入OpenNetMon机制的描述，但是文献[13]的在对文献[14]其他机制的转述上并不准确，本文在两者基础之上概括如表2.4所示。

表2.4 SDN中流量测量技术

Table 2.4 SDN traffic measurement technologies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测量技术 | 方法 | 注入流量 | 机制描述 | 优点 |
| PayLess[30] | 主动查询 | YES | 根据流的变化自适应轮询数据流 | 短周期：高精确性高开销  长周期：低准确性低开销 |
| OpenTM[31] | 主动查询 | YES | 周期性轮询每个活跃流经过的交换机 | 高准确性高开销 |
| OpenSketch[32] | 主动查询 | YES | 使用通配规则监测聚合流的信息以降低开销 | 低内存消耗高精确性 |
| OpenNetMon[33] | 主动查询 | YES | 自适应轮询每个流的边缘交换机 | 细粒度的参数测量 |
| FlowSense[34] | 被动推送 | NO | 利用PacketIn和FlowRemove消息建立监测点进行计算 | 与轮询相比有较高准确性较低开销 |
| OpenSample[35] | 被动推送 | NO | 使用sFlow采样方法和TCP序列号降低开销检测大象流 | 低时延测量网络负载和大象流（Elephant Flows） |
| MicroTE[36] | 被动推送 | NO | 使用服务器监控降低控制器处理开销 | 低网络开销 |

### 2.2.1 主动式测量

在主动式测量中，控制器通过询问交换机获取流的统计信息，交互消息由控制器率先发起，每次控制器需要收集交换机信息时，就主动向交换机发送Flow StatsRequest/Port StatsRequest消息，交换机只有在收到StatsRequest消息时才会将流或者端口的统计信息以Flow StatsResponse消息告知控制器。在测量时，控制器通过发送StatsRequest消息向网络注入流量，这部分流量会占用交换机到控制器的安全通道带宽，而测量开销也主要由这部分流量组成。

下面将展示了一些主动式查询的方法。

（1）Periodic polling：通过周期性调用OpenFlow提供的抽象接口，SDN控制器能够轻易获得数据流或者端口的统计信息。但是，在周期性轮询算法中需要注意两个问题：轮询周期和轮询对象的选择。越短的轮询周期意味着更高测量开销和更精确的测量结果；越长的轮询周期意味着更低的测量开销和较低的测量精确性。同样如果轮询数据流经过的所有交换机意味着有更高的开销和更精准的测量，轮询数据流经过的边缘交换机则意味着低开销和较低精确性。

（2）PayLess：作为一种SDN流量监测框架，PayLess提供了一组灵活的RESTFul API接口，实现了针对不同聚合级别不同需求等级上的流量信息采集。针对不同的应用它可以提供不同的监测指标，例如性能指标（时延、抖动、吞吐量、丢包率等）、安全性指标（防火墙预警、接入控制验证等）。同时，PayLess除了提供不同的信息聚合级别（流级别、流表级别、端口级别、用户级别等）以外，它还关注测量的精确性、长期性和测量开销。在轮询算法上Payless也提供了3种级别（实时、中级、低级和自定义级别），不同的级别对应着不同的测量开销。最后，Payless提供了一种流级别自适应的信息采集算法，该算法能够保证以低的测量开销获得高度精确的测量结果。在自适应的流信息采集算法中，PayLess主要根据相邻两次流传输的数据量动态改变流的轮询周期降低测量开销。

（3）OpenTM：利用OpenFlow交换机的内建特性，OpenTM流量测量技术通过准确测量两个交换机节点之间的流量构建流量估计矩阵。此外尝试了不同的算法查询不同的交换机以平衡交换机的负载和测量的精确性。

（4）OpenNetMon：它是一种根据流的速率动态调整监测时间间隔的方法，稳定的数据流增加轮询时间间隔，剧烈变化的数据流减小轮询时间间隔。同时，OpenNetMon对比了多种查询交换机选择算法，结果表明轮询流路径上最后一个交换机有着高精确性和较低的开销。此外，OpenNetMon关注时延和丢包率，因此它在选择查询交换机时采用查询流路径上首尾交换机。

### 2.2.2 被动式测量

被动式测量，通常采取避免向网络中注入探测流量，由交换机主动向控制器发送有关数据流的统计信息。它不会影响控制器与交换机之间的安全通道因此也不会影响网络的性能。

FlowSense：该机制利用每个流的PacketIn消息和FlowRemove消息在控制器中建立监测时间点，每当新的时间点建立时更新链路的占用带宽。如图2.4所示。假设数据流的到达时间分别为，其结束时间分别为。根据FlowSense的算法，控制器在时刻分别建立监测点，时刻当FlowRemove消息到达控制器，可以计算出的带宽，那么时间间隔内链路的带宽为的带宽。同理当时刻结束时，FlowRemove消息将能计算出的带宽，并且记录。但是由于未结束，故无法计算时间段内带宽，直到时刻触发了Flow remove消息。



图2.4 FlowSense算法场景[34]

Fig. 2.4 The scenario of FlowSense algorithm[34]

本文的研究关注的是SDN中链路的负载均衡与节能，故在流量测量时需要构建全局的流量矩阵。同时基于OpenNetMon，我们采用动态自适应算法查询流路径上最后一个交换机的方式收集基于流级别的信息。此外，在计算链路带宽借鉴FlowSense的方法。

## 2.3 SDN中流调度算法

在众多的SDN数据流调度算法中，根据不同的目的采取的调度策略不同。如果考虑网络链路的负载均衡则会通过调度数据流使得经过每条链路的负载尽可能平衡，如果考虑节能则会尽可能将数据流从负载最轻的交换机或链路上调走，然后关闭或休眠相应的设备。但是，无论以何种目的进行调度必须确保存在备选路径，这是进行调度的前提。此外，选择调度哪些数据流也是调度需要考虑的问题。图1.3中列出一些基于大象流的调度，这其实是一种悲观的做法。乐观的做法是选择当前链路上最小的流调度。但是基于最小流的调度可能在一次调度之后并不能满足要求，所以会调度多个数据流，导致开销要比悲观的做法大得多。

### 2.3.1 支持负载均衡的流调度算法

当前对于负载均衡的流调度机制分类如图1.3所示，下面介绍几种经典负载均衡的流调度算法。

（1）Round Robin调度：对于具有相同源和目的节点的数据流，将多条备选路径以Round Robin顺序从中选择一条路径进行调度。

（2）基于路径负载的流调度：每次在调度前计算各条备选路径的负载，选择负载最轻的路径进行调度。这里的负载可以是链路的带宽或者链路利用率。

（3）基于ECMP流调度：每次选择的路径是通过将数据流的头部信息进行Hash，根据Hash值为数据流选择相同代价的备选路径作为调度的路径。

（4）基于LABERIO流调度：是一种局部链路的调度策略。对于网络负载的标准差超过阈值，LABERIO将寻找网络中负载最高的链路，然后采取悲观的做法将该链路上的最大流调走。区别于其他几种算法，它不是使用备选路径替换整个路径而是计算一段可以替换负载最大链路的子路径，直到网络负载的标准差小于阈值。

### 2.3.2 支持节能的流调度算法

支持节能的流调度可以分为两类：数据流聚合的调度、基于大象流的调度。

（1）数据流聚合的调度：作为最为常见的一种方法，它通过将利用率低的链路或设备上的数据流调度到其他路径上，然后使空闲设备进入休眠状态，进而达到节能的目的。同时，这类流调度算法通常会采用多路径评价的模型，每次从多个调度路径中选择“最优”的，例如，每次选择引入能耗最少的路径等。但是，由于聚合数据流调度算法有两个弊端：1）无法避免局部链路的拥塞等问题；2）一定程度上降低了网络的吞吐量。因此，如果能解决好这两个问题，聚合数据流的调度将非常有前景。

（2）基于大象流的调度：例如Willow[37]，通常情况下优先对大象流进行路由和流调度，其次对老鼠流（Mice Flows）调度。基于大象流的调度并不对设备进行休眠，其节能效果差，但是网络吞吐量不受影响。

## 2.4 本章小结

本章主要介绍了SDN中动态负载均衡与节能机制设计相关的技术基础。首先简要介绍了OpenFlow协议的发展历史及其基本原理，其次介绍了流表组件和几种主要的OpenFlow消息。然后对SDN中的流量测量技术做了一个简要概括，并讨论了现有的技术中降低测量开销的方法。最后介绍了在SDN中常用的流调度算法。

# 第3章 SDN中动态负载均衡与节能机制的设计

本章将首先介绍SDN中动态负载均衡与节能机制的总体设计，然后将分别对系统框架中的每个模块进行详细介绍，系统模块主要包括流量监测机制、基于链路偏好的随机路由算法、两种不同的支持负载均衡与节能的流调度算法和路径安装更新策略。

## 3.1 总体设计

本节首先介绍所设计机制的整体系统架构，从整体上对负载均衡与节能机制的处理流程有一个直观的认识，然后对网络的节点和链路模型进行建模。

### 3.1.1 系统框架

为了实现网络的负载均衡与节能机制，本文设计了几个重要的功能模块。首先，控制器需要利用其逻辑集中的控制结构得到网络的全局视图，因此，需要对网络进行实时流量监测。其次，由于网络中链路的负载并不是一开始就不均衡，所以本文设计了一个路由模块用于对正常到达的数据流进行路由计算。最后，当流量监测机制发出有关网络链路负载不均衡、链路拥塞等预警信息后，控制器需要能够对相关的数据流实现近似实时的调度。

本机制设计的总体框架如图3.1所示，整个系统分为四部分：流量监测机制、动态负载均衡与节能路由和流调度机制（简称路由和流调度机制）、OpenFlow协议部分和基础设施部分。

流量监测机制首先利用OpenFlow协议根据设定的自适应轮询算法向基础设施的交换机发出Query请求，然后根据接收到的Reply消息对每个数据流进行速率监测，同时它将与路由和流调度机制进行交互。

路由和流调度机制首先根据流速率计算网络链路利用率并且构建链路利用率矩阵，然后再依据偏好计算方法将利用率矩阵转化为链路的偏好矩阵。偏好矩阵同时兼顾了负载均衡与节能的目标，将作为路由与流调度机制基础数据。当控制器收到PacketIn消息后，路由机制将为数据流提供合理的初始路径，并将新的路径通过OpenFlow协议安全通道安装到交换机上。同样当数据流被调度时，控制器将通过流调度模块为数据流计算新的转发路径，并通过OpenFlow协议更新到交换机。



图3.1系统框架图

Fig. 3.1 The system framework

OpenFlow协议作为数据平面与控制平面交互的中介，不但需要将每个数据流路径的添加、更新真实地反映到基础设施的交换机上，而且需要作为控制器收集数据流信息的通道。

基础设施部分主要是指交换机等网络设备。基础设施模块需要维护网络的拓扑，对数据流的信息做统计并存储在流表中。此外，它还需要及时响应控制器各种OpenFlow请求消息，将流表中未匹配的数据流Packet in到控制器，以及定时向控制器发送Echo消息确保与控制器间连接正常。

### 3.1.2 网络模型设计

SDN控制器可以通过OpenFlow协议对底层交换机进行管理，这使得SDN中通过有效管理网络设备实现节能成为可能。在本文的设计中，网络的状态依据网络流量动态地随时间变化，本文将网络建模为一个带权有向图。为降低测量开销，流量监测机制只需要获取某个时刻的网络快照，因此本机制采用表示在时刻网络的拓扑。其中，为所有网络节点集合，为网络中有向边集合。

（1）节点模型

每个节点都有两种状态开启或是休眠，本文将节点建模为一个三元组。其中，用来标识一个节点，表示网络节点的状态，表示从此节点出发到相邻节点的边的集合。，当时，节点处于休眠状态；否则，节点处于开启状态。

对于每个节点当与其相连的每条边都处于休眠状态时，节点将被控制器休眠，其能耗为固定常数；反之，当与其相连的有向边都处在开启状态时，节点也处于开启状态，其能耗为，并且。依据文献[38]交换机节点能耗与其开启的端口数和端口利用率相关。本文将交换机节点端口分为两种：特殊端口和正常端口。其中，特殊端口包含主机与接入交换机相连的端口和交换机与控制器节点相连的端口；而正常端口是与其他交换机节点有链路相连的端口。因为特殊端口需要时刻监听网络中的消息，所以这类端口不会被休眠。正常端口则会随着网络状态的变化进行休眠或者开启。同时本文假定特殊端口的能耗为端口能耗的最大值；正常端口能耗包括固定能耗和动态能耗，其中动态能耗部分与端口利用率成正比。因此，休眠状态的节点能耗和开启状态下节点能耗计算分别如公式（3.1）和（3.2）所示。

 （3.1）

 （3.2）

 （3.3）

 （3.4）

其中，表示休眠节点的固定能耗，；表示开启状态下节点的基础能耗，。和分别为特殊端口与正常端口的数目，为休眠端口的固定能耗，为处于开启状态端口的能耗。

和计算分别如公式（3.3）和（3.4）所示。其中，为休眠状态端口固定能耗，；为端口在开启状态下最大能耗，；为端口利用率，。

根据公式（3.2）~（3.4）得到开启状态下节点能耗，计算方法如公式（3.5）。

 （3.5）

本文将节点端口能耗归结为有向边的能耗，简化后开启状态下节点能耗计算如公式（3.6）所示。

 （3.6）

（2）链路模型

由于链路是全双工的工作方式，因此本文将一条真实的网络链路抽象为两条反向的逻辑链路，每条逻辑链路对应一条有向边，如图3.2所示。每一条有向边用表示。其中为边的标识，代表边的出节点，代表边的入节点，代表出端口，代表入端口，为边的状态，并且，分别表示边的休眠和开启状态。为边的占用带宽，为边的剩余带宽，为边的带宽容量，为边的带宽利用率，为边的偏好权重。和是为了防止边被频繁休眠或唤醒而设置的缓冲生命值，为边带宽利用率低于阈值的生命值，为边带宽利用率高于阈值的生命值，为经过这条边的所有数据流集合。



图3.2 链路模型

Fig. 3.2 The link model

一条链路的活动与否与其两端的端口相关。为简化问题，本文将两端端口状态与逻辑链路的状态绑定，两个端口的状态必须一致。由于一条真实链路代表了两条反向的逻辑链路，因此真实链路被休眠的前提条件是当且仅当相关的两条逻辑链路均已经被休眠。在通常情况下当边的带宽利用率为0就可以对其休眠，但是在实际的网络中由于网络中的流量随时间动态变化，某时刻可能网络拓扑中的一条边并没有流量经过，但是由于处于流量高峰时期在后续的数据流可能会间断性经过该边，进而造成有向边在休眠和开启状态之间频繁切换，所以即使一条边没有数据流经过也不能马上休眠。

为了避免一条边被频繁的休眠或者唤醒，本文为每条边设置了缓冲生命值，其作用与流表项中的空闲超时相似。当一条有向边利用率低于某阈值时，其逐渐递减；反之，当其利用率一旦超过阈值，则其将恢复到初始值。当且仅当两条反向有向边同时小于0时，两条边同时被休眠，即链路被休眠。当一条链路接收到控制器下发的FlowMod消息时，链路将被唤醒，即与链路相关的两条有向边开启。前文提到将端口与边进行关联，每条有向边的能耗与其出端口相关联，如图3.2所示的能耗为的能耗，的能耗为的能耗。假设休眠的边的能耗为，开启状态的边的能耗为，则和的计算分别如公式（3.7）和（3.8）所示。

 （3.7）

 （3.8）

（3）网络拓扑模型

由于节点和链路存在休眠和开启两个状态，因此本文对网络定义了两种拓扑模型：活动拓扑和全局拓扑。

活动拓扑：网络中所有处于开启状态的节点和有向边构成的网络拓扑。由于每个节点和每条有向边都处于开启状态，所以在活动拓扑中进行路由和流调度时不会增加网络能耗。

全局拓扑：网络中所有的节点和有向边构成的网络拓扑。在全局拓扑中节点和链路存在两种状态，因此在进行路由和流调度时可能会涉及到节点或有向边状态从休眠状态切换到开启状态，同时增加网络的能耗。

在本文的能耗模型中，网络能耗不包含控制器能耗，只考虑网络中所有交换机节点能耗，根据文献[38]，节点能耗与交换机固定能耗和端口配置相关。但是本文将正常端口能耗与交换机固定能耗分开，正常端口关联到网络拓扑中的一条有向边，网络能耗计算如公式（3.9）所示。

 （3.9）

其中，和为网络中节点数目和有向边的数目，和分别指第个节点的能耗和第条有向边的能耗。当第个节点处于休眠状态，计算如公式（3.1）所示，否则，计算如公式（3.6）所示。同理，在不同状态下，的计算分别如公式（3.7）和（3.8）所示。

（4）数据流模型

为了方便控制器对网络中数据流的监测、管理以及调度，需要在控制器中保存数据流的对象，包含对数据流详细信息的描述。针对数据流对象本文用如下元组结构描述。其中，用以标识一个数据流，和分别代表数据流的源地址和目的地址，为数据流的速率，为数据流从开始到现在总共传送的字节数，为当前时间戳，为轮询时间间隔，为数据流经过的最后一个交换机节点，为数据流在流表中的匹配项，为数据流经过的一系列交换机节点的有序序列，为数据流经过的有向边的集合。的取值与监测机制的轮询算法相关，当轮询算法为周期性轮询时，其值为常数。当轮询算法为自适应轮询则其值动态变化。

## 3.2 流量监测机制设计

流量监测机制时刻监测网络的状态。虽然它在整个负载均衡与节能的调度机制中处于基础地位，但是其重要性不容忽视。因为后续的路由及调度算法所做出的决策都依据其测量结果，所以测量结果的精确与否直接决定着本机制的性能。精确的测量结果确实可以得到，只要控制器频繁发送探测消息询问交换机，这些探测消息通常称为测量开销。但是在流量高峰时期控制器通常忙于处理新的PacketIn消息，并且控制器与交换机的带宽可能会被大量的PacketIn、FlowMod和FlowRemove消息占据，通常对这些消息的响应会直接反映到SDN的性能上。因此，相对于网络的性能，监测机制不得不舍弃一些精确性，使得控制器较少的被探测消息“打扰”，确保发送更少的探测消息以减少对控制通道带宽的占用。

### 3.2.1 动态时间间隔监测策略

在现有的监测体系中周期性轮询算法可以说是最简单有效的算法，它每隔一个固定的周期就向交换机发送流统计数据的请求消息，监测机制收到交换机响应后更新网络的状态。如果想要精确的测量结果，可以将周期设置的更短一些，如果想要开销更小，则周期设置的长一些。然而，周期性轮询算法并没有考虑到网络中数据流量的规律性，因为网络并不是一直处于繁忙状态。

本文结合OpenNetMon的方法设计了一种动态时间间隔监测方法，它能够根据网络中数据流量的大小和每个数据流速率的变化动态改变轮询的时间间隔从而减小测量开销。

动态时间间隔策略主要采用“加法增大乘法减小”的策略调整流的轮询时间间隔。首先，最小的轮询时间间隔由当前的网络流量决定，网络流量越大，其值越大。其次，当数据流到达网络时，其第一次轮询时间间隔为。最后在后续的时间间隔更新策略中，后续的时间间隔不仅与数据流速率变化相关还与上一次的时间间隔相关。对于每个数据流来说，根据速率的变化，将其分为三种情况。当时，轮询间隔线性增大；当时，轮询间隔保持不变；当时，轮询间隔乘法减小，并且其减小程度与流速率变化的剧烈程度有关。动态时间间隔的计算如公式（3.10）所示，其中和为设定的阈值常数，且。

 （3.10）

 （3.11）

其中，作为数据流初始时间间隔为一个正整数，为轮询时间间隔的最大值，表示向下取整，的计算如公式（3.11）所示。在公式（3.11）中，为数据流在当前时刻的速率，为数据流上一次测量的速率，为大于0的实数，避免分母为0。

### 3.2.2 流量矩阵构建

（1）数据流速率计算

本机制主要是通过对数据流的路由调度管理实现网络的负载均衡与节能，其对网络的监测是以数据流为粒度。同时，在OpenFlow流表项中只提供数据流的传输时间、传输的字节数和传输数据包个数，并不直接提供数据流速率。根据3.2.1节的动态时间间隔策略，本文可以测出相邻两次轮询的流的数据信息。依据两次相邻的测量结果，我们对数据流的计算如公式（3.12）所示。

 （3.12）

其中，为当前数据流的发送的总字节数；为上一次数据流测得的总字节数；为当前时间戳，精确到毫秒；为上一次测量的时间戳，也精确到毫秒。，因为不仅包含还包括控制器的处理时间以及Query/Reply消息在交换机与控制器之间传输的时间。

（2）有向边上带宽计算

表示从节点到节点的有向边，上的占用带宽等于经过该边的所有数据流带宽之和，设其带宽为，则计算如公式（3.13）所示。

 （3.13）

其中，为经过的第个数据流，为当前网络经过的数据流总数。上剩余带宽计算如公式（3.14）所示。

 （3.14）

其中，和分别表示有向边的带宽和带宽容量。上带宽利用率的计算如公式（3.15）所示。

 （3.15）

由公式（3.14）和（3.15）可得网络的剩余带宽矩阵和带宽利用率矩阵，分别如公式（3.16）和（3.17）所示。

 （3.16）

 （3.17）

其中，为网络中所有节点的数目，当节点与节点之间没有边连接时，。

### 3.2.3 算法描述

监测机制需要同时开启两个处理线程，一个用于接受处理OpenFlow消息如算法3.1描述所示，另一个用于发送轮询消息如算法3.2描述所示。特别地，必须由两个算法协作，才能完成对网络的监测。

算法3.1将OpenFlow消息分类进行处理。如果收到的消息是PacketIn消息，则监测机制对数据流进行一系列登记，初始化其监测时间点，并将其加入监测队列；如果收到的是StatsResponse消息，监测机制根据消息中的数据计算数据流的速率，并对一系列与该数据流相关的有向边进行更新，同时更新网络状态矩阵，最后更新数据流的检查点，并将其加入监测队列中；如果是FlowRemove消息，则更新相关的有向边，更新网络状态矩阵，同时将其监测点从中移除，最后将整个数据流从中移除。算法3.1的伪代码描述如下所示。

|  |
| --- |
| **算法3.1 OpenFlow消息处理** |
| **输入：**OpenFlow消息 |
| **输出：**更新、、 |

BEGIN

|  |
| --- |
| 1. Initialize the monitoring set  and ; |
| 2. Initialize the matrix to beequal the latest value; |
| 3. FOR each message that the controller receives, DO |
| 4. IF  is PacketIn message, THEN |
| 5. IF the new flow  THEN |
| 6. Do other thing, get the current time，set the ; |
| 7. Set the check point for， ; |
| 8. Add the check point to ; |
| 9. END IF |
| 10. ELSE IF  is Flow StatsResponse message, THEN; |
| 11. Update the flow rate of; |
| 12. Update the flow’s ; |
| 13. Update  of edges that are in the flow’s edgeset; |
| 14. Update **;** |
| 15. Break; |
| 16. ELSE IF  is Flow FlowRemove message, THEN |
| 17. Update  of edges that are in the flow’s edgeset; |
| 18. Update **;** |
| 19. Remove the  from the ; |
| 20. Remove the flow  from the ; |
| 21. Break; |
| 22. END IF |
| 23. END FOR |
| END |

在算法3.1中，第4~9行为PacketIn消息的处理，主要涉及到路径计算和数据流登记，路径计算的过程将在后续章节给出。第10~15行为测量消息的处理，主要涉及带宽计算、轮询间隔更新和流量矩阵构建。第16~22行为FlowRemove消息处理，涉及流表信息的清理和网络流量矩阵的更新等。

算法3.2为数据流轮询机制，首先初始化全局时钟，每隔一秒其值加一，然后对中的监测点比对，如果监测点的时间与当前时间相同，则对该检测点的数据流经过的最后一个交换机发送StatsRequest消息。最后为该数据流设置新的监测点，将其加入，并将原有的监测点从中删除。算法3.2的伪代码描述如下。

|  |
| --- |
| **算法3.2 轮询机制** |
| **输入：**监测点集合 |
| **输出：**更新 |

BEGIN

|  |
| --- |
| 1. Initialize the check time ; |
| 2. WHILE  is not , DO |
| 3. FOR each check point, DO |
| 4. IF, THEN |
| 5. Send the StatsRequest message to the flow  last switch; |
| 6. Remove  from ; |
| 7. END IF |
| 8. END FOR |
| 9. Update，; |
| 10. END WHILE |
| END |

在算法3.2中第3~8行表示在监测点集合中，对轮询时间为当前时刻的检查点发送请求消息。

## 3.3 路由和流调度机制设计

本节分为两部分，第一部分为基于链路偏好的路由算法，第二部分为支持负载均衡与节能的流调度算法。在第一部分路由算法的设计中，同时考虑负载均衡与节能因素，提出了基于链路偏好的随机路由算法。由于网络状态随时间变化，因此，路由算法选择链路的概率会随着链路的偏好而变化。第二部分，本文设计了两种流调度算法。其中，第一种每次调度只调整一个数据流，第二种调度方案对整个网络进行迭代式调度直到满足停止条件。

### 3.3.1 基于链路偏好的随机路由算法

在SDN中如果采取最短路径路由算法，可能会造成网络中负载极大的不均衡，甚至于局部链路会出现拥塞丢包。在负载均衡的调度算法中，通过将拥塞链路上的最大流调度到其他负载较轻的路径上实现网络设备负载均衡同时又缓解了局部链路的拥塞。在解决网络能耗问题时，通常对于利用率较低的设备调度数据流使其空闲下来，进而进入休眠模式。但是，这两种都属于流调度机制，也就是说问题出现之后解决问题。

本文提出的基于链路偏好的随机路由机制在路由阶段就采取措施，使其同时兼顾负载均衡与节能。它的特点是随机性与动态性。其随机性体现在选择下一跳的过程中，在这个过程中本文采用Softmax的思想，对于当前网络状态有利的下一跳并不是一定选择，而是赋予其较大的被选概率，同样对于当前网络状态不利的下一跳赋予其较小的被选概率。其动态性体现在偏好随有向边带宽利用率和网络流量动态变化，如图3.3所示。

本文将有向边的利用率通过函数映射为偏好。当网络整体流量偏低时，利用率与偏好的映射曲线如蓝色曲线所示。随着网络整体流量的上升，整个曲线随之向右移动，如黑色曲线或绿色曲线所示。偏好不仅随着链路利用率变化还随着整体网络流量动态变化，这使得路由和流调度算法也能动态地做出调整。

以黑色曲线为例说明它为什么能同时实现负载均衡与节能的效果。在曲线对称轴左侧，当一条链路上经过的数据流越多，其被后续数据流选中的机会越大；当然本文与双桥实验不一样的地方是在对称轴右侧链路会随着其利用率升高而受到抑制。假定映射曲线为图3.3中黑色曲线时，在对称轴左侧网络中利用率在0.3以下的链路，其偏好映射将非常低，进而导致这条链路被选中的概率很低，对于新到达网络的数据流起到抑制的作用。这条链路的利用率将持续降低，经过一段时间，它的利用率甚至可能会收敛到0，这时就可以使这条链路进入休眠模式。如果短时间内它连续被多个数据流选中或者被一个大象流选中，那么它的利用率迅速上升，偏好也随之上升。这将对它起到一个促进作用，有利于提升链路的利用率。反之，如果一条链路利用率在对称轴右侧，假定其被大象流选中利用率迅速上升，其偏好则会随之下降，它将抑制自己被新的数据流选中，可以有效缓解链路的拥塞。假定其在连续一段时间内没有被新数据流选中，其利用率可能会出现下降，偏好反而会上升，进而促进自己被新的数据流再次选中。

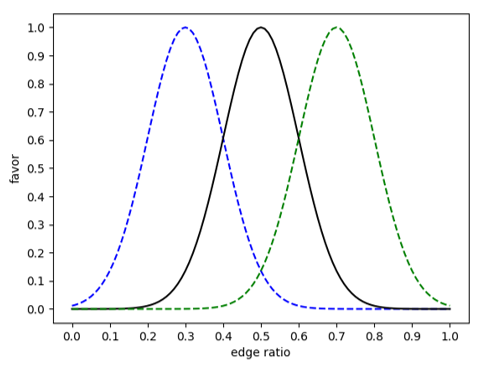


图3.3 链路利用率与偏好关系图

Fig. 3.3 Link utilization and preference relationship

（1）链路动态偏好计算

在3.2节通过流量监测机制，控制器通过监测数据流速率进而得到有向边上的带宽占用率。通过前文的分析，本文给出将利用率映射为偏好的计算如公式（3.18）所示。

 （3.18）

 （3.19）

其中，为大于0的实数，为网络中有数据流经过的有向边的平均利用率，为有数据流经过的有向边的数目。根据公式（3.18）和（3.19）可将带宽利用率矩阵转化为链路偏好矩阵。

（2）算法描述

为了同时兼顾负载均衡与节能，本文提出的基于链路偏好的随机路由算法（Preference-based Random Routing algorithm, PbRR）。假定节点到目的节点有个邻接点可供选择，并且到目的节点的路径不构成环路，则称为候选节点。算法3.3采取Softmax函数将与邻接点相邻的边的偏好转化为概率。在已知各候选节点被选择的概率的情况下，再以轮盘赌的方式进行选择。具体的做法是先随机一个区间内的实数作为选择的概率，然后累加候选节点的概率值直到大于，则当前节点就作为被选中的节点。假定的偏好为，其对应的概率为，则的计算如公式（3.20）所示。

 （3.20）

其伪代码描述如算法3.3所示。

|  |
| --- |
| **算法3.3 基于链路偏好的随机路由算法PbRR** |
| **输入：**剩余带宽矩阵，偏好矩阵，源节点和目的节点 |
| **输出：**源节点到目的节点的一条路径 |

BEGIN

|  |
| --- |
| 1. Initialize the  as null; |
| 2. ; |
| 3. Add  to the ; |
| 4. WHILE, DO |
| 5. Initialize  and  ; |
| 6. FOR each adjacent node  of , DO |
| 7. IF exist a acyclic  that the node  can reach the node , THEN |
| 8. Add  to ，; |
| 9. ; |
| 10. END IF |
| 11. END FOR |
| 12. Initialize a random ，; |
| 13. FOR each node  in , DO |
| 14. ;// is the probability of node |
| 15. IF , THEN |
| 16. Select node , add  to ; |
| 17. Update the node , set ; |
| 18. Break;// GO TO step 4 |
| 19. END IF |
| 20. END FOR |
| 21. END WHILE |
| 22. IF the node , THEN |
| 23. Set  as null; |
| 24. END IF |
| 25. RETURN ; |
| END |

其中，第5~11行为选择候选节点的过程。第13~20行为选择下一跳的过程。

为了确保在路由时能有较好的节能效果，本文的路由算法会优先基于活动拓扑调用PbRR路由算法为数据流计算路径。如果计算出的路径不能满足约束条件（例如带宽约束），则将会基于全局拓扑为数据流计算路径。

### 3.3.2 支持负载均衡与节能的流调度算法

尽管本文进行路由计算时已经兼顾了负载均衡与节能的因素，但是当数据流中的第一个数据包到达控制器时，控制器无法知道有关数据流的带宽等详细信息。在不知道有关数据流详细信息的情况下，控制器只能利用与该数据流相关的历史流量的特征，例如目的地址相同的数据流的带宽，进行估算。然后依据估算的特征对数据流进行路由计算。这样路由算法得到的路径一定程度上依赖于控制器对数据流特征估算的准确性。如果估算的不够准确同样会出现局部链路拥塞等情况，因此设计一个支持负载均衡与节能的流调度算法至关重要。

本文在路由及流调度算法中均采用链路的偏好作为启发式信息，为提高算法的性能，在进行路由或者流调度时需要计算多条路径，然后根据网络的局部状态（路径信息）或者全局状态对多条路径进行评价，选择最优的路径作为调度路径。

**3.3.2.1触发策略**

本文设计的支持负载均衡与节能的流调度算法中触发流调度的情况分为三种：1）对拥塞的链路上数据流进行调度，这种情况最为紧急，所以其调度优先级最高，尽快的对网络拥塞链路上数据流进行调度能有效降低网络的丢包率；2）是对可以节能的链路上的数据流进行调度以实现节能的效果，其优先级最低；3）当网络链路负载不均衡时，对数据流进行调度使网络尽快恢复到链路负载均衡的状态。

（1）拥塞链路上流调度

本文设计的流调度算法采用防御性的策略，在链路的利用率达到较高的阈值就出发预警，而不是检测到拥塞才发出预警。此外，当一个新的数据流到达网络时，链路的流量会出现一个瞬间的峰值，可能会导致链路呈现一种假“拥塞”的状态。为了避免这样的假“拥塞”造成的不必要的流调度，本文采取了一种弹性策略。如3.1节中介绍的链路模型，每条有向边都有一个值。只有当其连续触发预警信息号使值小于0时，才认定是一个真的预警；否则认为这是一个假预警就什么也不做。

（2）支持节能的流调度

由于网络的性能指标（丢包率、时延等）通常比能耗指标重要，因此在节能的流调度算法设计中，本文设置节能的流调度优先级最低。与拥塞链路上调度类似，为避免链路上流量时有时无造成的链路或节点的频繁状态切换，本文对需要进行节能调度的数据流采用弹性的策略。在链路模型中，每条有向边都有一个值。只有当其值小于0时，才需要对数据流调度。

（3）支持负载均衡的流调度

本文采用链路的带宽利用率表示链路的负载，并使用链路带宽利用率的标准差作为负载是否均衡的衡量指标，其计算如公式（3.21）所示。

 （3.21）

其中，设链路利用率标准差阈值为，当时触发负载均衡的流调度。

**3.3.2.2算法描述**

本文设计了两种数据流调度算法：负载均衡与节能的流调度（Load balancing and Energy-saving flow scheduling, LoadbE）和迭代式负载均衡与节能的流调度（Load balancing and Energy-saving flow scheduling with iteration, LoadbE-it），简称为非迭代式流调度和迭代式流调度。非迭代式流调度算法每次对触发调度的数据流进行一次调度。具体地，首先，分别在活动拓扑和全局拓扑上调用PbRR算法次，然后根据路径评价计算方法选出最优方案。非迭代的数据流调度算法伪代码描述如算法3.4所示。

|  |
| --- |
| **算法3.4 非迭代式流调度LoadbE** |
| **输入：**数据流标识，PbRR执行次数，网络状态 |
| **输出：**调度流，更新网络状态 |

BEGIN

|  |
| --- |
| 1. Initialize ; |
| 2. WHILE  // active topology PbRR, DO |
| 3. Using PbRR on the active topology to get a new  for ; |
| 4. IF, THEN |
| 5. Increase the counter of ; |
| 6. ELSE |
| 7. Add  to，and set the counter of  is 1; |
| 8. END IF |
| 9. END WHILE |
| 10. Initialize ; |
| 11. WHILE  // global topology PbRR, DO |
| 12. Using PbRR on the global topology to get a new  for ; |
| 13. IF , THEN |
| 14. Increase the counter of ; |
| 15. ELSE |
| 16. Add  to ，and set the counter of  is 1; |
| 17. END IF |
| 18. END WHILE |
| 19. Score each path in , and select the best one as ; |
| 20. Change the path of flow  with ; |
| 21. Update the network state such as ; |
| END |

其中，第2~9行是在活动拓扑上调用PbRR算法次并记录路径，第11~18行是在全局拓扑上执行次PbRR算法并记录路径，第19行采用基于局部的网络状态信息对中的路径进行评价，选出最佳路径作为调度路径。假定数据流执行流调度算法后产生的路径集合为，则为每条路径的评分方式如公式（3.22）~（3.24）所示。

 （3.22）

 （3.23）

 （3.24）

其中，为路径出现的次数，为路径经过的边上的最大链路利用率和最小链路利用率，为路径的能耗，和的计算分别如公式（3.23）和（3.24）所示。

基于局部信息的最佳调度路径计算如式（3.25）所示。

 （3.25）

迭代式的流调度算法每次执行都会调度多个数据流。因为每个数据流都可能有多种调度路径，因此如何选择数据流的调度顺序和调度路径同时保证调度后网络的状态最佳是本算法要解决的问题。迭代式算法的处理流程为前一半迭代在活动拓扑上进行，后一半的迭代在全局拓扑上进行。每次选出最优路径后都会计算调度后的网络状态，并与当前状态进行比较，只有比当前状态更好时才引起执行调度，所以每一次的迭代并不一定会引起调度。此外每次迭代的处理首先选择数据流，然后为数据流计算多条路径，最后对多条路径进行基于网络的全局信息的评分选出最佳调度路径。

迭代式的流调度算法解决的第一个问题是数据流的选择问题。对于当前网络中的数据流，本文根据数据流路径上的启发的信息选择调度后对网络状态改善最大的流。对数据流，启发式信息计算如公式（3.26）所示。

 （3.26）

其中，为数据流经过的有向边的带宽利用率，为数据流的路径跳数。值越大，代表其对负载的标准差的贡献越大。

迭代式流调度算法伪代码描述如算法3.5所示。

|  |
| --- |
| **算法 3.5 迭代式流调度算法LoadbE-it** |
| **输入：**迭代次数以及网络状态矩阵，拓扑类型 |
| **输出：**调度多个数据流，更新网络状态 |

BEGIN

|  |
| --- |
| 1. Initialize ,// 1 is active topology, 0 is global topology; |
| 2. WHILE  and trigger conditions are satisfied , DO//iteration |
| 3. Initialize ; |
| 4. IF, THEN |
| 5. ; |
| 6. END IF |
| 7. Select the best flow as  using（3.26）; |
| 8. Add the shortest path to ; |
| 9. Add each edge from the shortest path of  to  with the ; |
| 10. FOR each  of , DO |
| 11. Get the shortest path  of  without ; |
| 12. Add  to ; |
| 13. END FOR |
| 14. Add the path of PbRR returned to ; |
| 15. Score for each path in  get the best one as ; //score and select |
| 16. IF the schedule for  is better than current, THEN |
| 17. Change the path of  with new ;//schedule the flow |
| 18. Update the network state such as ; |
| 19. END IF |
| 20. ; |
| 21. END WHILE |
| END |

其中，第7行使用公式（3.26）从当前网络中选出一个数据流，第9~14行为数据流计算多条调度路径，第15行对路径进行评分并选择，第16~19行进行调度并更新网络状态。

就迭代式流调度算法而言，假定在每次迭代的过程中产生的路径集合记为，在全局拓扑或者活动拓扑的路径评分过程中均采用公式（3.27）的评分函数。其中，为网络中链路利用率最大值，代表数据流按路径调度网络活动链路利用率的标准差，为数据流按路径调度网络的能耗，为网络以最大功率工作的总能耗。

 （3.27）

基于全局网络信息进行路径选择如公式（3.28）所示，即每次选出评分最大的路径，其中路径集合需要和当前的数据流路径比较。

 （3.28）

## 3.4 数据平面路径安装与更新策略设计

在控制器为数据流计算了一条路径后，路径必须通过FlowMod消息才能安装到交换机。当对数据流进行调度后，经常需要对交换机中的流表项做修改和删除。然而，在OpenFlow v1.3版本中当流表项发生冲突时交换机流表项不会做任何改变，例如安装流表时发现已经存在相同匹配项的流表项但是转发接口不同。本节主要讲述路径的安装、更新和删除策略，其中对流表项的安装、更新和删除在OpenFlow协议中均是通过FlowMod消息实现。

### 3.4.1 路径流表安装策略

为了能精确定位到交换机中已安装的流表项，本文使用了OpenFlow协议中的字段。其作为数据流全局唯一的标识符，每当有新的数据流到达时自动更新。除此以外，本文设计的路径安装策略只产生一次PacketIn消息就完成路径流表项的安装，如3.4所示，当主机到的数据时到达交换机时，首先控制器根据PacketIn消息将为数据流计算转发路径，然后根据路径将FlowMod消息依次下发至交换机。也就是说，当数据流到达及后续节点时，将直接匹配流表项而不用再次产生PacketIn消息。



图3.4流表项安装

Fig. 3.4 The installation of flow table entry

### 3.4.2 路径流表更新策略

数据流发生调度时，数据流的路径必然会发生变化，控制器必须在尽可能短的时间内将新的路径安装到交换机。本文设计了两种不同的路径更新策略：删除式更新和修改式更新。但是在更新之前必须确定删除或者修改的流表项，在OpenFlow协议规范中，如果的值与相等，则认定是同一流表项。

（1）删除式更新

删除式更新采取的是由控制器直接对路径上的交换机发送带有删除命令的FlowMod消息，交换机收到消息后将删除对应的流表项。当新的数据流到达时，数据流将再次产生PacketIn消息，执行PbRR算法得到新的路径并安装。这种更新方式简单直接，只需要依次发送FlowMod消息删除流表项就能实现。然而，删除式更新的缺点也显而易见，一方面路径更新并不能利用数据流监测机制测量的流带宽等信息，只是单纯把它作为新的数据流看待；另一方面，新计算的路径可能会与调度路径不同。

（2）修改式更新

修改式更新解决了上述更新方式的缺点，在原数据流流表项的基础上，通过添加、修改或者删除操作，对原来路径的流表项更新为新路径的流表项。如图3.5所示，假定原来数据流路径为，用黑色实线表示，调度路径为用黑色虚线表示，则控制器将会对中的流表项进行修改，使其指令中动作项由改为，同时在中增加的流表项，对中的流表项不做改变。此外，还需要对交换机中的流表项进行删除操作。



图3.5 流表项更新

Fig. 3.5 The updating of flow table entry

由于对流表项的修改是按照控制器发出的FlowMod消息进行的，所以控制器与交换机之间交互过程及交换机从流表中修改某一条流表项过程都存在着时间开销。在这个时间段内，如果修改了某条流表项之后将会导致路径流表项的不连贯，例如在图3.5中，当交换机将其流表项指令中动作项由改为后，即安装成功，未安装成功。如果此时交换机收到新的数据包，数据包会被转到，但是由于的路径并未安装，因此将产生PacketIn消息，这样无疑浪费了控制器的资源。

为了能平滑的进行新路径与原路径的切换，本文采用从后向前的路径更新方式，先增路径段的流表项，再对交换机中的流表项改为路径段。这样的好处是在新路径安装完成之前，数据一直会从原路径转发，当新路径更新之后，新路径马上就可以投入使用，而不用产生PacketIn消息。

## 3.5 本章小结

本章首先介绍了动态负载均衡与节能机制的系统架构，然后介绍了每个模块的详细设计。其中在流量监测机制中介绍了本文的自适应流量监测算法；在动态负载均衡与节能模块介绍了链路动态偏好的计算、PbRR路由算法和两种数据流调度算法（迭代式和非迭代式）；在流表安装与更新模块介绍了两种流表更新策略，并进行了对比。

# 第4章 仿真实现与性能评价

本章对SDN中动态负载均衡与节能机制进行了仿真实现与性能评价。在仿真实现部分给出了动态负载均衡与节能机制的主要数据结构和函数。同时，为了公平地与本机制进行对比，分别将经典的ECMP算法和LABERIO算法进行了改进，并进行仿真实现，改进后的算法将具有节能功能。最后，本文将改进的ECMP（Improved Equal-Cost Multi-Path, IECMP）和改进的LABERIO（Improved LABERIO, ILABERIO）分别与本文提出的LoadbE和LoadbE-it机制进行了性能对比，并从网络的负载均衡、能耗和性能等角度进行了评价与分析。

## 4.1 仿真平台搭建

本文SDN中动态负载均衡与节能机制的仿真实现硬件环境为Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz，4.00G 内存，操作系统为Ubuntu 16.04 LTS 。整个机制基于Python2.7实现，以Sublime text 2为主要开发工具，在Ubuntu 16.04平台下调试运行。

### 4.1.1 实验环境

在SDN平台搭建过程中，对控制平面的仿真，本文采用的是开源控制器Ryu对数据平面的仿真，本文采用的是开源的仿真工具Mininet。

（1）Ryu控制器

Ryu是一个基于组件的软件定义网络开源控制器框架。在Ryu开源项目中，有一系列预先定义的基础组件，研究人员可以在这些组件的基础之上进行扩展修改以创作自定义的应用。

此外，Ryu是基于Python语言编写，这使得研究者在Ryu中开发网络应用时能更加关注算法逻辑。在对OpenFlow协议的支持上，Ryu持续跟进，当前最新版本的Ryu v4.19中提供了对OpenFlow v1.0至v1.5的接口支持。通过Ryu提供定义良好的API，使开发人员可以轻松创建新的网络管理和控制应用程序。

（2）Mininet

Mininet是一个开源的网络仿真器，或者更准确地说是一个网络仿真协调系统。它在单个Linux内核上运行一系列终端主机、交换机、路由器和链路。它使用轻量级的虚拟化，使单个系统看起来像一个完整的网络，运行相同的内核，系统和用户代码。Mininet主机的行为就像一个真正的机器，用户可以通过ssh进入它并运行任意程序。

Mininet中的交换机采用的是Open vSwitch（OVS），OVS是一个支持OpenFlow协议的开源虚拟交换机。通过对OVS的封装，Mininet能够很容易地构建出复杂的网络拓扑。同时，在用户在使用Mininet进行网络仿真时，可以给链路设定一定的带宽、时延和丢包率等参数。数据包经过这些链路时，就像真实的环境中一样。

此外，Mininet的优点就是：易用、易迁移。用户可以使用Python脚本创建网络拓扑，也可以通过可视化界面创建拓扑。当仿真程序从一台机器迁到另一台机器，代码几乎不需要进行任何的修改。Mininet还封装了很多Linux系统中的工具，例如，用户可以使用Iperf工具对网络性能进行测试，可以通过配置linux HTB（Hierarchy Token Buffer）队列管理规定实现网络的QoS保证等。

### 4.1.2 实验拓扑

本文采用文献[39]和文献[21]中的拓扑进行实验测试，其中，文献[39]中的拓扑较为简单，命名为Topo1，如图4.1所示；文献[21]中的拓扑为数据中心拓扑，命名为Topo2，如图4.2所示。其中，为了方便控制器管理，在拓扑Topo1中本文对主机的IP地址和MAC地址配置如表4.1所示，对交换机配置如表4.2所示。



图4.1 实验拓扑Topo1

Fig. 4.1 Experiment topology Topo1



图4.2 实验拓扑Topo2

Fig. 4.2 Experiment topology Topo2

表4.1 主机配置

Table 4.1 Host configuration

|  |  |  |  |
| --- | --- | --- | --- |
| Host | IP | MAC | MAX CPU rate |
| h1 | 10.0.0.1 | 00:00:00:00:00:01 | 10% |
| h2 | 10.0.0.2 | 00:00:00:00:00:02 | 10% |
| … | … | … | … |
| h8 | 10.0.0.8 | 00:00:00:00:00:08 | 10% |

表4.2 交换机配置

Table 4.2 Switch configuration

|  |  |
| --- | --- |
| Switch | Datapath |
| s1 | 1 |
| s2 | 2 |
| … | … |
| s7 | 7 |

在拓扑Topo1的链路配置中，交换机s1与主机h1/h2/h3/h4相连的端口分别为1/2/3/4，s7与主机h5/h6/h7/h8相连的端口分别为1/2/3/4，网络中其他有关链路的参数配置如表4.3所示。

表4.3 链路配置

Table 4.3 Link configuration

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Switch1 | Switch2 | Port1 | Port2 | Bandwidth | Delay | Maximum queue-size |
| s1 | s2 | 5 | 1 | 1000bps | 1ms | 100Byte |
| s1 | s3 | 6 | 1 | 1000bps | 1ms | 100Byte |

表4.3（续）链路配置

Table 4.3 (Continued) Link configuration

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Switch1 | Switch2 | Port1 | Port2 | Bandwidth | Delay | Maximum queue-size |
| s1 | s4 | 7 | 1 | 1000bps | 1ms | 100Byte |
| s2 | s5 | 3 | 1 | 1000bps | 1ms | 100Byte |
| s5 | s7 | 3 | 5 | 1000bps | 1ms | 100Byte |
| s4 | s6 | 3 | 1 | 1000bps | 1ms | 100Byte |
| s6 | s7 | 2 | 7 | 1000bps | 1ms | 100Byte |
| s3 | s2 | 2 | 2 | 1000bps | 1ms | 100Byte |
| s3 | s4 | 5 | 2 | 1000bps | 1ms | 100Byte |
| s3 | s5 | 3 | 2 | 1000bps | 1ms | 100Byte |
| s3 | s7 | 4 | 6 | 1000bps | 1ms | 100Byte |

## 4.2 仿真实现

本节首先介绍控制平面中动态负载均衡与节能机制各模块的实现，然后介绍数据平面的数据流模拟，最后介绍在实验过程中重要参数的设置。

### 4.2.1 主要数据结构及函数实现

（1）主要数据结构

表4.4 主要数据结构与类

Table 4.4 The main data structure and class

|  |  |
| --- | --- |
| 数据结构 | 描述 |
| CoreEdge | 核心链路类，链接两个交换机的链路 |
| Edge | 接入链路类，链接交换机与主机的链路 |
| Host | 主机节点类，存储主机IP等信息 |
| Node | 交换机节点类，存储节点状态信息 |
| Flow | 数据流，存储数据流的相关信息 |
| Checks | 数据流监测点集合 |
| BwTM | 网络链路带宽占用矩阵 |
| RwTM | 网络链路剩余带宽矩阵 |
| FavorTM | 网络链路偏好矩阵 |
| RatioTM | 网络链路利用率矩阵 |

在动态负载均衡与节能机制的设计中，本文分别对网络的节点、链路以及数据流进行了建模，以此定义了节点、链路、数据流和主机的类。同时为了存储网络的状态，本文构建了网络的剩余带宽矩阵、链路偏好矩阵、链路利用率矩阵等，具体见表4.4所示。

（2）流量监测机制的主要函数

在流量监测机制中涉及的功能是控制器定时对数据流发送轮询请求，并且在收到响应后，对链路的剩余带宽矩阵进行更新，计算链路偏好矩阵，更新数据流的状态，为数据流设置新的监测点。当收到FlowRemove消息后需要对数据流产生的数据结构进行清理。流量监测机制的主要函数如表4.5所示。

表4.5 流监测机制的主要函数

Table 4.5 The main function of traffic monitoring

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| \_monitor() | 轮询函数 |
| \_record() | 记录每个时刻网络的能耗、吞吐量等 |
| \_request\_stats() | 发送轮询请求 |
| \_add\_check() | 添加监测点 |
| updateLinksRate() | 更新链路状态 |
| getWeight1() | 计算链路偏好 |
| getNewInterval() | 更新轮询间隔 |
| recordFlows() | 流信息登记 |
| flow\_removed\_handler() | FlowRemove消息的处理 |

（3）流表项的安装与更新

在流表安装和更新阶段，主要涉及流表的增加、删除和修改功能，其主要函数如表4.6所示。

表4.6 流表项的安装与更新的主要函数

Table 4.6 The main function of installing and updating for flow entries

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| change\_path() | 将原路径修改为新路径 |
| add\_flow() | 安装指定的流表项 |
| del\_path() | 删除原路径上多余的流表项 |
| del\_flow() | 删除指定交换机的指定流表项 |
| modify\_flow() | 修改指定交换机的指定流表项 |

（4）路由机制的主要函数

路由机制中主要包含对PacketIn消息的处理、获取流的头部信息、基于偏好的路径计算和流路径的安装等功能，其主要函数如表4.7所示。

表4.7 路由机制的主要函数

Table 4.7 The main function of routing

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| \_packet\_in\_handler() | PacketIn消息处理，提取包头信息 |
| receiveARP() | ARP消息单独处理，避免广播风暴 |
| getFirstPath() | 网络中只有一个数据流时的路径计算 |
| getInitMatrix() | 根据拓扑类型构建流量矩阵和偏好矩阵 |
| PbRR() | 基于链路偏好进行路由 |
| dijkstra() | 根据拓扑类型计算最短路径 |
| BFS() | 判断目的节点是否可达 |
| addPathFlow() | 路径安装 |
| getOutport() | 获取与目的主机相连的交换机端口 |

（5）流调度机制的主要函数

表4.8 非迭代流调度机制的主要函数

Table 4.8 The main function of non-iterative flow scheduling mechanism

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| scheduleFlows() | 调度指定的数据流 |
| getNewPath() | 为数据流计算多条备选路径 |
| getRerouteMatrix() | 根据拓扑类型构建流量矩阵和偏好矩阵 |
| evalue() | 路径评分预处理 |
| score4() | 路径评分 |

表4.9 迭代流调度机制的主要函数

Table 4.9 The main function of iterative flow scheduling mechanism

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| getNewSchedule\_it () | 迭代地调度数据流直到满足终止条件 |
| getRerouteMatrix\_it () | 根据拓扑类型构建流量矩阵和偏好矩阵 |
| getpaths\_it () | 为调度计算多条备选路径 |
| selectpath\_it () | 从备选路径中选择评分最高的路径 |
| score4\_it() | 路径评分 |

非迭代的流调度的主要函数见表4.8所示，其主要包含数据流的调度、多路径计算和路径评估功能。迭代的流调度的主要函数见表4.9所示，其主要包含迭代式数据流的调度、迭代式多路径计算和路径选择功能。

### 4.2.2 实验设置

由于本机制需要与不同的算法进行对比，在同一次实验中只能选择一种流调度算法，因此本文设计了如图4.3所示的实验框架。



图4.3 实验框架

Fig. 4.3 Experiment framework

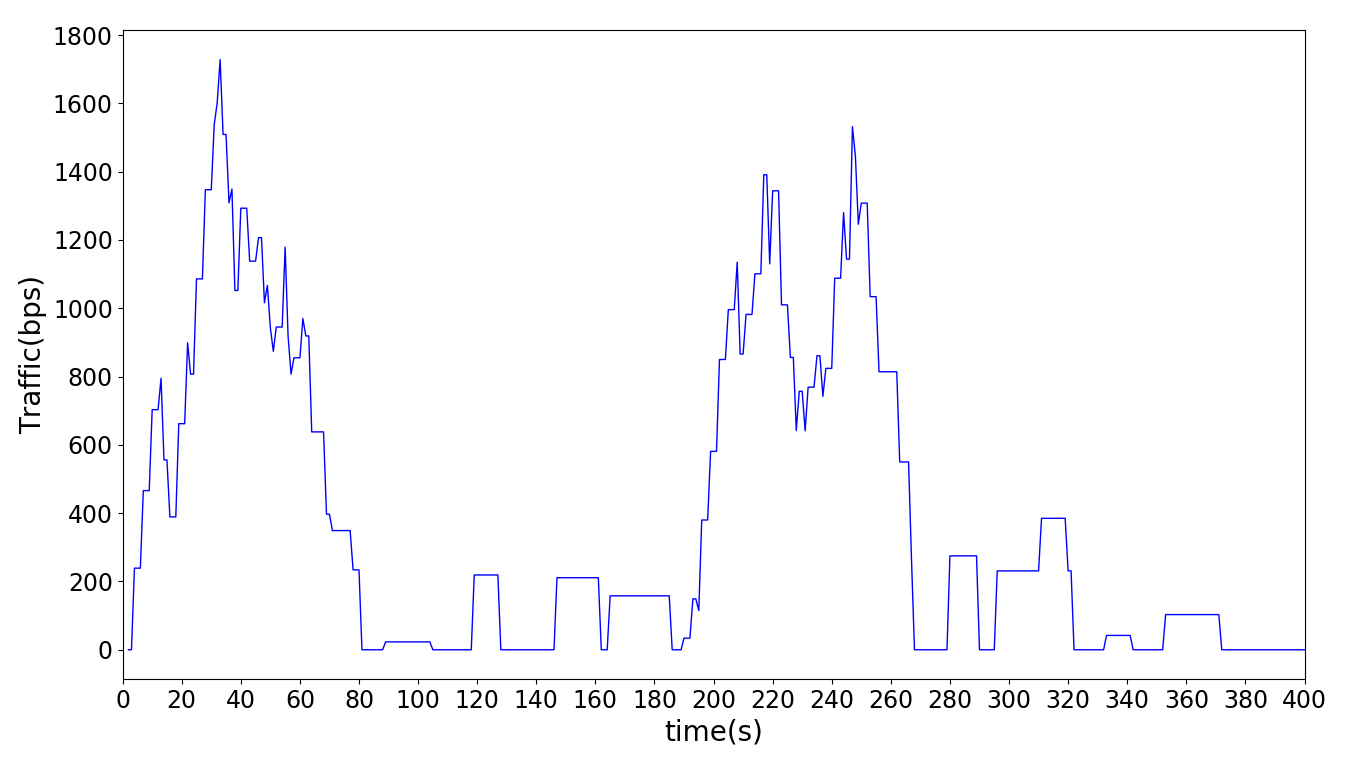


图4.4 网络流量模拟

Fig. 4.4 Network traffic simulation

其中，Ryu和Mininet根据拓扑配置文件生成一致的拓扑，流文件用于确保四种算法输入的一致性。为了尽可能准确地模拟网络的真实情况，本文对不同时段的数据流进行了模拟，生成了连续两天不同时段的数据流。流生成规则如下：1）数据流到达时间间隔随机；2）发送方和接收方随机；3）数据流的持续时间随机在内取值；4）数据流的发送速率随机在内取值；5）白天时段数据流到达时间间隔范围为；6）夜间时段数据流到达时间范围;根据数据流的流量计算得出，在每个时刻网络的整体流量如图4.4所示。

本文在Mininet中对Iperf发包工具进行封装，自定义了Iperfread命令，实现了自动化测试，它通过读取生成的数据流文件，确保在各算法的输入一致。同时，对于每个数据流，Iperfread命令将结合流文件调用Iperf工具进行发送或接收数据。

### 4.2.3 参数确定

经过多测实验之后，本文对于SDN中动态负载均衡与节能机制中的参数设置如表4.10所示。

表4.10 主要参数描述

Table 4.10 The main parameter description

|  |  |
| --- | --- |
| 参数设置 | 参数描述 |
|  | 休眠节点固定能耗 |
|  | 开启状态节点固定能耗 |
|  | 端口负载最大时能耗 |
|  | 端口空载时能耗 |
|  | 流速率变化阈值 |
|  | 流速率变化阈值 |
|  | 流速率轮询时间间隔基数 |
|  | 偏好映射曲线参数 |
|  | 低于链路利用率的缓冲初始值 |
|  | 高于链路利用率的缓冲初始值 |
|  | 链路利用率休眠阈值 |
|  | 链路利用率过高预警阈值 |
|  | 流表项软超时 |
|  | 流表项硬超时 |

## 4.3 性能评价

为检验本文设计的SDN中的动态负载均衡与节能机制的可行性与有效性，本文分别以链路的平均利用率、链路负载标准差、网络能耗、吞吐量等性能指标为参考，对比了IECMP和ILABERIO机制。

### 4.3.1 评价指标

由于本文设计的机制包含负载均衡与节能，因此在评价指标的选择上必须能体现两者的特点。

首先，对于负载均衡的评价，本文参考LABERIO机制中的评价方法，设计了以链路利用率的标准差作为衡量网络在数据流调度时的负载均衡度。其值越小代表网络负载越均衡；反之，其值越大则反映网络的链路负载差别很大。时刻链路利用率的标准差计算如公式（4.1）所示。

 （4.1）

其中，是时刻活跃链路的利用率，是时刻活跃链路的平均利用率，为活跃链路的数目。

其次，对于节能的指标采用全网的能耗功率来衡量。网络的状态会随着流量的变化而变化，因此只要能够通过计算画出网络的功率曲线，那么曲线下的面积则代表这段时间内网络的能耗。同时功耗曲线的变化一定程度上能够反映节点与链路的休眠或开启状态是否稳定。时刻网络能耗计算方式如公式（4.2）所示。

 （4.2）

最后，也最为重要的是本机制对网络的性能产生的影响。因此，本文需要从网络性能的角度考虑不同机制对网络造成的影响。在这里，本文将关注活跃链路的平均利用率、网络的吞吐量、能效等指标。其中，网络在时刻的活跃链路的平均利用率、吞吐量和能效分别如公式（4.3）~（4.5）所示。

 （4.3）

 （4.4）

 （4.5）

其中，为时刻活跃链路数目，为时刻数据流数目，为时刻数据流的速率。

### 4.3.2 对比机制

在经典的ECMP算法和LABERIO算法中是不包含节能机制，本文为了实现公平的对比，在两种算法原来的基础之上引入了节能模块。

改进的ECMP算法机制：数据流到达控制器时，控制器为其计算多条相同代价的路径，控制器根据第一个数据包的头部信息进行Hash为数据流选择一条路径进行路由。同时，本文在经典ECMP机制上加入了节能策略，当链路利用率为0时，链路被休眠，当一个节点的所有相关链路都被休眠，节点也将被休眠。

改进的LABERIO机制：LABERIO机制会预先计算存储任意两个交换机间的多条路径，同时监测每个时刻网络的状态。当数据流达到时，根据每条路径的剩余带宽，选择最大的一个作为初始路径。当链路出现负载不均衡时，LABERIO算法使用替代路径代替负载最大的链路。本文，在LABERIO机制中加入了节能模块，使网络中带宽利用率为0的链路进入休眠状态，同时，如果节点的所有相关链路被休眠，则节点也将被休眠。

### 4.3.3 性能测试与分析

本文首先分别使用非迭代流调度机制、迭代流调度机制、IECMP机制ILABERIO机制进行实验，并对结果进行了对比。最后，本文从负载均衡、节能和网络性能三个角度对实验结果进行了分析。

由于四种机制的仿真实验结果曲线波动较大，不容易直接进行对比，因此本文对部分实验结果采用累积分布函数（Cumulative Distribution Function）进行对比。累积分布函数曲线是反映数据分布的曲线，表示概率分布函数，在图中以Topo1-b或者Topo2-b表示。其中“Topo1”或者“Topo2”代表在不同拓扑下的实验结果，“-a”代表原始数据随时间变化曲线，“-b”表示累积分布函数曲线。

（1）负载均衡度量

网络中活跃链路利用率的标准差随时间变化曲线和累积分布函数曲线如图4.5所示。在Topo1-a和Topo2-a中可以看出在负载均衡度量上，当网络处于流量高峰时期（比如白天），指标波动较大，当网络流量较低时（比如夜间），指标波动较小。这是由于白天时段，数据流密集到达，容易引起局部链路利用率突然变化，进而导致指标波动较大；夜间时段，数据流较稀疏，网络会对数据流的路径进行一定程度的聚合，进而造成活跃链路的利用率同时升高或降低，不影响指标。

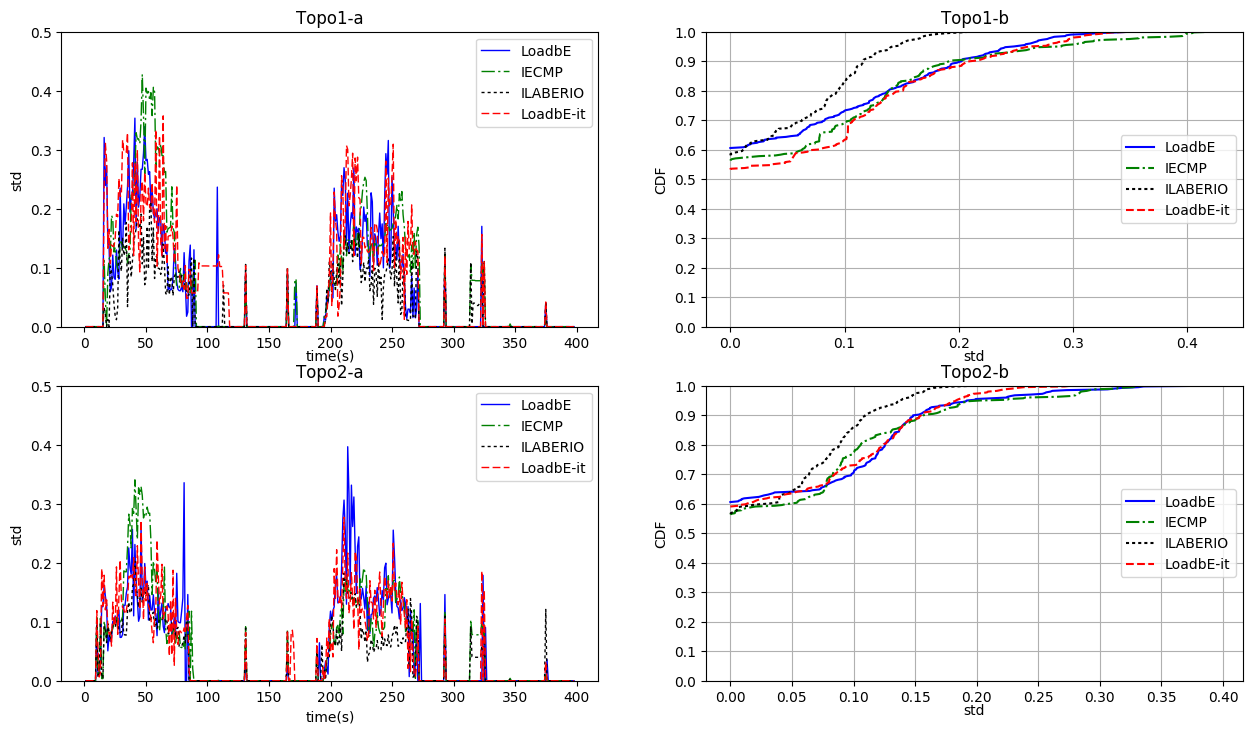


图4.5 链路利用率标准差

Fig. 4.5 Standard deviation of link utilization

不同算法的链路利用率方差的累积分布如Topo1-b和Topo2-b所示，曲线越高代表在该值以内概率分布越大。两种拓扑上总体趋势相同，表现最佳的是ILABERIO机制，其次是LoadbE，IECMP和LoadbE。其中，后三种机制总体表现相差不大。在Topo1-b中，指标在0.1以内，ILABERIO机制可达0.85，LoadbE分别为0.74，IECMP为0.7，LoadbE-it为0.64。出现这种情况的原因可能是由于每次进行路径调度的时候ILABERIO使用子路径取代高负载的链路，其影响的链路范围较少。然而，其它机制直接使用计算新的路径替代原来的一条流路径，影响的链路范围广，进而导致网络链路的利用率波动较为剧烈，产生较大的标准差。在Topo2-b中，指标在0.1以内，可以看出ILABERIO机制仍领先于其他机制，其它机制略有不同，印证了子路径替换的ILABERIO机制比全路径替换的其它机制在链路利用率标准差上表现更好。

（2）网络能耗度量

网络的功率随时间变化曲线和累积分布函数曲线如图4.6所示。网络的能耗则表现为功率曲线下的面积。从图Topo1-a和Topo2-a中可以明显看出当网络中流量较大时，当网络中流量不断增加时，网络功率逐级升高，当网络流量下降时，功率逐级下降。这是因为随着流量的升高，网络为避免造成局部链路的拥塞，不断开启新的节点和链路，组成新路径进行数据传输。从曲线的波动情况看，LoadbE-it最剧烈，ILABERIO最稳定，说明ILABERIO机制的网络状态比较稳定，LoadbE-it对网络流量的变化比较敏感，这是因为LoadbE-it每次调度多个数据流，对网络整体影响较大。此外，从Topo2-a中已经明显可以看出ILABERIO机制的功率曲线总是处在其他机制以下。

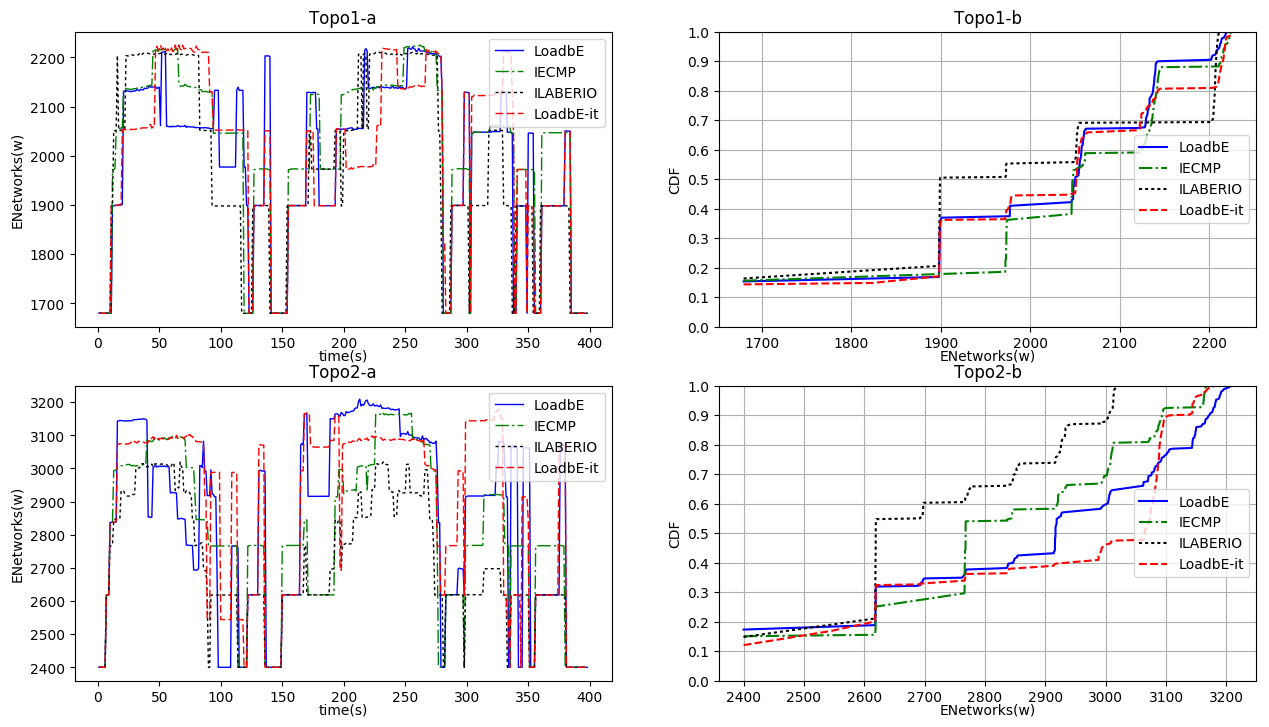


图4.6 网络功率

Fig. 4.6 Network power

在Topo1和Topo2仿真结果中网络功率的累积分布曲线如Topo1-b和Topo2-b所示。在Topo1-b中，在相同的功率下，ILABERIO机制曲线最高，代表相同功率下，其功耗分布概率较大，能耗较小；接下来是LoadbE和LoadbE-it机制，其功耗在累积分布曲线上高于IECMP机制。在Topo2-b中，四种机制差异明显，ILABERIO机制表现最佳，其功率主要集中于2600W左右（约占总体20%），接下来是IECMP机制，其相同功耗下概率分布要优于本文提出的LoadbE和LoadbE-it机制，分析其原因，发现主要与网络的拓扑相关。在Topo2这样的全连接拓扑中，2跳的路径有24条，占4跳以内路径总数的1/7，这些路径已经能满足基本流量的传输，因此很少需要开启其他节点。而拓扑Topo1中2跳的路径只有1条，占4跳以内路径的1/11，不能满足测试流量的需求，所以造成四种算法在短路经的选择上都没有优势。并且，ILABERIO和IECMP机制会优先选择较短路径，而LoadbE和LoadbE-it机制加入了随机偏好机制，每次以概率选择下一条链路，在最终路组成路径，并不一定能选中较短的路径。

节能效率的累积分布曲线如图4.7所示，整体上看，四种机制节能效率最高可达25%左右。在相同的节能效率下，其纵坐标变化越大，代表分布越集中。在Topo1-b中最下方的ILABERIO机制表现最好，主要集中于0.16处；IECMP最差，主要集中于0.05、0.1和0.125处；LoadbE和LoadbE-it机制表现良好，LoadbE机制集中于0.05、0.1和0.16处，LoadbE-it分布于多个值之间。在Topo2-b中，节能效率集中于0.05至0.2之间，ILABERIO机制表现最好，主要集中分布于0.2左右，LoadbE-it效果最差集中分布于0.075左右，IECMP主要集中于0.16左右，LoadbE机制分布较均匀，但是其性能比LoadbE-it机制略好，比IECMP机制差。

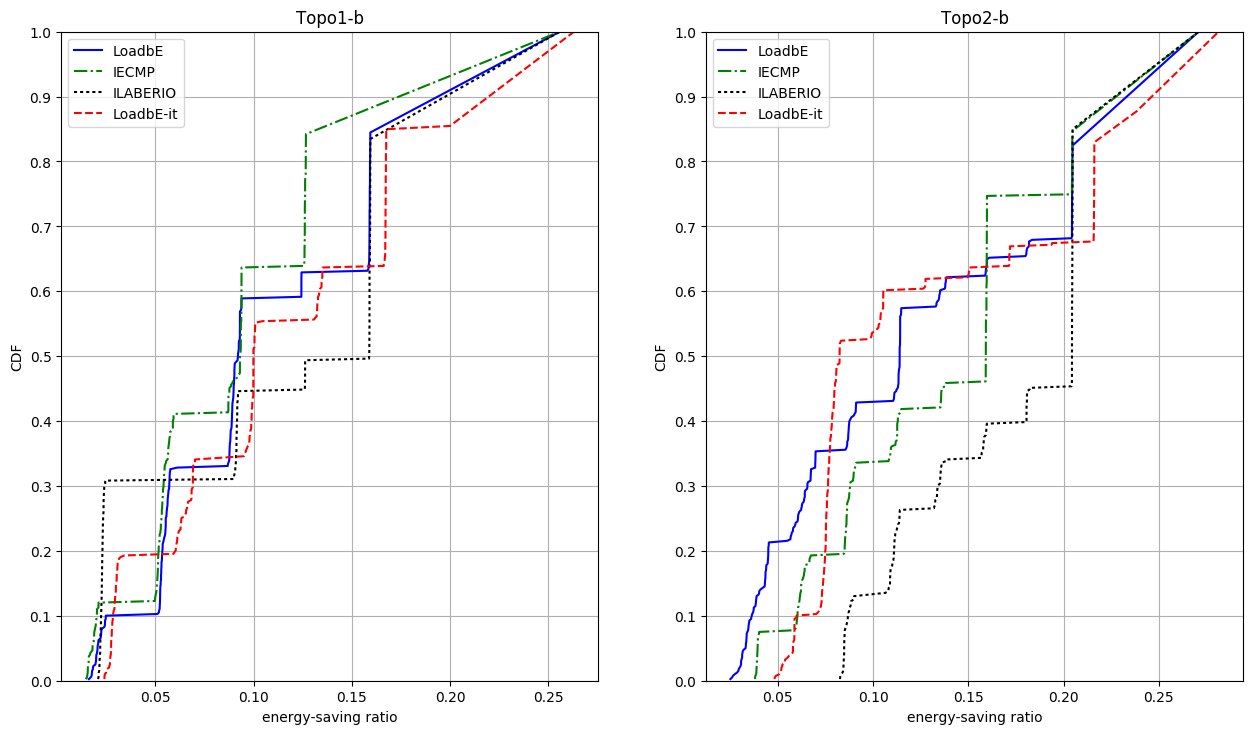


图4.7 节能效率累积分布曲线

Fig. 4.7 Cumulative distribution curve of energy-saving ratio

（3）网络性能度量

在网络性能指标上，本文从活跃链路的平均利用率、网络的吞吐量、能效、数据包平均路由时延和平均路由跳数进行度量。

1）平均链路利用率

网络中活跃链路的平均带宽利用率随时间变化曲线如图4.8所示。在Topo1-a和Topo2-a中可以看出，网络中活跃链路的平均利用率随着网络整体流量的大小而波动，当处于流量高峰时期，每条活跃链路利用率随之升高，当网络流量较为稀疏时，链路利用率随之下降，但是同时四种机制对流量有一定的聚合效果。此外，结合图4.6的Topo1-a和Topo2-a分析图4.8中出现的峰值，在20秒之前由于网络中流量的增加，导致LoadbE和LoadbE-it的平均链路利用率迅速升高，进而引起网络在20秒时刻开启新的节点和链路，促使网络能耗的升高，开启的新的链路又反过来促使了活跃链路平均额利用率的下降。同理在50秒和220秒的波峰处也是如此。

根据图Topo1-b和Topo2-b中的累积分布曲线，可以得出本文提出的机制在活跃链路平均利用率上，LoadbE和LoadbE-it机制明显高于对比机制，说明LoadbE和LoadbE-it在对数据流的聚合效果上表现更佳。其中，在对比机制中，IECMP机制性能高于ILABERIO机制。这是由于本文提出的机制，在将链路利用率转化为偏好时，转化曲线的对称轴就是网络的平均链路利用率，利用率处于对称轴附近的链路，其偏好更大，转发概率也更大。

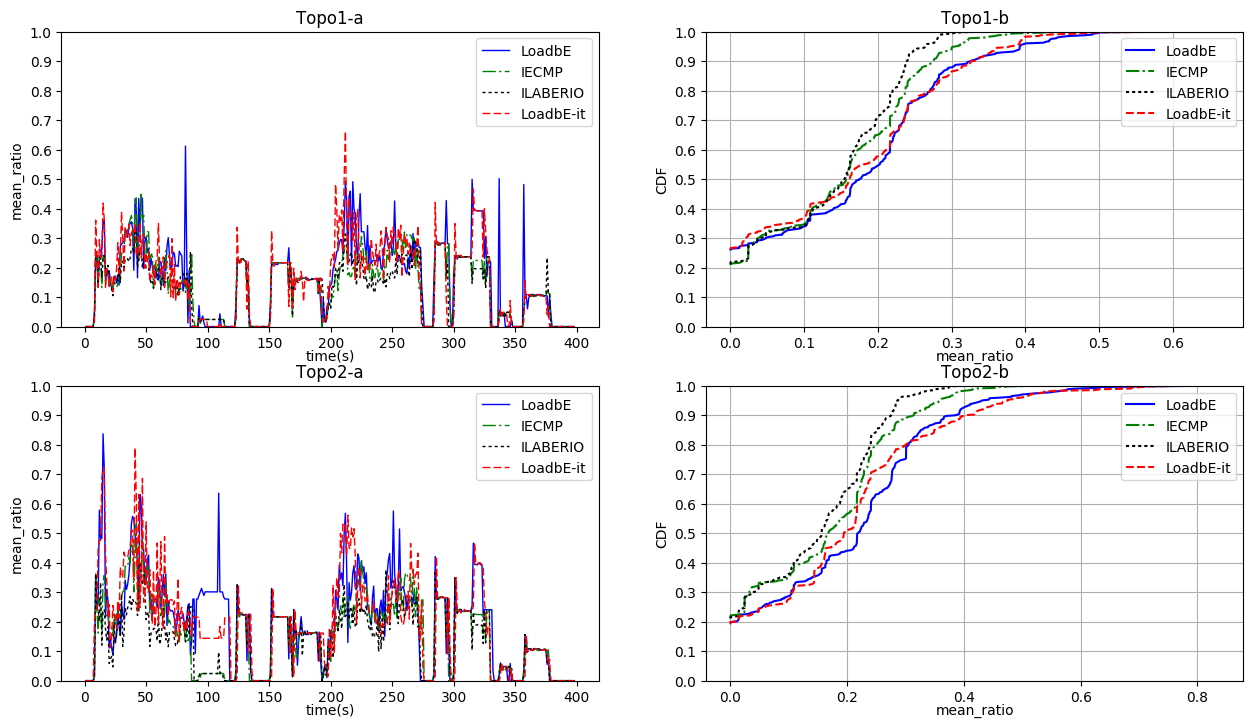


图4.8 平均链路利用率

Fig. 4.8 Average link utilization

2）吞吐量

网络吞吐量随时间变化曲线如图4.9所示。从Topo1-a和Topo2-a中可以看出在吞吐量指标上，四种机制没有明显差别，但是整体上符合吞吐量流量升高而升高，降低而降低的规律。此外，对LoadbE-it机制来说，网络处于流量高峰时期，吞吐量波动较大，这与它每次迭代式调度多个数据流有关，数据流来回调动，造成网络状态不稳定。

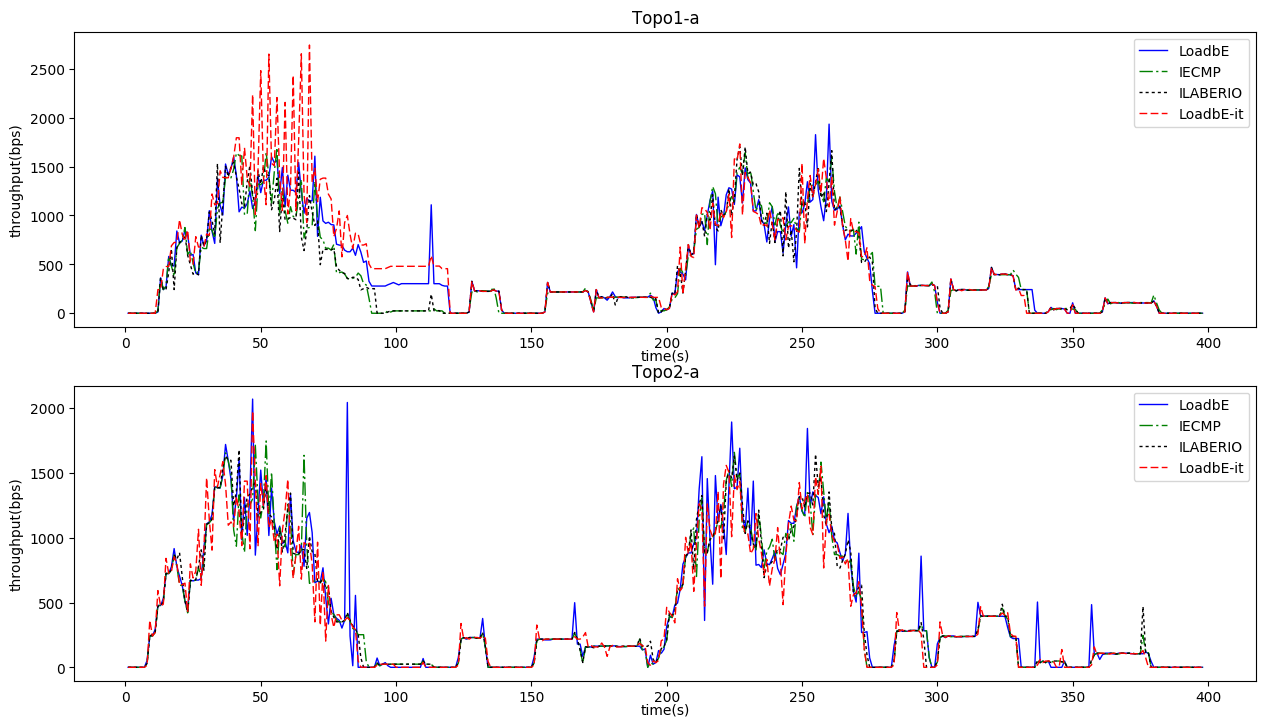


图4.9网络吞吐量

Fig. 4.9 Network throughput

对于网络吞吐量，不应该比较大小，而是应该比较与标准输入的相似度，相似度越高，代表网络处理速度越快。本文将每一秒的输出看做一个维度，将四种算法的吞吐量看做400维的向量，采用余弦相似度衡量四种算法与输入流量的相似度，结果如图4.10所示。

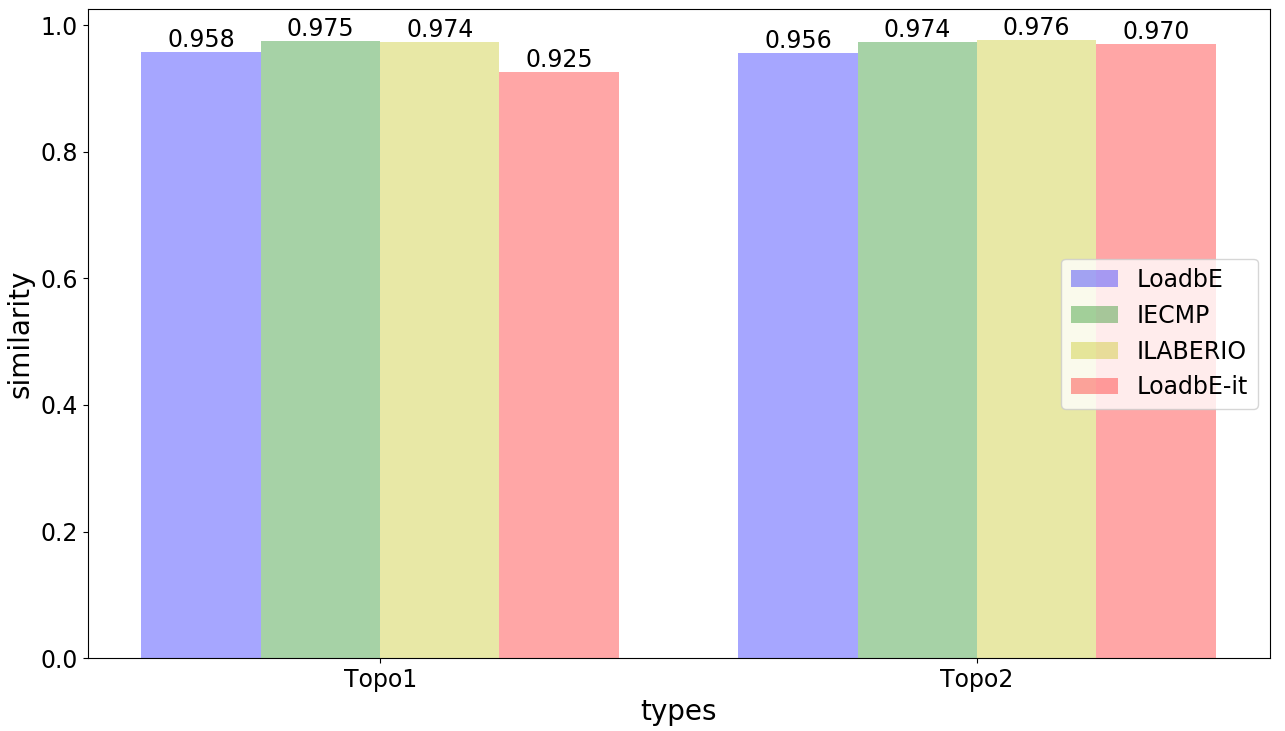


图4.10 不同机制吞吐量相似度

Fig. 4.10 Throughput similarity among different mechanisms

在图4.10中，拓扑Topo1中相似度从高到底依次为IECMP和ILABERIO、LoadbE、LoadbE-it。结合图4.9的第100秒至120秒，LoadbE和LoadbE-it机制吞吐量高于对比机制，相似度却低于对比机制，可以得出，本文提出的机制在处理数据包的效率上比对比机制差。这是因为在LoadbE和LoadbE-it中算法需要实时计算数据流路径，并且对数据流的调度较为频繁，算法复杂度高；而IECMP和ILABERIO机制采用预计算路径，在路由或调度时只需要路径替换，算法复杂度低。因此，当网络流量处于高峰时期，LoadbE和LoadbE-it控制器处理开销较大，造成一定的流量积压。拓扑Topo2中相似度从高到底依次为ILABERIO和IECMP、LoadbE-it、LoadbE。在对比机制与本文提出的机制对比上与Topo1结论类似，但是LoadbE-it机制相似度高于LoadbE机制，是由于Topo2提供了更多的路径选择，因此在迭代式流调度阶段能够找到更多合适的路径加速流量的转发，也就是说处理相同的流量，LoadbE-it用到了更多的设备，这也就意味着其功耗会很高，这点可以在图4.7的Topo2-b中得到印证。

3）能效

网络的能效为吞吐量与网络功率的比值，其随时间变化曲线和累积分布曲线如图4.11所示。从图Topo1-a和图Topo2-a上看，曲线变化趋势与吞吐量指标类似，LoadbE-it由于数据流的频繁调动造成能效波动较大。在Topo1-b的累积分布曲线中可以看出，本文提出的LoadbE-it和LoadbE机制好于对比机制，其中LoadbE-it最好，ILABERIO性能最差。这说明LoadbE-it机制虽然其频繁调度引起了吞吐量的剧烈波动，但是其总能将流调度到最合适的链路上使其能效最好。

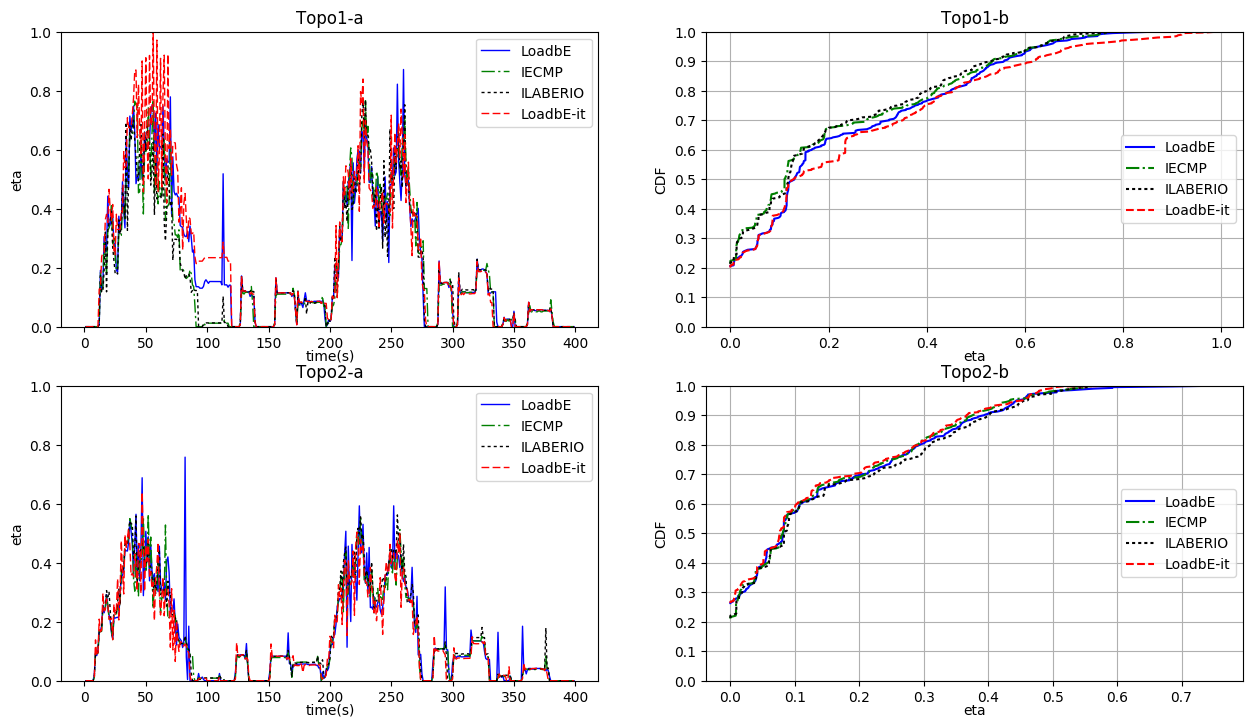


图4.11 能效

Fig. 4.11 Energy efficiency

在Topo2-b中可以看出，四种机制在拓扑Topo2中能效相差不不明显，但是与Topo1相反，能效分布最差的却是LoadbE-it机制，这是因为它在迭代式流调度时，转发了相同的流量，却使用了更多的设备，当然使得能效降低。

4）平均路由延迟

四种机制所有输入数据流的数据包的平均延迟如图4.12所示。在数据包平均延迟对比中，总的来看本文设计的机制比对比机制略差。在Topo1中，四种机制由低到高依次为：ILABERIO、IECMP、LoadbE和LoadbE-it。在Topo2中，四种机制由低到高依次为：ILABERIO、IECMP、LoadbE-it和LoadbE。本文提出的两种机制比对比机制略差。众所周知SDN中数据流的第一个数据包会被Packet in到控制器，并且需要进行一系列计算，这样会导致第一个数据包延迟非常大。在对比机制中，由于简化了控制器路径计算，提高了控制器处理的效率，因此降低了数据包的平均延迟。

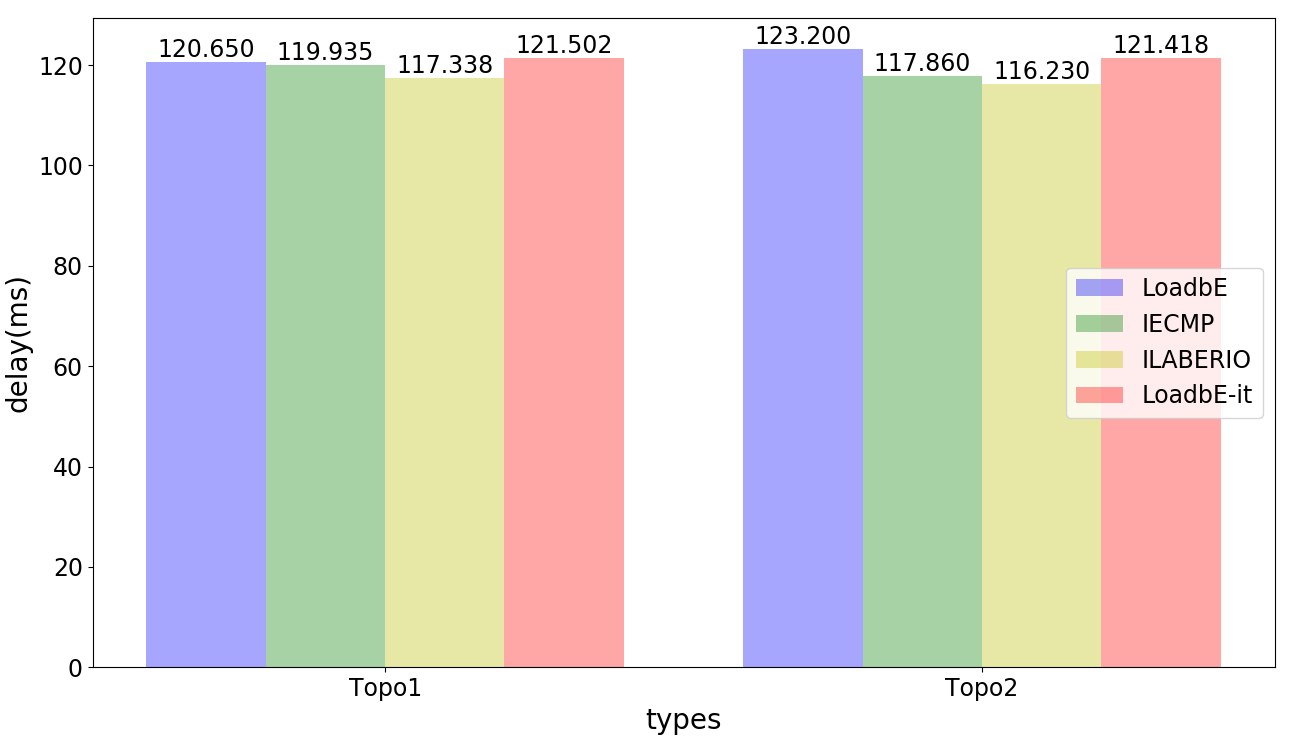


图4.12 不同机制平均延迟

Fig. 4.12 Average delay among different mechanisms

5）平均路由跳数

平均路由跳数的对比如图4.13所示，在Topo1拓扑中，ILABERIO机制平均路由跳数最低，LoadbE和LoadbE-it机制居于中间，IECMP平均路由跳数最高。这是因为在进行流调度时，在网络流量较少的时候，所有的机制可能都会选择最短路径对流量聚合，随着网络流量的升高，出现局部链路拥塞，这时ILABERIO机制使用子路径替代，跳数会增加1~2跳，而其他三种机制则使用新的路径替代导致其跳数增加较多。在Topo2拓扑中，本文提出的机制不如对比机制表现优良，ILABERIO机制平均路由跳数最低，其次，依次是IECMP、LoadbE和loadbE-it机制。这是由于Topo2提供了更多的短路经，ILABERIO机制和IECMP机制倾向选择更加有利的短路经，而LoadbE和LoadbE-it机制会先在活动拓扑上寻找路径，其次才考虑短路经，造成路由跳数增加。

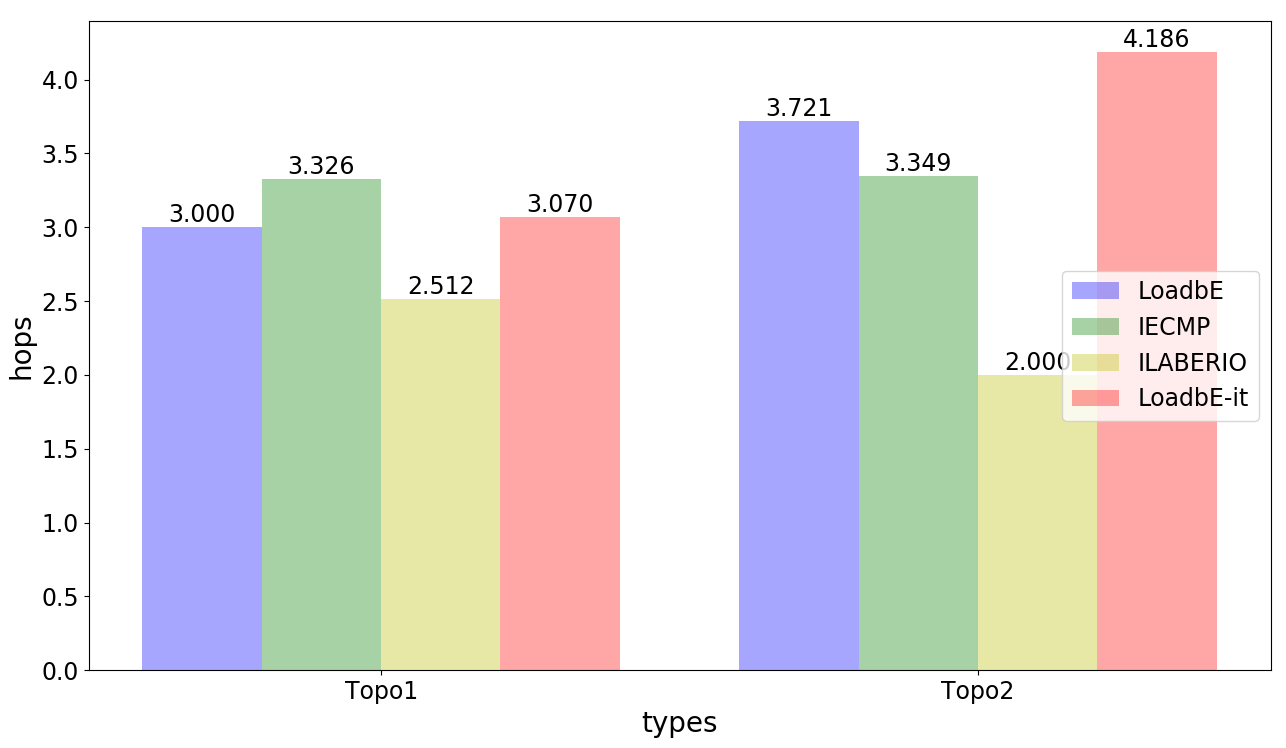


图4.13 不同机制平均路由跳数

Fig. 4.13 Average routing hops among different mechanisms

## 4.4 本章小结

本章介绍了动态负载均衡与节能机制的仿真实现与性能评价。首先，介绍了仿真实现的相关平台及工具；然后，分别介绍了控制平面和数据平面的仿真实现；最后在两种拓扑上进行了仿真实验，并从负载均衡、节能和网络性能等指标进行了对比分析，从而得出了本机制在活跃链路的平均链路利用率和利用率标准差等指标上优于对比机制的结论，证明了其可行性和有效性。

# 第5章 结束语

SDN通过将网络的控制与转发相分离对网络的硬件进行抽象，使得网络技术的革新变得更加方便快捷。本文通过对当前SDN中负载均衡与节能技术的调研，抽象出负载均衡与节能问题中的共性——数据流调度。然后针对两种不同目的的数据流调度，设计了一种同时兼顾两种因素的路由与流调度机制。本文的主要工作及取得的研究成果如下：

（1）对当前的SDN中负载均衡技术和节能技术进行了调研和分析，发现了当前针对两者的研究本质上都是对数据流进行调度，只不过调度的目的和方式不同，这为我们后续将两者结合起来研究奠定了基础。在负载均衡与节能的共同点之后，对现有的研究进行分析发现，当前针对二者的研究分为三类：只涉及负载均衡的流调度、只涉及节能的流调度和涉及负载均衡与节能但是分两阶段进行流调度。本文则在两者流调度的基础之上，将两个目标放在一个阶段解决，在路由或者流调度时同时考虑两个目标。据本文所知，本文是第一次将两者放在一个阶段同时考虑，这是本文的创新点之一。

（2）对网络节点/链路和数据流进行了建模，并且根据调研当前对网络能耗的研究，设计了节点和链路能耗模型。在此基础之上提出了“活动拓扑”和“全局拓扑”的概念。

（3）结合OpenNetMon和OpenTM的基础之上提出了同时基于数据流速率变化和网络流量的自适应数据流轮询算法，设计了流量监测机制，能够在较小的开销下对网络实施链路级别和流级别的监测。

（4）兼顾负载均衡与节能的因素设计了一种将链路利用率映射为数据流对链路的偏好的机制。首先，网络链路的偏好映射随着网络流量动态变化，其次，在同一时刻不同链路的偏好将随链路利用率动态改变。通过计算链路的偏好，从而对数据流产生促进或者抑制作用，最终实现负载均衡与节能的路由与流调度，系本文首创。

（5）在构建链路偏好矩阵的基础上，提出了基于链路偏好的随机路由算法。通过采用Softmax的思想，在数据流进行路由时，对于不同的可选链路偏好将其转化为不同被选择的概率。对于偏好小的链路，其被选择的机会较小；反之，对于偏好大的链路，其被选择概率大。这样将原来选择最佳链路的确定性的路由问题，转化为基于概率选择的路由问题。如果说确定性的路由成为“硬路由”，则基于概率的随机路由便属于“软路由”。

（6）在路由算法的基础上提出了非迭代和迭代两种流调度机制。非迭代的方式每次只调度一个数据流；迭代的方式每次调度多个数据流。在每次调度中设计了流的选择策略和路径评价方法。并且针对链路的突发流量造成的拥塞及链路的休眠状态的频繁切换等问题，设计了一种“缓冲”机制。

（7）在流表安装和更新阶段设计了两种更新策略：删除式更新和修改式更新，并且对比了两种更新方式的特点。

（8）最后使用开源控制器Ryu和Mininet仿真软件搭建了仿真平台。为了模拟网络中不同时间段的数据流量，对Mininet源代码进行了修改实现了连续2天的数据流的自动化模拟。

当然本文的设计也还存在一些问题：针对网络节点的休眠，当前的OpenFlow交换机并不支持，并且OpenFlow协议也没有设计关于交换机休眠等OpenFlow消息。因此本文的仿真实验是通过控制器对节点和链路进行映射和记录实现的。除此以外，数据流带宽估计不够准确将导致多次的调度，并且未考虑延迟和抖动约束，实验部分使用的输入流量是模拟生成，如果使用网络中真实的流量会更有说服力。

# 参考文献

1. Feamster N, Rexford J, Zegur. E. The road to SDN: an intellectual history of programmable networks[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(2): 87-98.
2. 黄韬. 软件定义网络核心原理与应用实践[M]. 人民邮电出版社, 2014.
3. Jain S, Kumar A, Mandal S, et al. B4: experience with a globally-deployed software defined WAN[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 3-14.
4. Cui L, Yu F R, Yan Q. When big data meets Software-Defined Networking: SDN for big data and big data for SDN[J]. IEEE Network, 2016, 30(1): 58-65.
5. [Xia W, Wen Y, Foh C H, et al. A survey on Software-Defined Networking[J]. IEEE Communications Surveys & Tutorials, 2015, 17(1): 27-51.](https://www.opennetworking.org/sdn-definition/)
6. Kreutz D, Ramos F M V, Verissimo P E, et al. Software-defined networking: a comprehensive survey[C]. Proceedings of the IEEE, 2015: 14-76.
7. Yin H, Jiang Y, Lin C, et al. Big data: transforming the design philosophy of future Internet[J]. IEEE Network, 2014, 28(4): 14-19.
8. Heddeghem W V, Lambert S, Lannoo B, et al. Trends in worldwide ICT electricity consumption from 2007 to 2012[J]. Computer Communications, 2014, 50(13): 64-76.
9. Gelenbe E, Caseau Y. The impact of information technology on energy consumption and carbon emissions[C]. Ubiquity, 2015: 1-15.
10. Rawat D B, Bajracharya C. Software defined networking for reducing energy consumption and carbon emission[C]. IEEE Southeastcon, 2016: 1-2.
11. Rawat D B, Reddy S. Recent advances on software defined wireless networking[J]. IEEE Southeastcon, 2016: 1-8.
12. Rawat D B, Reddy S R. Software defined networking architecture, security and energy efficiency: a survey[J]. IEEE Communications Surveys & Tutorials, 2017, 19(1): 325-346.
13. 周桐庆, 蔡志平, 夏竟等. 基于软件定义网络的流量工程[J]. 软件学报, 2016, 27(2): 394-417.
14. Akyildiz I F, Lee A, Wang P, et al. A roadmap for traffic engineering in SDN-OpenFlow networks[J]. Computer Networks, 2014, 71(3): 1-30.
15. Tootoonchian A, Ganjali Y. HyperFlow: a distributed control plane for openflow[C]. Internet Network Management Conference on Research on Enterprise Networking, 2010: 1-6.
16. Yu M, Rexford J, Freedman M J, et al. Scalable flow-based networking with DIFANE[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(4): 351-362.
17. Hu Y, Wang W, Gong X, et al. Balanceflow: controller load balancing for openflow networks[C]. Cloud Computing and Intelligent Systems, 2012: 780-785.
18. Handigol N, Seetharaman S, Flajslik M, et al. Aster\* x: load-balancing web traffic over wide-area networks[C]. Geni Engineering Conference, 2013: 1-3.
19. Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: dynamic flow scheduling for data center networks[C]. USENIX Symposium on Networked Systems Design & Implementation, 2010: 1-15.
20. Curtis A R, Kim W, Yalagandula P. Mahout: low-overhead datacenter traffic management using end-host-based elephant detection[C], IEEE INFOCOM, 2011: 1629-1637.
21. Long H, Shen Y, Guo M, et al. LABERIO: dynamic load-balanced routing in openflow-enabled networks[C]. Advanced Information Networking and Applications, 2013: 290-297.
22. Tu R, Wang X, Yang Y. Energy-saving model for SDN data centers[J]. The Journal of Supercomputing, 2014, 70(3): 1477-1495.
23. Heller B, Seetharaman S, Mahadevan P, et al. ElasticTree: saving energy in data center networks[C]. USENIX Conference on Networked Systems Design & Implementation, 2010: 249-264.
24. Li D, Shang Y, Chen C. Software defined green data center network with exclusive routing[C]. IEEE INFOCOM, 2014: 1743-1751.
25. Markiewicz A, Tran P N, Timm-Giel A. Energy consumption optimization for software defined networks considering dynamic traffic[C]. IEEE International Conference on Cloud Networking, 2014: 155-160.
26. Kakadia D, Varma V. Energy efficient data center networks-a SDN based approach[C]. IBM Collaborative Academia Research Exchange, 2012: 1-3.
27. Lara A, Kolasani A, Ramamurthy B. Network innovation using openflow: a survey[J]. IEEE Communications Surveys & Tutorials, 2014, 16(1): 493-512.
28. <https://www.opennetworking.org/software-defined-standards/>specifications/.
29. http://flowgrammable.org/sdn/openflow/message-layer/.
30. Chowdhury S R, Bari M F, Ahmed R, et al. Payless: a low cost network monitoring framework for software defined networks[C]. Network Operations and Management Symposium, 2014: 1-9.
31. Tootoonchian A, Ghobadi M, Ganjali Y. OpenTM: traffic matrix estimator for OpenFlow networks[C]. International Conference on Passive and Active Network Measurement, 2010: 201-210.
32. Yu M, Jose L, Miao R. Software defined traffic measurement with OpenSketch[C]. USENIX Conference on Networked Systems Design and Implementation. 2013: 29-42.
33. Van Adrichem N L M, Doerr C, Kuipers F A. OpenNetMon: network monitoring in openflow software-defined networks[C]. Network Operations and Management Symposium, 2014: 1-8.
34. Yu C, Lumezanu C, Zhang Y, et al. FlowSense: monitoring network utilization with zero measurement cost[C]. International Conference on Passive & Active Network Measurement, 2013: 31-41.
35. Suh J, Kwon T T, Dixon C, et al. OpenSample: a low-latency, sampling-based measurement platform for commodity sdn[C]. IEEE International Conference on Distributed Computing Systems, 2014: 228-237.
36. Benson T, Anand A, Akella A, et al. MicroTE: fine grained traffic engineering for data centers[C]. The Seventh Conference on Emerging Networking Experiments and Technologies, 2011: 1-12.
37. Li D, Yu Y, He W, et al. Willow: saving data center network energy for network-limited flows[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(9): 2610-2620.
38. Bianzino A P, Chaudet C, Rossi D, et al. A survey of green networking research[J]. IEEE Communications Surveys & Tutorials, 2012, 14(1): 3-20.
39. Adami D, Antichi G, Garroppo R G, et al. Towards an SDN network control application for differentiated traffic routing[C], IEEE International Conference on Communications, 2015: 5827-5832.

# 致 谢

时光荏苒，转眼间两年半的研究生生涯即将画上句点。回顾自己在研究生期间经历的点点滴滴，一下子涌上心头，竟一时不知从何说起。从接到录取通知书那一刻，就注定了我与东北大学的不解之缘。

首先我要感谢东北大学，感谢它为我提供了这样一个平台，让我在这两年半里能与众多优秀的老师和同学相遇，感谢它让我收获了同窗同门之谊，也感谢它成为我步入职场的最后一个充电站，让我收获到丰富的科研经历。在东北大学学习的这段经历必将影响我的一生，必将是我人生路上最宝贵的财富之一。

其次，我要衷心感谢我导师王兴伟教授。王老师严谨的科研态度对我产生了深远影响。“一劳永逸地解决问题”、“楞看、楞听、楞想、楞记”、“做点感动上帝的事”，这些凝聚王老师多年感悟的话语让我深受触动，“教书育人”在王老师身上体现得淋漓尽致，也正是受到王老师的影响改正了生活中的很多陋习。感谢王老师作为我人生路上一盏明灯，让我对自己对未来有了更加清晰的认识，您的谆谆教导，将始终牢记在我心中，伴我继续前行。同时感谢课题组的其他老师和同学。感谢李福亮老师的指导帮助让我对SDN的环境平台有了更深一步的认识。感谢吕建辉师兄、张闯闯师兄、何强师兄和赵勇博士在我毕业设计论文撰写阶段给予的帮助。特别要感谢张闯闯师兄，作为博士师兄的他不仅仅是我科研路上的领路人，他更像兄长一样关心照顾着我，每次当我遇到难以解决的问题时，他总是能给我最及时的帮助，准确的诠释了“师兄”的含义。感谢潘石、薛惠中、卢鹏飞、吴思锦、周宇、白芳、任可欣、裴丽雅、郭静等实验室同学，感谢你们在我遇到困难踌躇不前时的鼓励，我的每一点滴的成长都离不开你们的帮助。

然后感谢我的舍友和挚友胡金林、李东阳、黄希文和贺旭等，忘不了你们陪伴的日日夜夜，忘不了我们每天晚上在宿舍讨论分享各种面试难题、忘不了一起在异地网吧参加笔试，感谢大家。在秋招的岗位争夺战中我们互相鼓励，一起拼搏，一起战斗，我们是亲密的战友和兄弟，祝愿大家都拥有美好的人生。

接下来感谢我的父母、亲人，感谢您们对我的关心和照顾，感谢您们在我研究生期间给与的理解支持与鼓励。您们是我的坚强后盾，您们永远是我最爱的人！

最后感谢答辩组的各位老师，感谢你们辛勤付出，感谢你们在答辩过程中严谨的科研态度，感谢你们的宝贵意见！

# 攻读硕士学位期间发表的论文

1. 马连博, **胡书培**, 王兴伟等. 小生境粒子群优化ABC支持型QoS组播路由机制[J]. 华中科技大学学报(自然科学版), 2016, 44(11):97-102.