密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：基于 SDN 负载均衡技术的研究与实现

学号：2015140655

姓名：孙雪松

专业：软件工程

导师：王东滨

学院：软件学院

年　　月　　日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。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了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

基于 SDN 负载均衡技术的研究与实现

摘 要

随着互联网的迅速发展，网络流量日益增大，数据中心面临的流量压力也不断增加。由于传统网络架构无法对网络设备进行全局性管控，因此容易造成节点负载不均，导致网络服务质量下降。软件定义网络（SDN, Software Defined Network）将传统网络设备的转发层与控制层分离，利用控制层实现对网络的集中式管理。

SDN 架构为数据中心的负载均衡提供了良好的思路。针对数据中心内网络节点负载不均的问题，有人提出通过单链路指标评价路径进行流量的均衡，为了更好的评价路径，本文综合评价了路径的多个指标，提出了基于链路多指标的最小花费负载均衡（LCLB, Least Cost Load Balancing）算法，降低了链路中负载的抖动率，达到了负载均衡效果。

本文设计与实现了控制器中的链路发现模块、拓扑管理模块、拓扑计算模块、信息统计模块、负载均衡模块及流表下发模块, 并在此基础上研究和设计了基于 SDN 的 LCLB 算法策略。通过拓扑计算模块求得主机间的所有路径，利用统计模块对链路状态进行实时监控，在负载均衡模块中综合考量整个路径中的链路指标值并根据链路状态获取最小花费值的路径，避免路径因单个链路产生的拥塞情况。通过多次实验表明，本文的算法降低了链路负载抖动率和丢包率，能够有效缓解网络拥塞，具有良好的负载均衡效果。

关键词：软件定义网络 数据中心 负载均衡 OpenFlow

THE RESEACH AND IMPLEMENTATION OF LOAD BALANCING TECHNOLOGY ON SDN

ABSTRACT

With the rapid development of the Internet, increasing network traffic, data center traffic pressure is also increasing. As the traditional network architecture can not control the overall network equipment, it is likely to cause node load uneven, resulting in decreased network service quality. Software Define Network (SDN) protects the forwarding layer of the traditional network device from the control layer, and uses the control layer to realize the centralized management of the network.

The SDN architecture provides a good idea for load balancing in the data center. And aiming at the problem of uneven load of network nodes in the data center, it is proposed to balance the traffic through the single link index evaluation path. In order to better evaluate the path, this paper comprehensively evaluates the multiple indexes of the path and proposes a multi-algorithm (LCLB, Least Cost Load Balancing),which reduces the jitter rate of the load in the link and achieves the load balancing effect.

This paper designs and implements the link discovery module, topology management module, topology calculation module, information statistics module, load balancing module and flow table module in the controller and designs the strategy of LCLB based on SDN. Through the topology calculation module, all the paths between the hosts are obtained, and the link state is monitored in real time by the statistical module. In the load balancing module, the link index value in the whole path is taken into account and the minimum cost value is obtained according to the link state. Avoid congestion due to a single link. The experimental results show that the proposed algorithm can reduce the link load jitter rate and packet loss rate, and can effectively alleviate network congestion and have good load balancing effect.

KEY WORDS: Software Defined Network, data center, load balancing, OpenFlow

目录

第一章 绪论

1.1 课题研究背景及意义

自上个世纪四十年代中期计算机诞生起至今，在短短七十年间计算机从专业实验室配备的昂贵设备发展到与我们的日常生活息息相关的产品，互联网的诞生与普及对加速了计算机的发展进程起着至关重要的作用。据中国互联网网络信息中心的相关数据[1],截止到 2016 年 6 月，中国网民的数量为 7.1 亿，互联网普及率达到 51.1%，这一数字仍将保持高速增长。一根网线连接地球已经成为了现实，互联网在给人类带来极大便利的同时也使我们对互联网的依赖更加加深。办公、社交、娱乐、购物乃至于几乎日常生活的一切都可以通过网络来解决。随着网络使用的普遍，网络中的访问量和数据流量呈现出爆炸式的增长，这种爆发式的增长势必使得现有网络架构拓展的速度无法满足用户的性能需求。因此，如何合理分配网络流量、增强网络架构的可扩展能力、充分利用网络带宽及计算资源为用户提供更加流畅的网络使用体验是目前面临的重要问题。

传统网络的设计目标是单纯地实现端到端的数据传送，由于互联网中几乎全部的流量都是建立在 TCP/IP 架构之上，在这种情况下网络本身的架构不会随着设备性能的提高而产生突破性的变化。正是因为这种网络架构的限制，新的网络技术大多都是以“打补丁”的方式进行修补，这一根本的弊端导致了简单的网络设备变得越来越复杂、网络也越来越臃肿。而且，由于厂商设备的封闭性，导致网络管理的灵活性大大降低，并且高昂的研发成本和设备门槛的限制也使得网络的升级变得异常困难。为了推进互联网技术的创新发展，打破设备的封闭性，Nick McKeown 等人[2]提出软件定义网络（SDN,Software Defined Network）这一概念，并首次将 OpenFlow 协议成功用于校园网络的创新中。由于 SDN 在实验中的良好表现使得人们相信这是一种前瞻性的技术，可以更加便利的管理网络并解决现有网络架构中出现的臃肿等一系列问题。相较于传统网络，SDN 具有两大突出特点——数据平面与控制平面分离以及网络的可编程性。正是在这两大特点的支持下使得整个网络更加开放并能够适应不断更新的业务需求，给他们独立发展带来了很高的灵活性。控制器和转发完全分离的这种设计使得控制器专注于控制决策，交换机专注于转发工作，数据、状态及控制信息通过 OpenFlow 这一标准接口实时地在控制器与交换机之间传递。目

前，在众多企业、高校及设备厂商的研究、支持下，SDN 技术得到了迅猛的发展。

负载均衡是解决网络服务质量的重要方法之一，其原理是将发来的请求按照某种算法将其分担到网络的各个节点中从而能够均匀使用网络中的资源，不会因某些节点太过拥塞而导致服务质量的降低。对大流量的数据中心网络来说，最大化吞吐量和最小化网络时延是保证服务质量的重要指标。在传统 TCP/UDP 的网络中，通常使用静态交换机来实现负载均衡。流中的每一包在链路中都按照相同的权重以预先设置好的路径进行转发，随着网络的增大，这一策略会变得笨重且低效。而将 SDN 技术与负载均衡相结合，利用 SDN 的集中控制对整个网络进行管控，不仅可以使负载均衡的成本得到有效控制，而且也可以使整个网络变得更加灵活有效。由于 SDN 与传统网络的存在的差异，必然会导致传统的负载均衡技术不能完全适应于 SDN 中，因此如何更好的利用 SDN 来实现负载均衡技术也是目前急需解决的问题之一。

1.2 国内外研究现状

SDN 作为一种新型的网络架构，将网络中的控制平面和数据平面互相分离，对分布式的网络提供了集中控制式的管理，从而为传统网络中存在的问题的解决提供了新的思路。

2005 年，为了加快推动网络创新的发展，美国国家自然科学基金会（National Science Foundation，NSF）资助启动了全球网络创新实验环境（Global Enviroment Networking Innovations,GINI）[3]计划，2006 年由斯坦福大学主导的 Clean-Slate[4]项目，参与者 Martin Casado 从中吸取经验，并在其领导的 Ethane[5]项目中将控制与转发完全解耦，控制器通过 Pol-Ethane 语言向交换机分发策略，形成了 SDN 的雏形。2008 年，Nick McKeown 的校园网络创新实验获得成功，并提出 OpenFlow 协议。OpenFlow 协议作为控制平面和转发平面之间的交互协议可以实现网络中控制和转发的分离，使数据层和控制层无需关注对方的工作，从而使得网络编程能力得到极大的提升。2013 年，在 Google 的 B4[6]网络中，利用 SDN 将链路带宽利用率提高了 3 倍以上，接近 100%，大大降低了链路成本、提升了网络的稳定性。这一成果充分证明了 SDN 技术具有解决现有网络一系列棘手问题的能力，超出业界对最初的期望；随着 Google 的 B4 网络的成功落地，让业界增加了对 SDN 作为商业项目使用可行性的信心。

随着 SDN 的不断发展，在 SDN 网络中解决传统网络的各类问题也成为业内研究的热点。随着互联网的飞速发展，大规模的流量充斥在互联网中，人们对网络数据的需求也在急速增长，如何高效的使用计算资源和网络链路等都在

学术界被广泛关注。传统的负载均衡方法主要包括网络地址转化负载均衡技术、反向代理技术的应用服务器负载均衡技术以及基于 RR-DNS 的负载均衡技术。而在 SDN 网络中，传统的负载均衡技术会适应 SDN 技术而做出改进，从而更好的实现网络资源的最优化。在基于广域网和数据中心网络之间均衡问题的最著名的例子就是上文中提到的 Google 的 B4 网络和 Microsoft 的 SWAN 控制器[7]。后者除了具有高链路利用率以外，还能够使用非定制的商业交换机，这一改变能在很大程度上控制成本。然而其缺憾在于二者对流量的需求是已知的，很难应对未知流量，并且对小规模或数据中心内部的网络而言，其消耗的代价过高。Handigol N 等人中提出的 plug-n-Serve 方法（现已改名为 Aster*x）[8]分别监测服务器和网络的工作负载并对这些信息进行反馈，根据反馈的实时的负载信息，对 OpenFlow 规则进行重定义，从而将用户请求按照服务器的能力进行调整和分发，有效的减少了服务的响应时间。Wang R[9]等人在论文中利用发送请求的客户端 IP 地址前缀作为通配符形成规则并最小化规则集，交换机在规则集的指导下进行转发，从而减少控制器的干预次数，降低请求的平均延时。Niranjan Mysore R[10]提出通过模拟退火算法完成对数据中心网络负载进行均衡的算法，该算法主要利用与 PortLand[11]数据中心网络中。Li Y[12]提出了基于数据中心的胖树网络中实行的动态负载均衡算法（Dynamic Load Balancing，DLB）。通过端口流量信息监控和数据流的调度，统计出链路的负载情况，利用单跳贪心算法进行选路，将当前最小负载链路添加到流的路径中达到较高的网络传输率，从而实现负载均衡。

1.3 本文的主要工作

本文设计的是基于胖树拓扑[13]的链路多指标综合评估的负载均衡策略，借助于 SDN 中的控制器对链路状态进行监测，利用胖树网络特性维护路径，充分考虑链路传输数据的复杂性和多样性，通过实验方法改变相关指标权重；根据链路指标、权重，对路径中链路花费进行动态计算，从而为流量选择一条最优路径，从而实现网络的负载均衡，提高网络传输效率。

主要的研究工作如下：

（1） 总结目前存在的基于 SDN 的负载均衡技术的实现方法，分析其各自的优势和技术缺陷，通过对链路中指标的综合评估，设计一种基于数据中心网络胖树拓扑的负载均衡算法。

（2） 基于控制器开发并实现，了解控制器的架构，对控制器的代码进行梳理、解构。对实现负载均衡所涉及的链路发现模块、拓扑管理模块、拓扑计算模块、统计模块以及流表下发模块开发和修改。按照

胖树拓扑特征在控制器中改造路径计算模块，增加负载均衡模块，在此基础上进行负载均衡的设计与实现。

1.4 论文组织架构

负载均衡是计算机网络中一种流量均衡技术，通过这一技术可以避免某些节点因太多请求而形成拥塞，导致高延时、高丢包率等，从而影响服务质量。在 SDN 技术出现前，大部分数据中心网络主要依靠硬件来实施负载均衡操作，这些硬件负载均衡虽然能够达到高吞吐率和低延时率，但是因为其成本过高且可操作性比较弱。而传统的软件方法所提供的负载均衡技术虽然成本低但因为处理时间过长，灵活性低并不能很好的实现负载的均衡。随着 OpenFlow 技术的出现，我们旨在利用 SDN 网络的优势，结合软件负载均衡方法，在流量的传输过程中实现网络中的负载均衡。

第一章为绪论，首先介绍了课题的研究背景与意义，并对 SDN 技术的发展做了进一步的介绍。对国内外提出的负载均衡技术进行的简单的介绍与分析。论述了本文的主要工作。

第二章主要介绍了本文相关的技术研究，首先描述了 SDN 的架构及 OpenFlow 工作原理，分别从 OpenFlow 技术的发展、OpenFlow 协议进行研究；接着详细介绍了 OpenFlow 交换机和主流的 SDN 控制器，并分析了控制器的特点；通过对 Minint 和性能测试工具 Iperf 的简要概述，介绍了本文中实验的仿真平台。

第三章主要介绍了本文中提出的基于 SDN 的负载均衡算法——LCLB 算法。结合了算法的应用场景及其他相关的基于 SDN 的负载均衡算法，提出了本文中的负载均衡算法并对此算法进行了详细的描述，提出负载均衡评价指标，利用此指标分析实验结果。

第四章主要阐述基于 SDN 负载均衡的设计细节和实现部分。详细分析所用核心算法及相关模块。分析负载均衡涉及到的模块，详细介绍如何对控制器内模块进行修改。

第五章主要对本文提出的负载均衡策略进行验证，并分析实验结果。

第六章总结全文，并对未来工作进行展望。分析本文工作的优点和不足之处，并据此提出下一步研究的方向。

第二章 相关技术介绍

本章将介绍了 SDN 的基本概念、相关的 Openflow 技术以及不同类型的 SDN 控制器，分析控制器的特点并选择本文算法中将要使用的控制器。介绍 Mininet 仿真平台及其内置的网络性能测量工具 Iperf。

2.1 SDN 与 OpenFlow 协议

2.1.1 SDN 框架结构

SDN 即软件定义网络，是一种新型网络创新架构。不同于传统网络， SDN 不仅实现了转发和控制的分离，还令网络具有了可编程性。它实现了网络设备的控制层面和数据层面的互相分离，将控制平面集中实现，使得控制平面专注于决策，数据平面专注于数据的传输。除此之外，SDN 可以使用户通过其提供的强大的编程接口对网络资源进行定制，从而满足不用用户的不同需求。从 ONF 到 SDN 的发展来看，SDN 不仅重构了网络的系统功能，实现了数控分离，也对网络资源进行了抽象，建立了新的网络抽象模型。SDN 主要有三个特征，只要符合以下三个特征的网络都可以称之为软件定义网络。在这三个特征中，控制平面和数据平面互相分离为逻辑集中控制创造了条件，逻辑集中控制又为开放可编程控制提供了可能，因此网络的开放可编程才是 SDN 的真正核心特征，

（1） 网络开放可编程：SDN 建立了新的网络抽象模型，为用户提供了一套完整的通用 API，使用户可以在控制器上编程实现对网络的配置、控制和管理，从而加快网络业务部署的进程。

（2） 控制平面和数据平面的分离：此处的分离是指控制平面与数据平面的解耦合。控制平面和数据平面之间不再互相依赖，两者可以独立完成体系结构的演进，类似于计算机工业的 Wintel 模式，双方只需要遵守统一的开放接口通信协议即可。控制平面与数据平面的分离是 SDN 架构区别于传统网络体系结构的重要标志，是网络获得更多可编程能力的架构基础。

（3） 逻辑上的集中控制：主要是指对分布式网络状态的集中统一式管理。在 SDN 架构中，控制器会担负起收集和管理所有网络状态信息的重任。逻辑集中控制为软件编程网络功能提供了架构基础，也为网络自动化管理提供可能。

SDN 的基本架构如图 2-1 所示，分为应用层、控制层和基础设施层。在 SDN 中，集中式的控制平面利用控制-转发通信接口对分布式的转发平面中的网络设备进行控制，并且向上可以提供灵活的可编程能力[14]。控制器通过控制-转发通信接口对交换机进行集中控制，这部分控制信令的流量是独立于终端间的通信数据流量，发生在控制器和交换机之间。交换机接收到控制信令后生成转发表，并根据转发表决定相应数据流量的后续处理。

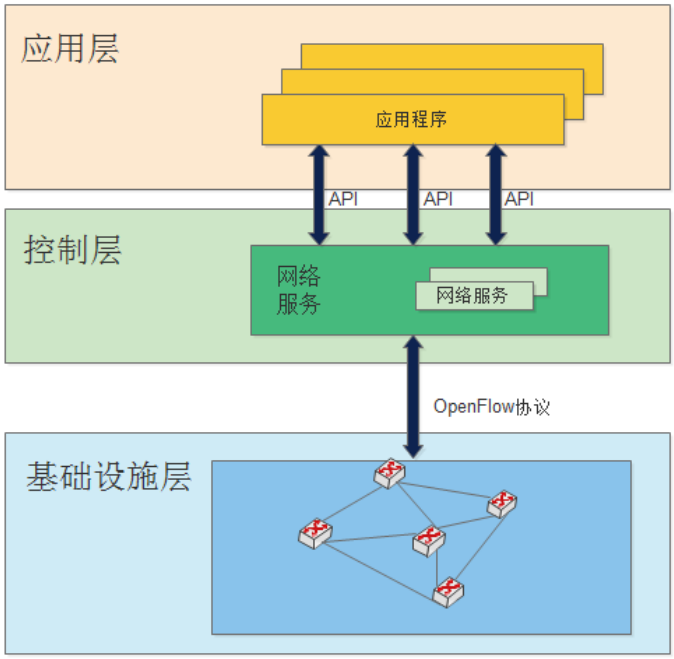

图 2-1 sdn 架构图

SDN 发展的期间，各大厂商对 SDN 的架构都有自己的理解和实现。但 ONF（Open Network Foundation）[15]作为 SDN 最重要的标准化组织者，它提出的架构对 SDN 的发展产生了深远的影响。在 ONF 提出的 SDN 网络架构中，将网络分为三层，分别为应用层、控制层和数据层。其中应用层为最上层，负责为用户提供服务；中间层为控制层，负责网络的调度；最下层为数据层，负责数据流量的传输。应用层与控制层之间的接口被称为北向接口，控制层与数据层之间的接口被称为南向接口。

（1）数据层

数据层即基础设施层，由若干个网络单元（Network Element）构成，是一个被管理的资源在逻辑上的抽象集合。数据层中的设备通过南向接口接收来自于控制层的命令，根据控制层的决策来执行相应的动作。

（2）控制层

SDN 的控制层就是 SDN 中的控制器所在的网络平面，SDN 控制器是一个逻辑上的实体，它连接了底层设备和上层服务，能够将应用层的请求发送到数

据层并为 SDN 应用层提供底层网络的抽象描述。SDN 控制层中可包含一个或者多个 SDN 控制器，每个 SDN 控制器包含控制数据平面接口驱动、SDN 控制逻辑以及北向接口代理。

（3）应用层

SDN 的应用层也称为业务层，是由若干个 SDN 的应用构成，是 SDN 中的最上层结构，主要为用户提供服务，用户可以通过应用层为网络添加拥塞控制、防火墙等服务。应用层的主要作用是为用户提供服务开发人员可以通过北向接口与控制器进行交互，进行模块的开发，以实现不同的用户需求。随着 SDN 的不断发展，专注于上层应用的开发也越来越多，为网络的多样化提供了业务服务。

（4）SDN 南向接口

SDN 南向接口主要用于网络设备与控制器的通讯，允许控制器在硬件上安装控制平面决策从而控制数据平面，对数据平面的转发行为进行控制、统计、报告登吧。南向接口可以屏蔽网络设备的差异，使控制器管理网络设备时不因设备的差异而产生消耗。因此南向接口的统一意味着实现软件与硬件的完全解耦，这在 SDN 技术的发展与应用中将起着重要的作用。

目前，由 ONF 组织开发并维护的 OpenFlow 协议是最具发展前景的南向接口协议，他们致力于将南向接口标准化与统一化；除此之外其他的厂家和组织也根据他们对 SDN 的理解推出了自己的南向接口协议，例如 Juniper 公司提出的基于 XMPP 的 OpenContrail[16]南向接口协议等。

（5）SDN 北向接口

SDN 北向接口主要负责提供抽象的网络视图从而使业务应用能够便利地调用底层的网络资源。通过北向接口，开发人员能以软件编程的形式调用其所需的网络资源；同时，上层的管理系统可以通过北向接口获悉整个网络中资源的状态，并通过控制器实现对资源的统一调度。由于北向接口直接面向业务应用服务，故其设计需求也必须密切联系业务应用，保持特征多样化，因此，北向接口的设计尤其需要合理、便捷，从而被业务应用广泛调用。然而要抽象出一套适合所有应用场景的北向接口从目前来看基本是不可能完成的，因此该接口尚无标准化。

2.1.2 OpenFlow 协议

OpenFlow 诞生于 2008 年 Nick McKeown 教授的论文“OpenFlow：Enabling Innovation in Campus Networks”[17]中，并利用于斯坦福大学的校园网实验创新中。OpenFlow 协议是为了简化 Ethane 项目中的交换机的设计而被提出，它是一个控制平面和数据平面之间的交互协议，将网络设备中的控制平面和数据平

面完全分离。与传统网络相比，OpenFlow 的控制器维护着网络拓扑、监控网络流量等，所有的决策都由控制器指定并通过 OpenFlow 协议下发到交换机，而所有的交换机仅仅根据控制器下发的指令进行操作，完全不关心网络中的任何情况。2011 年 ONF 组织成立后，在 ONF 的大力推动下，经过多年发展，OpenFlow 协议成为当前最主流的南向接口协议之一。

OpenFlow 是一种基于流的协议，所谓流是将通信中产生的共同特征的数据分组抽象而成，使得网络设备能对其进行统一的操作。这一改进将会比传统网络单独处理数据分组的效率有了很大的提高。OpenFlow 基于流的操作使得 OpenFlow 控制器将会根据流中第一个数据分组的特征，在 OpenFlow 交换机上部署其下发的策略。OpenFlow 交换机收到信令后将置于流表中进行维护，用于以后的转发之中。这一操作实现了灵活的网络转发策略，达到了软件与硬件解耦的目的。OpenFlow 架构图如图 2-2 所示，OpenFlow 网络的主要包括 OpenFlow 交换机、Openflow 协议以及控制器。其中，OpenFow 交换机依据其所维护的流表进行转发工作，控制器在南向接口中通过 OpenFlow 协议与交换机进行交互，实现控制功能。2009 年 12 月，ONF 组织发布了第一个较为成熟的版本，OpenFlow v1.0，在随后的几年中又更新了几个版本，到目前为止发布到 OpenFlow v1.5，并且协议还在不断的更新中。其中 OpenFlow v1.3 是 ONF 组织承诺的稳定版本，并将对其进行长期维护。在 v1.3 版本中，提供了包括 IPv6 扩展头、服务质量等在内的更丰富的特性，目前大多数 OpenFlow 交换机都实现了对 v1.3 版本的支持，也出现了很多基于 OpenFlow v1.3 的商业支持。

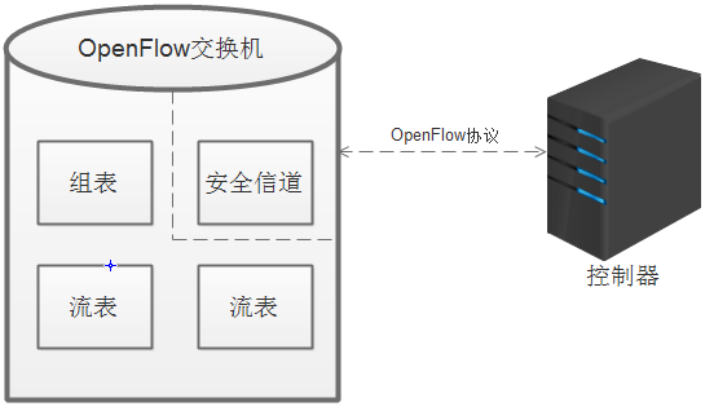

图 2-2 OpenFlow 架构图

2.2 OpenFlow 交换机

OpenFlow 交换机[18]包含流表和安全信道两个部分。流表就是用来存放流表项，在 OpenFlow 协议中控制器通过下发流表至交换机来决定交换机如何处理到来的数据包。安控制器与交换机通过安全信道实现安全连接。通常情况下，安全信道可以直接建立在 TCP 之上，也可以基于 TLS 加密之后的 Socket 建立。

在 OpenFlow 交换机和控制器连接的初始化阶段，需要将自身支持的特性和端口描述等信息上报纸控制器中。当数据包从入端口进入交换机时，交换机将提取数据包相关的特征信息，并将其与流表信息进行匹配。若匹配成功，就按照流表项中的动作执行；若匹配失败，交换机将数据包封装成 Packet_In 报文上报至控制器。当控制器接收这一报文后会选择下发 Packet_Out 报文或下发流表项等方式告知交换机如何处理这一数据流。因此，在 OpenFlow 协议架构中，控制器是策略的决策者，交换机是策略的执行者。目前 OpenFlwo 中又新增了组表和 Meter 表。

目前，支持 OpenFlow 协议的设备分为 OpenFlow 交换机和支持 OpenFlow 协议的交换机[19]。前者支持 OpenFlow 协议栈，后者同时支持 OpenFlow 协议栈和传统的网络协议两种运行方式。

（1） 流表

流表是 OpenFlow 交换机中一个非常重要的组成部分，在 OpenFlow 网络中交换机根据流表中的信息来决定如何处理到达交换机中的数据包。因为单流表支持的逻辑程序太简单，从 OpenFlow1.1 开始支持多级流表这一概念。多级流表将处理逻辑分为多个子逻辑，从而使数据包处理变成一条流水线。多级流表的设计使流表项聚合变成可能，节约了流表空间，提高了编程处理逻辑的灵活性。每一条流表项包括匹配域、指令集和计数器三个部分，如图 2-3 所示。在交换机中每个流表中包含多个流表项，流表项用以匹配数据包。当 OpenFlow 交换机接收到数据包后，交换机开始执行查找流表的操作，将数据匹配字段从数据包中提取出来，按照优先级的高低从每个表的第一个匹配表项开始匹配，若找到相匹配的项，那么将数据包按照匹配项中的指令进行操作，否则将取决于漏表的流表项配置。

图 2-3 流表结构示意图

匹配字段：对数据包进行匹配。包括入口端口、数据包报头及由前一个表制定的可选的元数据。

优先级：流表项的匹配顺序。

计数器：更新匹配数据包的计数。

指令：修改行动集或流水线处理。

计时器：最大时间计数或流有效时间。

Cookie：由被控制器选出的不可见的数据值来表示，用来进行对流的处理操作。

（2） 组表

组表这一概念是 OpenFlow1.1 中提出来的，组表示的是一组泛洪的指令集及更复杂的转发操作且这组动作可被多条流表项使用，从而实现组播、负载均衡、容灾备份等，其结构如图 2-4 所示。例如当交换机需要将发送到端口 2 中的数据都弹出 VLAN 标签，那么可以通过组表创建一个”ALL”类型的动作桶，然后将弹出 VLAN 动作和转发到端口 2 动作放入其中。如此，所有需要转发到端口 2 的流表项的动作集就只需跳转到对应的组表项，然后执行组表项内动作桶的动作即可。组表的存在降低了流表项的逻辑复杂度，也减少了流表存储空间。

图 2-4 组表结构示意图

（3） Meter 表

Meter 表用于计量和限速，其结构如图 2-5 所示。Meter 表项可以针对流制定对应的限速等规则，从而实现丰富的 QoS 功能。Meter 表和端口队列不同，Meter 表项是面向流而非面向端口，所以更细致、更灵活。

图 2-5 Meter 表结构示意图

（4） OpenFlow 安全信道

OpenFlow 的安全信道是控制器与交换机之间的通信接口。控制器可以通过这一接口获取到交换机的消息并对交换机进行控制和管理。交换机和控制器之间的交互消息被封装为 OpenFlow 协议中规定的格式在二者之间进行传输。交互消息支持三种类型。controller-to-switch——控制器初始化并发送给交换机的报文类型，其可能要求交换机回复对应报文；asynchronous——由交换机异步发送给交换机的报文类型，无需等待控制器请求。交换机通过异步报文告知控制器新数据包的到达和交换机状态的改变；symmetric——交换机或控制器发起，无须得到对方的许可或邀请，用以建立二者之间的连接。

（5） OpenFlow 端口

OpenFlow 端口是控制器处理进程及网络中传递数据包的网络接口。在 OpenFlow 交换机间利用 OpenFlow 端口在逻辑上进行互连。在 OpenFlow 网络中，入端口是数据包的属性，数据包从入口端口进入，根据流表中的指令从出口端口转发。OpenFlow 中交换机支持逻辑端口、物理端口以及保留端口。

2.3 SDN 控制器

2.3.1 SDN 控制器介绍

在 SDN 架构中，控制器可以说是 SDN 的最核心的组成。如果将 SDN 比作一个人，那么控制器就相当于一个人的大脑，对网络的各个部分进行集中的控制和统一的调度、管理，而 OpenFlow 交换机就相当于一个人的双手，在控制器的管理、支配下完成一系列的网络中的行为。控制器是上层应用和底层设备之间的交互桥梁，它不仅可以通过南向接口协议对底层网络交换设备进行状态监测、转发决策等一系列的集中管理，也通过北向接口为上层应用提供可编程的 API 接口。由于 SDN 技术发展潜力巨大，很多开源社区为 SDN 研究者们提供了开源控制器，他们虽然拥有各自的优势，但都封装了 OpenFlow 协议，对上层应用提供编程接口，使开发人员能够更加便利的进行开发工作。接下来，我们将简单介绍几款主流的控制器。

（1） NOX/POX

虽然 NOX 的社区随着其他控制器的崛起而不再活跃，但是 NOX 作为 SDN 的第一款控制器对于 SDN 技术来说具有里程碑式的意义。由于其开发语言的问题使得它在日后的竞争中渐渐处于劣势。由 Murphy McCauley 运营的开源社区所提供的 POX 控制器是 NOX 的兄弟版本，在 SDN 发展的初期扮演着相当重要的作用。虽然内部机制与 NOX 相同，但是由于他开发语言简单，容易入门而得到了广泛的关注和应用。然而随着 SDN 的发展，更多的控制器纷纷涌现，他们更成熟的框架和更优秀的性能使得 POX 控制器逐渐在竞争中处于下风。

（2） Ryu

Ryu 是由日本电报电话公司——NTT 公司开发的模块化控制器，其设计目标是为 SDN 提供一个具有控制能力的网络操作系统，开发者可以通过调用这个网络操作系统提供的 API 来进行二次开发。Ryu 是基于 Python 语言进行编写，遵循 Apache 许可，并能支持 OpenFlow 各个版本。由于其架构清晰、性能良好、文档齐全及有社区的 Plug-in 集成到 OpenStack 的众多优点而被 SDN 开发者所青睐。

（3） OpenDayLight

OpenDayLight（ODL）控制器是由 Linux Foundation 及 Cisco、Juniper 及 Broadcom 等多家网络巨头公司在 2013 年共同创立的开源项目，其目的在于搭建一个能够在 SDN 网络中通用的使用平台，让其成为 SDN 中的核心组件，从而能够降低网络管理人员的运营复杂度，扩展网络架构中现有设备的生命周期并能够支持 SDN 的新业务。因此 OpenDayLight 是一个具有极大野心的开源项目，控制器反而在其中的重要程度并不是那么高。OpenDayLight 基于 Java 语言编写，支持多种南向接口协议，采用模块化的设计思想、OSGI（Open Service Gateway Initiative）[20]体系架构，能够实现各种功能之间的低耦合、增强控制平面的可拓展性。

（4） ONOS

ONOS（Open Network Operating System）由非盈利组织 ONLAB 于 2014 年末推出。与 OpenDayLight 由设备商驱动不同，ONOS 的主要面向对象是企业骨干网络和设备提供商，其目的是打造一个开放的 SDN 网络操作系统，提高更高的灵活度及更好的性能，市场定位于运营商级别的网络市场。ONOS 同样采用 Java 语言编写，利用 Karaf[21]为其 OSGI 框架、Maven 管理项目的构建，在最新版本中使用 Raft 作为分布式框架。ONOS 支持设备管理、链路发现等一系列基本网络服务，也提供 API 供上层应用使用。

（5） Floodlight

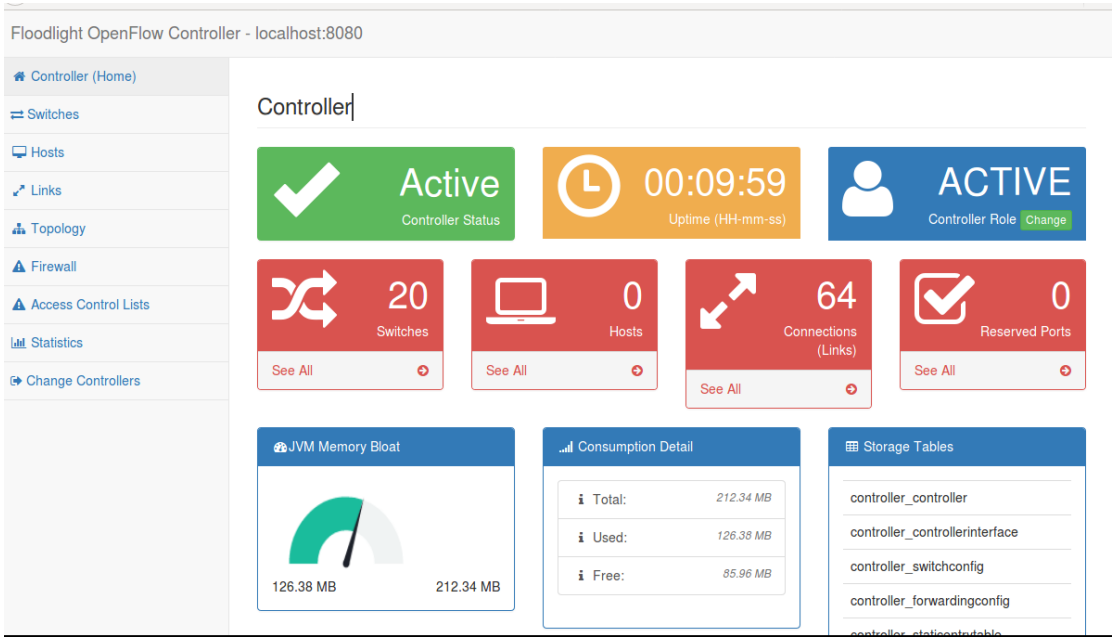

图 2-6 Floodlight 控制器界面

Floodlight 是由 Big SwitchNetworks 公司主导并参与维护的开源的控制器项目，是一个基于 BSN 公司提供的 BNC（Big Switch Controller）架构的、Apache 许可，基于 Java 语言的 OpenFlow 控制器。Floodlight 具有良好的易用性和扩

展性，更低的耦合度、更加稳定的性能及对商业网络应用的友好性受到了很多开发者的欢迎。相较于基于 Python 编写的 Ryu，采用 Java 编写的 Floodlight 更加高效，相比于更多功能的 OpenDayLight，Floodlight 更加轻量易读，能使开发者更加专注于对算法的研究，因此本文将选择 Floodlight 作为控制器进行本文算法的开发。

2.3.2 SDN 控制器特性分析

控制器在 SDN 中占据着核心的地位，控制器的性能将直接影响网络能力，因此控制器的设计也是 SDN 种极其关键的部分，它既要能够融合传统网络的技术，又要能够遵循 SDN 架构的设计理念。基于多款控制器的性能比较，总结出控制器的关键设计特点。

（1） 支持 OpenFlow 协议

OpenFlow 协议是目前最主流的南向接口协议，是控制器与底层网络交互的桥梁，所以控制器必须能够实现对 OpenFlow 协议的支持。不管是控制器与交换机之间通过安全通道的交互，还是控制器下发 LLDP 消息、交换机上传 Packet_In 消息都是通过 OpenFlow 协议完成的，因此支持 OpenFlow 协议是控制器最为基础的功能。

（2） 网络虚拟化

SDN 最大的价值就是网络功能的虚拟化[22]，简而言之，是将网络底层的资源统一分配，使得能够共享网络资源，从而提高资源利用率。为了能够达到价值最大化，网络虚拟化以一种端到端的、联合计算资源和类似于服务器虚拟化抽象的方式完成对网络资源的抽象和整合。利用这些功能来创建面向特定租户来源于基础设施层的虚拟网络，通过北向接口将虚拟网络分离出来。

（3） 网络功能化

在传统网络中，在处理相关的网络问题时需要交换机相对独立的支持网络协议，然而在 SDN 中控制器需要对底层网络的集中控制进行网络功能的实现。例如对于路由流量来说，控制器需要具有在源和目的之间发现多条路径的能力，正是这种能力使得 SDN 能够具有消除生成树协议或 TRILL 等能够增加网络复杂性的协议，增加解决方案的有效性和可拓展性。

（4） 可拓展性

在传统的多层网络架构中，由于端到端的路径中至少存在一台第三层设备，因此网络的拓展性并不理想，而 SDN 允许使用可拓展的网络模型，SDN 控制器的基于模块化设计和开放的程序接口以及允许在横向拓展模型上添加物理网络功能，将网络抽象成单个设备管理，这些特性都有助于网络功能的拓展。

（5） 可靠性

由于控制器在 SDN 中处于核心的地位，因此很多人质疑当控制器失效时，整个网络都将会因此而受到影响。针对这一问题，SDN 控制器采取了很多有效的解决措施。例如控制器会在源和目的节点之间寻找出多条路径，通过控制器对网络的监控，当链路失效时将流重定向到其他的路径上。在控制器自身的可靠性方面，针对控制使用支持冗余设计的硬件和软件从而能够令单点故障不会影响整个系统的运行，除此之外，可以采用控制器集群的方式也会增加控制器的可靠性。然而为了能使集群控制器遇到故障快速切换，处于活跃状态的控制器和处于等待状态的控制器之间的同步显得尤为重要[23]。

（6） 网络安全

由于 SDN 中控制器的集中控制能力使得控制器成为了攻击者的首要攻击对象，因此维护控制器的安全很大程度上就是维护网络的安全。为了保证网络的安全性，控制器也必须提供网络管理员授权和身份认证对 SDN 的访问进行控制。除此之外，SDN 控制器需要对租户之间进行隔离并通过过滤器对数据包进行过滤，根据限制的通信速率在可疑攻击来临时提醒网络管理员。

2.3.3 Floodlight 架构分析

在 SDN 的架构中，控制器处于最核心的位置。控制器扮演者 SDN 的大脑的角色，起着监控、调度、管理等一系列至关重要的作用。基于上文在对不同控制器的分析，我们将选择 Floodlight 作为实现算法的控制器。为了相应模块的开发，我们将分析 Floodlight 的整体架构和功能模块。

Floodlight 是采用的是模块化设计，这一低耦合、高复用的特点使网络的管理者可以根据服务需求而加载相关的模块。Floodlight 中每一个模块都定义为一个 Java 类，各个模块之间可以通过 API 互相调用。其主要实现的功能为：[24]

（1） 通过 OpenFlow 协议实现交换机和控制器之间的交互；

（2） 发现网络状态和网络事件，例如设备、拓扑结构等；

（3） 处理交换机发来的 Packet_In 消息，生成流表，管理交换机中数据的流向；

（4） 提供 web 页面和调试服务器；

（5） 管理 Floodlight 模块，共享存储、线程、测试等资源。

Floodlight 的核心的架构包括由模块管理系统管理的各个模块，例如拓扑管理、设备/终端管理、路径/路由计算、计数器存储等。从功能上，我们将 Floodlight 分为两个大类，即用于实现 SDN 核心功能的控制器模块以及用于实现不同业务的应用模块。开发者可以利用控制器模块提供的 Java API 或 Rest

API 来进行新模块的开发，也可以利用应用模块提供的 Rest API 进行上层应用的开发。

应用模块	实现功能
forwarding	响应Packet_in消息，根据Packet_in消息制定转发路径并下发流表
StaticFlowPusher	支持静态流的插入和删除
VirtualNetworkFilter	支持二层网络虚拟化
Firewall	无状态防火墙，提供基于REST API的配置接口

图 2-7 应用模块及功能

控制器模块是实现控制器核心功能的模块，其主要功能就是实现控制器对网络的监控、管理和调度，包括发现和获取网络中的状态和事件、实现与交换机之间的交互和管控。通过管理控制模块完成模块间的资源共享，提供一个 web 界面，实现网络功能的可视化操作。其主要的功能如图 2-8 所示

控制器模块	实现功能
FloodlightProvider	提供两个主要的功能模块。 1）处理交换机连接并将OpenFlow消息转换为事件使其他模块得以监听。 2）决策特定的OpenFlow消息转发到监听模块的顺序。
DeviceManagerImpl	跟踪设备在网络中的移动并且为新流量定义目的设备
LinkDiscoverManager	通过下发LLDP包来发现交换机并维护交换机的状态
TopologyService	获取网络中的拓扑信息，寻找网络中的路由，完成路径的计算
ThreadPool	用来创建某一特定时间运行或周期时间运行的线程
PacketStreamer	能够将OpenFlow数据包转发到网络中任意一个监测的设备上。
Flowcache	鼓励开发人员根据他们的需求处理网络中的事件
Memory Storage Source	存储共享数据，查询所有模块信息及其变化

图 2-8 控制器模块及功能

2.4 Mininet 仿真平台

2.4.1 Mininet 概述

Mininet 是一个轻量级的 SDN 网络研发、仿真、测试平台。它是斯坦福大学的研究人员基于 Linux 容器开发的、支持 OpenFlow 协议的虚拟化仿真平台

[25]，利用这一进程虚拟化的技术可以迅速在一台主机上完成整个网络的部署。通过 Miniet 仿真平台搭建的 SDN 网络，可以以 Open vSwitch 或 OpenFlow 为基础，验证关于 SDN 网络的新想法，并且通过 Mininet 的代码能够支持无缝隙的迁移到真实的硬件环境中[26].Mininet 主要特点：

（1） 支持 OpenFlow、SNMP 等协议。

（2） 提供可拓展的 Python API 接口，便于用户自定义网络拓扑。

（3） 内置如 Iperf、窗口监测软件等多种网络测试软件。

（4） 提供 CLI 接口，方便对测试参数的配置。

（5） 很好的硬件移植性与高扩展性。

作为一个基于 Python 开发的网络仿真平台，Mininet 主要包括 Python 库和运行文件两大块内容。其中 Python 库主要包含 link 模块、node 模块、cli 模块以及 topo 模块等，用以提供对网络中元素进行抽象和实现这一功能；而运行文件是通过运行文件中的 mn 脚本实现基于 Python 库实现自定义的拓扑。其中 mn 脚本中定义 MininetRunner 类模拟网络运行中的主程序。

Mininet 作为轻量级的网络仿真平台，能够通过 CLI 接口快速实现网络模型的创建，当 Mininet 中对网络模型进行验证后即可无缝的部署到真实的网络环境当中去。在完成网络环境的部署后，可以通过 CLI 命令获取网络相关状态信息来进行相关测试，其常见命令如表 2-1 所示：

表 2-1 CLI 常见命令

命令	说明
nodes	列出拓扑中的所有节点
dump	打印出当前拓扑中所有节点的详细信息
net	展示出拓扑中所有的网络链接的详细信息
link	禁用或启用相邻节点之间的链路
iperf	调用内置 iperf 工具进行两点间的 TCP 测试
Iperfudp	调用内置 iperf 工具执行指定带宽的 UDP 测试
Pingpair	调用内置 ping 工具在两主机之间进行 ping 测试

2.4.2 Iperf 概述

Iperf 是一个网络性能测试工具，其源码是采用面向对象的 C++语言实现的。Iperf 可以测量出 UDP 和 TCP 的带宽质量，通过对 Iperf 的的多参数设置，可以测量 TCP 最大带宽及 UDP 的带宽、延迟抖动和数据包丢失等一系列指标，因此本文将利用 Iperf 来进行网络的性能测试。

表 2-2 iperf 常用参数与说明

命令行选项	描述
-s，--server	Iperf 的服务器模式
-c，--client	Iperf 的客户端模式，需要连接指定的 Iperf 服务
-u,--udp	指定使用 UDP 模式，默认状态下为 TCP
-b --bandwidth	客户端参数值，用于 UDP 模式下使用的带宽，单位为 bits/sec，默认值为 Mbit/sec
-i，--interval	设置每次报告之间的时间间隔，单位为秒，默认值为 0。
-t，--time	客户端参数值，用于设置传输的总时间，在指定时间内重复发送指定长度的数据包。默认值为 10 秒。

Iperf 主要用于测试 TCP 和 UDP 的性能，其参数与说明如表 2-2 所示。当其测试 TCP 带宽时是在在客户端和服务端之间建立三次握手连接后，客户端发送的带宽为发送数据和时间的商值。服务器端所测得的带宽是接收数据和时间的商值。通过 TCP 内核接口函数直接获得 MSS 的大小。TCP 的主要功能如下：

（1） 测量网络带宽

（2） 报告 MSS/MTU 值的大小和观测值

（3） 支持 TCP 窗口值通过套接字缓冲

（4） 当 P 线程或 Win32 线程可用时，支持多线程。客户端与服务器支持同时多重连接

UDP 主要功能如下：

（1） 客户端可以创建指定带宽的 UDP 流

（2） 测量丢包率

（3） 测量延迟抖动

（4） 支持多播

（5） 当 P 线程可用时支持多线程。客户端与服务器支持同时多重连接（不支持 Windows）。

客户端可以设定 UDP 数据流发送的速率和数据包大小，根据数据包内带有的 ID 号对报文进行唯一标示，服务器端可以依据 ID 号的顺序来确定数据包是否丢失或乱序；网络层包所丢失的数据可以通过将设置 UDP 报文大小恰好可以置于网络层包内时，测量此时 UDP 的丢失报文；除此之外，服务器端也可测得数据包传输延迟抖动，传输延迟抖动是一个相对值，它所反映的是传输过程中时延的抖动情况。

2.5 本章小结

本章首先介绍了 SDN 的定义和架构，在 SDN 架构的基础上引入了 OpenFlow 技术的概念及 SDN 控制器。从 OpenFlow 协议的历史与发展、OpenFlow 网络架构、OpenFlow 交换机等方面详细的描述和分析了 OpenFlow 技术，介绍 SDN 控制器，并对主流控制器进行介绍与分析，选定本文算法中使用的控制器，并对控制器的特点进行了详细的分析。最后介绍了 Mininet 仿真平台以及内置于 Mininet 平台中的网络性能测试工具——Iperf。本章通过介绍相关的技术与理论知识知识，为后续的研究提供了理论支持。

第三章 基于 SDN 负载均衡技术研究

本章通过介绍基于SDN负载均衡算法的实验场景和其他相关的基于SDN的负载均衡的算法以及基于时延的 Dijkstra 算法，提出本文的负载均衡算法——LCLB 算法。针对于 Dijkstra 算法路径范围庞大，容易引起局部拥塞等问题，LCLB 算法通过对路径剪枝从而缩小路径范围，对路径中链路指标的综合评估，计算每条路径的花费值，通过确定的源、目主机获得主机间的路径，根据路径花费值选择花费值最小的路径作为转发路径，从而避免拥塞，达到负载均衡的目的。

3.1 数据中心网络

随着云计算的兴起，数据中心的使用也越来越普及。数据中心承载着海量数据的传输和存贮，关键数据的访问和海量数据的同步都需要高质量的网络服务来支持，因此数据中心内部对负载均衡有着极高的要求。胖树拓扑是数据中心中非常常见的一种网络拓扑，本文的负载均衡策略的实验正是基于此场景而实现。

3.1.1 数据中心网络概述

数据中心网络（Data Center Network）是应用于数据中心内的网络,用来在网络基础设施上传递、计算、存储数据信息。由于数据中心内的流量具有典型的交换数据集中、东西流量增多等特点，因此数据中心网络必须能够满足大规模、高扩展性、高健壮性、低配置开销、服务器间的高带宽、高效的网络协议、灵活的拓扑和链路容量控制、绿色节能、服务间流量隔离和低成本[27]等要求。传统的 TCP/IP 网络架构越来越难以满足这种需求，网络扁平化、网络虚拟化及网络可编程化的网络成为数据中心网络架构的新趋势。

3.1.2 胖树拓扑

数据中心采用的网络拓扑通常多种多样，例如网状拓扑、树形拓扑、多级分支-主干拓扑、水母型拓扑及超立方体拓扑等。通常情况下水母型拓扑和超立方体拓扑都是用于某些特殊网络，而数据中心中常用的拓扑类型主要为树形结构及其改进后的胖树结构。如图 3-1 所示的为典型的胖树结构拓扑，本文所做的基于 SDN 负载均衡的研究与实现均是基于胖树网络拓扑。

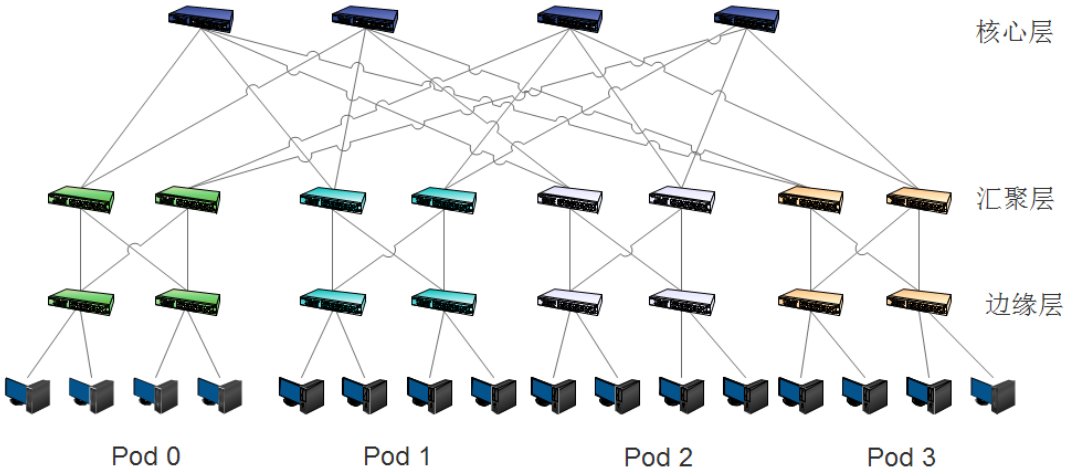

图 3-1 胖树结构

一个 K 叉三层胖树拓扑由 K 个阵列（Pod）构成，每个交换机中都包含着 K 个端口。在每个阵列中均包含了$K/2$个边缘层交换机和$K/2$个汇聚层交换机。边缘交换机中$K/2$个端口与$K/2$个主机相连，剩余$K/2$个端口与同一阵列中的$K/2$个汇聚层交换机相连；汇聚层交换机除了与边缘层交换机相连外，剩余的$K/2$个端口与核心层交换机相连。每一个核心层交换机中的第 i 个端口与第 i 个阵列相连。这一规定保证了所有的主机到核心交换机的跳数均为固定的。通常来说，由 K 个端口交换机所构成的胖树拓扑由$K^3/4$台主机和$5K^2/4$台交换机所构成。这种设计的拓扑网络因为交换机的功能完全一致，能够支持大规模的拓展，大大降低了部署费用。

胖树网络中，由于主机与边缘交换机直接相连，因此可以将与主机相连的边缘交换机作为路径的起点和终点。由于胖树网络的特点，我们可以得出：若源主机和目的主机处于同一边缘交换机，那么只需要访问到边缘交换机即可；若源、目主机与相同阵列中不同的边缘层交换机相连，那么最高只需访问到汇聚层交换机；若源主机和目的主机处在不同阵列中的不同边缘交换机，那么最高层才需访问到核心交换机。根据这种特性，我们将设计出一种动态的负载均衡算法，来提升胖树网络中的网络性能，从而实现网络资源利用率的优化。

3.2 负载均衡相关技术

负载均衡技术就是通过对网络中的节点的负载均衡化，调整各节点的负载情况，使得每一条链路上承担的流量更为平均，从而避免某些节点因为负载过重而引起蒸个网络性能的降低，从而降低系统的丢包率，提高系统的吞吐量，提高网络的稳定性与可靠性。

3.2.1 相关 SDN 负载均衡算法

ATM 网络时期，主要通过使用流量工程技术来解决拥塞控制问题；随着 IP 网络快速发展，负载均衡主要通过路由优化得以实现；而面对路由优化的缺点，以标签作为决策依据的多协议标签交换技术（MPLS）渐渐得到业内认可；然而，MPLS 控制层管理和实现过于复杂这一难题恰巧被 SDN 解决。

相较于传统网络中负载均衡技术，SDN 由控制器实现集中管理的功能，控制器通过与交换机的交互来获取到全局信息，统计网络节点和链路的实时信息、执行控制功能、完成全局部署。基于 SDN 负载均衡技术比硬件负载均衡技术成本更低、灵活性更强，可以通过全局信息的掌控更改策略；与基于软件负载均衡相比，可以获取全局状态，实时对策略进行改变，具有更高的性能。

Plug-n-Serve 方法是控制器执行一个综合的优化算法 LOBUS，自助扩大了网络的视图，共享新增设备的负载，最小化了平均响应时间，从而实现负载均衡。该方案主要面向校园网，在其他场景中适应情况并不理想。

Wang R[28]提出的方法更改了传输机制，修改了 OpenFlow 协议，以发请求的客户机 IP 地址前缀作为通配符，在二叉树上分配通配符到对应的服务器形成规则，若干个规则构成一个规则集，通过建立规则集实现流分布，降低控制器介入次数、请求延时，从而实现了负载均衡。这个算法的缺点在于只检验了简单网络，无法适应复杂的网络拓扑。

Koerner M[29]提出的是支持多服务的 OpenFlow 负载均衡，即控制可以处理多种不同服务，控制器可以根据不同的服务选择不同的负载均衡策略。这一方案能够对不同服务做差异化处理，但在数据中心中会因为突发流量导致网络拥塞。

Li Y[30]提出了数据中心网络中基于的胖树拓扑的动态负载均衡算法（DLB，Dynamic Load Balancing）。在该算法中以流为最小单位，利用深度优先算法，使用单跳贪心算法选取当前节点下可用带宽最大的链路作为下一跳直至最高层。该算法与传统算法进行比对，得到 DLB 算法具有更小的传输时延、更高的带宽利用率。然而由于 DLB 使用的贪心算法会局限于局部最优导致可能无法达到全局最优的选择。

吴宇文[31]在 DLB 的基础上提出了全局负载均衡算法（GLB，Global Load Balancing）。GLB 为网络中的每一对主机维护着他们之间可能存在的路径集合，针对每一个路径中的完整路径计算出路径权重，最后根据路径权重进行选路。这一算法改进了 DLB 算法陷入局部最优的问题但因为需要维护所有路径，当网络变大时有可能出现性能瓶颈。

陈云[32]提出的也是基于 DLB 算法改进的 DSLB 算法。DSLB 算法分别维护着上行、下发路径及链路负载表，按照最大最小化法则选择路径可用带宽，根据变异系数选择出最后路径。

3.2.2 基于时延的 Dijkstra 算法

Dijkstra 算法是网络路由算法中最为经典的算法之一，基于时延的 Dijkstra 算法是将链路的时延作为节点间的权值，获取到时延最短的路径作为两点间的最短路径。在基于时延的 Dijkstra 算法中，首先将所有相连的链路置于同一个簇中，从而使整个拓扑形成一个或多个簇结构，将每一个簇内的交换机形成一个广播树结构从而确保 Dijkstra 能够计算簇内任意节点之间的最短路径。随着链路流量的不断变化，链路的时延也会随之发生改变，那么最小路径也会随之而变，这样就能够实现网络流量的负载均衡。

Dijkstra 算法基于当前网络中链路的时延对流量进行调度，首先控制器通过下发 LLDP 报文来获取网络链路信息，然后通过链路信息生成网络拓扑图。当新数据流到来时，控制器将从 OpenFlow 交换机接收到的 Packet_In 报文交给拓扑模块进行处理。根据报文的源、目地址确定相应的源、目主机，通过 Dijkstra 算法在等价路径中确定时延最小的路径。

基于时延的 Dijkstra 算法虽然简单有效，但是塔尔的缺陷也较为明显。它在计算路径时会将计算出所有交换机之间的路径并将这些路径进行存储，并且由于 Dijkstra 是基于时延寻路，因此它会牺牲跳数使路径时延最小。除此之外，由于算法计算路径中所有链路的总时延，并按照总时延的大小进行排序，因此算法对某条链路是否拥塞并不会特别敏感，当路径中某条链路负载较大而其他链路负载较小时，路径的总时延仍可能较小而被选择，这一结果会导致数据流汇聚，使原本负载较大的链路发生拥塞，造成丢包，从而影响了负载均衡的效果。针对 Dijkstra 算法的不足之处，本文提出了基于最小花费的负载均衡算法。

3.3 基于 SDN 负载均衡算法设计

3.3.1 链路多指标综合评估

在传统的网络中，实现负载均衡往往通过基于时延或跳数的路由策略，这种情况下的选路不能考虑到网络中链路的传输状态，因此并不能达到很好的效果。对链路的性能评估在对网络测量的基础上，通过对其性能数据进行量化分析反应出链路的性能和相关的负载状况。为了能够表示链路的综合情况，我们可以将多个表示链路性能的指标按照一定的逻辑关系进行整合，转换成一个可以权衡这个链路负载情况的综合指标，按照这个综合指标的评估方式对整个链路进行评估。

在 SDN 中，可以通过控制器的集中管理得到整个网络拓扑中的每一条链路的实时信息，因此我们可以借助这一特点，为实现基于 SDN 的动态负载均衡策略计算出一个链路的综合评价权重值，根据这个综合指标引导流量时刻处于最优路径，从而实现我们的研究目标。

全球互联网技术权威组织 IETF 的 IPPM[33]提出了一系列的网络 QoS 的性能指标，例如时延、网络带宽、路径利用率、丢包率等。在 SDN 中，OpenFlow 交换机是网络测量技术的重要单元，通过两个交换机之间的链路工作情况能够评测出网络的整体或部分性能状况[34]。因此我们把两个网络设备之间的链路作为网络中的最小测量对象。根据网络测量获取到每条链路中各个性能指标的数据，建立一个能最大化反应链路状态的评估模型，对链路状态进行评估，最后得出链路的综合性能情况。

根据文献[35]我们可以看出，评价一条链路性能的指标有很多，然而每一个指标对链路整体的状况的影响是不同的，因此需要数学变换的方式，对每个指标进行规格化的处理，从而消除指标数值之间的差异。性能评估指标的大小反应出链路的负载能力的强弱，这一指标越高表明链路的负载能力越强，反之则表明链路的负载能力越弱。在本文中，以负指标（时延、丢包率等）为测量值，负指标值越小，我们认为链路的花费越小，则网络的性能越好。

$$V_j = \sum_{i=1}^{n} v_i \sum_{j=1}^{n} \quad (3.1)$$

如公式（3.1）所示，V_j是链路 j 的综合评价值，x_{ij}是指 T 时刻第 i 个指标在链路 j 中的测量值，v_i是指第 i 项指标在综合评价值中的比重，0<v_i<1。我们会根据负载均衡中的指标比重的不同来调整其权重系数。

3.3.2 LCLB 算法设计

本文在基于 SDN 的数据中心网络胖树拓扑中提出了一种综合考虑路径中链路指标的最小花销负载均衡算法（LCLB,Least Cost Load Balancing ）。不同于 Dijkstra 算法中需要找到所有交换机之间的路径，LCLB 算法中只计算并存储源、目交换机为边缘交换机的路径，这一操作将大大减少路径占用空间，缩小路径的搜索时间；当确定数据流的源、主机后，根据源、目主机找到相对应的边缘交换机并取到相应的等价路径，比较每条路径的花费，选取最小花费的路径作为转发路径对数据流进行转发。

在胖树网络拓扑中，当流量从源主机传输至目的主机时，只要确定出需要访问的最高节点，则其下行路径唯一。因此选择一条上行路线至最高节点后，其下行的路径也就随之确定。根据这一特点，我们只需为网络中的每一对主机H_{src}和H_{dst}找到其上行的最高层便可确定整条路径，然后存储他们之间所有

可能的路径集合{PathId：{Paths}}，由H_{src}和H_{dst}能确定出唯一的一个键值，因此主机间的路径表示为：

$\{\{H_{src},H_{dst}\}:Path_1,Path_2,\ldots Path_n\}$

对于每一条路径$Path_u$，维护构成它的每一条链路集合{links}：

$\{\{link_1,link_2,\ldots link_m\}$

随着网络的规模不断增大，路径集合和链路集合也随之变大，但是在发现拓扑阶段确定网络拓扑后，路径和链路也随之确定。

根据上文中的介绍，我们将对本算法中涉及到的指标做如下定义：

定义 1.链路占用带宽 Bu 和链路带宽利用率δ_{ij}。对于源、目主机之间所有路径所维护的所有链路，我们可以直接根据控制器统计到其带宽的占用情况，调用控制器统计模块可以获得链路的占用带宽，路径 i 中的第 j 条链路占用带宽、带宽利用率的计算公式如下：

$$B_{u_{ij}} = srcTrans/T \tag{3.2}$$

$$\delta_{ij} = B_{u_{ij}}/B \tag{3.3}$$

其中，srcTrans——周期 T 内链路源端口已发送的字节数，

T——监测周期，本文中我们设定 T 为 15s，

B——链路的最大带宽，本文中设置为 10Mbps。

根据公式（3.3）中路径中的链路带宽利用率δ_{ij}的计算结果，根据最大化原则，我们可以认为路径 i 的带宽利用率 δ_i为：

$$\delta_i = \max\{\delta_{i1}, \delta_{i2}, \ldots, \delta_{in}\} \tag{3.4}$$

这里根据的是链路的带宽利用率的大小能够反映出这条路径的网络负载情况。将路径中所有链路中的最大的链路带宽利用率作为这条路径的带宽利用率。路径的带宽利用率越大则表明这条路径上的负载越大，选择这条路径转发所引起拥塞的可能性越大，越要避免这一选择。基于此，我们将以路径带宽利用率作为判断依据，为备选路径设置一个拥塞阈值，当路径带宽占用率达到拥塞阈值时，我们认为路径 i 达到拥塞状态。

定义 2.路径花费值$cost_i$。根据前文的链路多指标综合评估可知，在算法中路径的性能评估指标可以负指标作为评价值，包括时延、占用带宽等。

$$cost_i = \begin{cases} \delta_i * N & \delta_i \geq \text{拥塞阈值} \\ \beta * \delta_i + \gamma \frac{L_l}{L} & \delta_i < \text{拥塞阈值} \end{cases} \tag{3.5}$$

其中：$L_l = \sum_{j=1}^{n} L_{ij}$ （3.6）

N——表示一个较大的整数，本算法中设置 N 为 100，

L_i——表示路径 i 中的总的时延，

L——表示超时时间，本算法中设置为 1000ms，

β，γ——表示影响因子，并且满足$\beta+\gamma=1$，即带宽占用率、时延在路径花费中所占的比例。由于带宽占用率对载均衡的影响最大，因此 β 设为 0. 7，γ 设为 0.3。

我们根据路径中链路的指标综合评估得出路径的花费值$cost_i$。$cost_i$越小，表明该条路径的花销越小，其越不会引起拥塞从而导致过载。$cost_i$的值会随着链路统计信息的实时更新而得到更新，这就保证了$cost_i$能够实时的反应出每条路径的花销，从而能够让我们在多个等价路径中选择性能最好、花销最小的一条。

3.3.3 LCLB 算法描述

本文提出了在胖树网络中基于路径最小花费的负载均衡算法，以路径中的链路为最小单元，综合考虑链路的多项评估指标从而计算出路径的最小花销，动态地为网络中的流量选择路径，从而避免了网络拥塞的发生，减少丢包率，使网络中的负载更为均衡。算法的伪代码如图 3-3 所示。

algorithm 1 Least Cost Load Balancing
Input: *srcHost*, *dstHost*
output: *bestPath*

```
1: procedure MinCostPath()
2:     srcEdgeSwitch ← srcHost
3:     dstEdgeSwitch ← dstHost
4:     topLayer := GetTopLayer(srcEdgeSwitch, dstEdgeSwitch)
5:     if is edgeLayer then
6:         bestPath := srcEdgeSwitch
7:         return bestPath
8:     else
9:         getupPath := GetUpPath(srcEdgeSwitch, topLayer)
10:        paths := GetAllPaths(getupPath, topLayer, dstEdgeSwitch)
11:        for each path in paths do
12:            BandWithRatioOfPath := GetBandWithRatioOfPath(path)
13:            LatencyOfPath := GetLatencyOfPath(path)
14:            if BandWithRatioOfPath ≤ threshold then then
15:                curCost := GetCosts(BandWithRatioOfPath, LatencyOfPath)
16:            else
17:                curCost := GetCosts(BandWithRatioOfPath)
18:            end if
19:            costs.Add(curCost)
20:        end for
21:        bestPath := MinCosts(costs)
22:    end if
23:    return bestPath
24: end procedure
```

图 3-3 LCLB 伪代码

在 LCLB 算法中，首先需要根据源、目主机找出其直接相连的边缘交换机，在算法中，我们将直接与主机相连的边缘交换机作为路径的起点和终点。根据

胖树网络的特点，源、目主机的公共父节点对应的交换机所在的层数就是流量上行的最高层，当流量从源主机传输到最高层时，其下行路径也就随之确定，因此算法中根据源交换机和目的交换机确定出最高层交换机是计算路径的第一步。确定出流量需要到达的最高层交换机后，LCLB 算法就可以根据流的源交换机和所要到达的最高层确定出最小跳数的 n 条等价路径，并对这些路径以及组成路径的链路进行维护。计算出路径中每一条链路带宽占用率，根据最大化原则，将路径中的最大链路带宽利用率作为整条路径带宽利用率。设置路径拥塞阈值，当路径带宽利用率小于阈值时，将路径中的负指标（时延、路径带宽占用率）归一化，综合评估后得出一个开销值$cost_i$；当路径带宽利用率大于阈值时，将$cost_i$置为一个远大于 1 的值。在策略下发时基于反应式地流表下发，也就是说，当有新的流到来时，交换机将流的信息封装到 Packet_In 消息发送到控制器，控制器对 Packet_In 消息进行解析，调用本文提出的负载均衡算法 LCLB 计算出源、目主机之间路径的花费值，选出最小花销路径后将该路径通过流表下发模块下发至交换机的流表中。当流量经过交换机时将按照流表中的规则进行转发，从而完成了负载均衡。

3.3.4 负载均衡评估指标

为了验证本文提出的 LCLB 算法的正确性和有效性，我们将通过和 Floodlight 内置的 Dijkstra 算法的性能进行比对。在 LCLB 的验证实验中，我们将主机在每秒钟发送的数据量定义为流量负载，通过改变注入网络的流量分析流量负载对全网性能的影响，为此，本文将设定四个性能评估指标。

（1） 平均丢包率 μ

平均丢包率是指数据流从 Iperf 客户端发送到服务器端的丢包所占总包数的比例。由于网络的性能会影响到传输的质量，因此通过对丢包率的评估可以反映网络的实时性能状态。若客户端发送的包数为 ps，接收到的包数为 pr，则其计算公式如下所示：

$$\mu = \frac{\sum_{i=1}^{n} \frac{ps_i}{pr_i}}{n} \tag{3.7}$$

通过对全部流的丢包率的统计得到网络的平均丢包率，并将此作为网络性能的评价指标。丢包率也就会越小，表示网络中发生拥塞的可能性越低，网络的传输质量越好，负载被均衡的效果也就越好。反之丢包率上升，说明网络发生拥塞的情况越严重，负载也越不均衡。

（2） 链路负载抖动 σ

链路负载抖动是指拓扑中所包含的所有链路负载{$load_{link1}$，$load_{link2}$，…，$load_{linkn}$}在周期 T 内的标准差，其计算公式如下所示：

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}\left(load_{linki} - \overline{load_{links}}\right)^2}{n}} \tag{3.8}$$

通过对所有链路负载的统计得到链路负载的标准差作为网络性能的评价指标,为了令获得到的链路负载抖动率不至于过大或过小，本文中将取链路负载中的带宽的单位为 kbps。针对网络中的所有链路进行统计，若链路的负载抖动越大，则表明链路中的流量偏离均值越大，链路的负载分布越不均匀，负载均衡的效果越差。

(3)平均路径时延 ζ

路径时延是网络拓扑主机对之间路径时延的平均值，其计算公式如下所示：

$$\zeta = \frac{\sum_{i=1}^{n} l_i}{n} \tag{3.9}$$

l 指的是主机对之间路径的时延，统计出平均路径时延，当平均路径时延越小，则说明网络中拥塞的可能性越小。

3.4 本章小结

本章主要针对基于 SDN 负载均衡技术算法的设计与实现两个方面进行分析和阐述。

首先介绍了数据中心网络的定义、特点及应用场景，在此基础上介绍了本文所应用的网络拓扑——胖树网络；

然后介绍了基于时延的 Dijkstra 算法，指出其算法的不足之处。在此基础上，研究了在负载均衡策略的制定中如何运用多个指标进行综合性的评估，提供了 LCLB 算法的评价指标设定的理论依据，并提出本文的基于 SDN 的负载均衡算法——LCLB，根据胖树网络的特性，对算法所依赖的评价指标进行设计并详细介绍了算法的思想。

最后，介绍了平均丢包率和链路负载抖动率这两个负载均衡的评估指标，并将在第五章按照这两个指标对 LCLB 算法的性能进行评估。

第四章 基于 SDN 负载均衡技术的实现

本文基于 Floodlight 控制器实现 LCLB 算法，通过分析控制器内部模块与 OpenFlow 交换机交互过程，对算法中涉及到的链路发现模块、拓扑管理模块、信息统计模块、负载均衡模块及流表下发模块进行详细的分析，从而阐述如何通过对控制器进行模块开发实现算法策略。

4.1 LCLB 系统结构

在 SDN 中，控制器扮演着集中控制和策略管理的角色，因此基于 SDN 的 LCLB 算法的实现是在控制器中完成的。根据上文的分析，本文将选取 Floodlight 控制器作为 LCLB 算法开发的控制器。Floodlight 控制器通过与交换机之间的交互感知到网络的拓扑结构，按照一定的周期不断获取拓扑的更新情况，并根据拓扑的结构管理网络中所有主机之间存在的所有路径。当流量到来时，交换机将流量信息封装后发送到控制器，控制器收到后转交给负载均衡模块处理，当负载均衡模块完成计算后再由流表下发模块转回给交换机中。Floodlight 控制器中的这种模块化处理能够令各个模块之间既相对独立又能协同合作，具有耦合性低、结构清晰的优点，使修改 Floodlight 控制器内部模块实现负载均衡变得容易操作。LCLB 系统既然构图如图 4-1 所示：

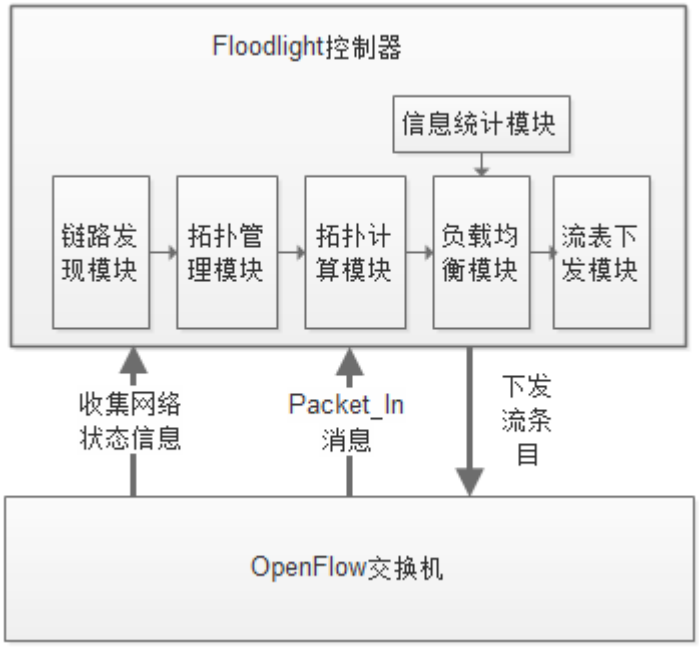

图 4-1 负载均衡系统结构图

当启动 Floodlight 控制器后，控制器下发数据包到 OpenFlow 交换机，根据交换机回传的信息获取到网络中的交换机的位置信息及相关的链路信息。随后将按某一周期不断进行更新操作以获取拓扑内交换机的最新状态，并进行实时更新。随后运行拓扑管理模块，对拓扑中的链路进行更新并完成拓扑的构建，调用拓扑计算模块进行实例化操作。当获取到整个网络的拓扑结构后将交换机按照其位置信息分为边缘层、汇聚层及核心层，结合拓扑中的链路信息，使交换机感知到其相连交换机的位置信息。通过对交换机传回信息的处理，将交换机形成的拓扑结构分为簇、岛、广播树等进行标记和存储。计算所有边缘交换机之间存在的最短跳数的等价路径，并将这些路径传给负载均衡模块，用以求得最小花费路径。调用统计模块中的方法得到实时消耗带宽信息及链路带宽占用率等链路指标信息，并将其带入到负载均衡模块中。将同一条路径中最高链路带宽占用率作为路径带宽占用率，当路径带宽占用率小于负载均衡阈值时（本文设置值为 0.7），我们将综合考量链路指标值并将其归一化，求得路径的花费值；当路径带宽占用率大于负载均衡阈值时，我们将设置路径花费值为一个远大于 1 的值。当数据流到达时，OpenFlow 交换机将相关信息封装成 Packet_In 消息发送到 Floodlight 控制器，控制器解析这一消息，根据源、目主机选择相应的边缘交换机从而得到相应的备选路径，通过比较备选路径中的路径花费值选择花费值最小的一条路径并将其交付给流表下发模块，通过流表下发模块将路径等信息封装成 Packet_Out 信息下发至交换机。交换机根据这一信息装载流表后，数据流根据流表中的这一规则进行转发。

4.2 模块实现

LCLB 算法是在 Floodlight 控制器中进行修改、实现的，而 Floodlight 控制器的模块化结构使得各个模块之间功能清晰、易于开发等优势。针对于负载均衡策略，主要涉及链路发现模块、拓扑管理模块、拓扑计算模块、信息统计模块、负载均衡模块以及流表下发模块。本节将对这几个模块的开发工作进行详细的介绍与分析。

4.2.1 链路发现模块

链路发现模块的主要功能是发现网络中的交换机的位置信息、找到交换机之间的链路并维护链路状态。控制器通过链路发现模块向所有的交换机下发 LLDP 包（Link Layer Discovery Protocol，链路发现协议）及 BDDP 包（广播包），交换机接收到包后将所有端口相连的链路信息回传到控制器中。在 LLDP 和 BDDP 包中包含链路的源目端口、源目交换机的 MAC 地址等信息，LLDP 包标识了交换机的信息后向邻居节点广播以获取对方存在的信息，BDDP 包是

一种类似于 LLDP 的数据包，他可以通过非控制器管理的设备，用来发现不是直连的节点组合。也就是说控制器将 LLDP 和 BDDP 包下发至交换机后，存在以下几种情况：

（1）当网络设备接收到 LLDP 包时，表示为支持 OpenFlow 协议的交换机，那么它将 LLDP 包封装为 Packet_In 消息回传到控制器中，控制器的链路发现模块对回传消息进行解析，提取出交换机的 DPID（标示交换机唯一的 id 信息），从而获取到交换机之间的链路。

（2）当网络设备没有接收到 LLDP 包时，表示网络拓扑中存在非控制器管理的中间设备而导致两台交换机之间无法通过 LLDP 包进行感知，该网络设备是非 OpenFlow 交换机或是主机，此时控制器通过下发 BDDP 包利用中间设备获取相邻交换机之间的关系。

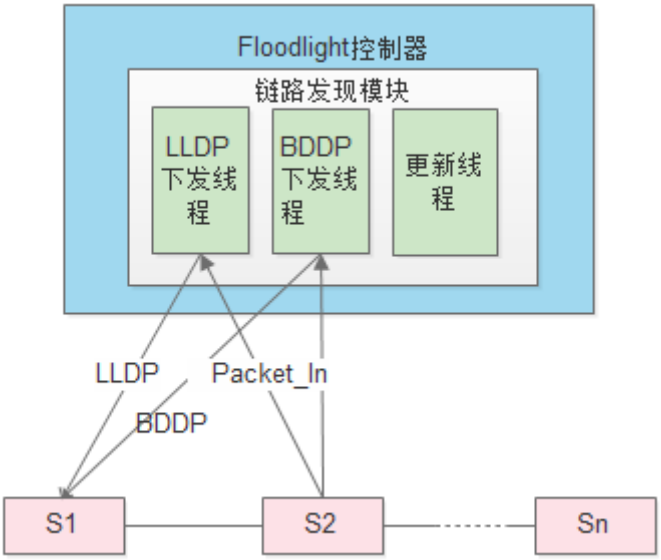

图 4-2 链路发现模块工作图

通过链路发现模块下发 LLDP 包和 BDDP 包并对其进行处理后可以获得整个网络中所有的交换机及链路信息。其具体过程如图 4-2 所示，在链路发现模块中定义了三个线程，通过运行这三个线程对拓扑中交换机进行感知，并发现交换机之间的链路信息。其中：

LLDP 下发线程的功能是发送 LLDP 包，按照不同时间周期向所有交换机的全部端口发送 LLDP 包，根据返回的信息处理过时的链路；

BDDP 下发线程是实现下发 BDDP 包的任务，即遍历所有已发送的 LLDP 包的端口，若端口尚未建立连接，那么便下发 BDDP 包，完成链路的发现；

更新线程主要负责消息的回调，若有端口或链路发生变化，那么便处理这些更新并发布这些更新消息。当 LLDP 包下发到交换机后，交换机处理完毕后将信息封装成 Packet_In 消息上传回控制器，控制器随之解析 LLDP 包。

4.2.2 拓扑管理模块

拓扑管理模块是利用链路发现模块中感知到的交换机和链路的信息维护网络的拓扑结构，为获取网络设备之间的路径提供服务。当链路有更新后，拓扑管理模块会根据链路的最新状态对网络拓扑进行更新处理，每次更新后创建新的实例，重新计算路径信息。若拓扑结构不改变，模块启动后会每隔 500ms 自动进行一次更新操作，如图 4-3 所示。本文中所提出的负载均衡策略将通过拓扑管理模块中创建的实例进行调用，通过负载均衡模块计算路径后将通过后续模块进行下发。

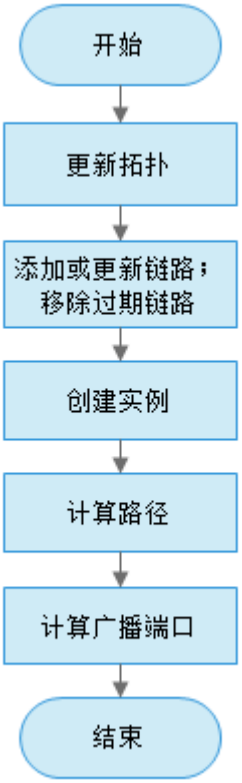

图 4-3 拓扑管理模块流程图

4.2.3 拓扑计算模块

拓扑计算模块主要作用是生成网络中的拓扑信息并计算两个节点之间的路径。本文的 LCLB 算法是采用的是全局最优解，因此要在计算最小花费路径之前求出所有主机之间存在的最短跳数的路径。由于 LCLB 算法的运用场景为数据中心的胖树拓扑，根据胖树拓扑的特点，在寻路的过程中只需找到最高层交换机，下行路径也随之确定。在 Dijkstra 算法中将胖树拓扑按照一般拓扑进行逐步寻路，不能有效的利用他的这一便利性，我们将利用这一特点，将每一个 OpenFlow 交换机封装成为一个 FattreeNode，将胖树拓扑中层数的信息以及上下节点的层数关系填充到 FattreeNode 中。这样每一个 FattreeNode 中除了具

有交换机本来的信息外又含有其在网络拓扑中的位置信息。根据胖树拓扑的特点，当两个主机位于同一交换机下，直接转发；当两个主机位于同一个阵列中，上行的最高层为汇聚层交换机。当主机位于不同阵列时才会上行至核心层交换机，因此通过源、目交换机即可判定出这条路径需要上行的最高层交换机的层数，这样也就得到了所有边缘交换机之间的存在的路径，也就是拓扑中所有转发的备选路径，如图 4-4 所示。由于相同源、目主机之间可能存在多条路径，因此这些路径将会在负载均衡模块得到调用用以计算每一条路径的花费，从而选出最小花费路径。

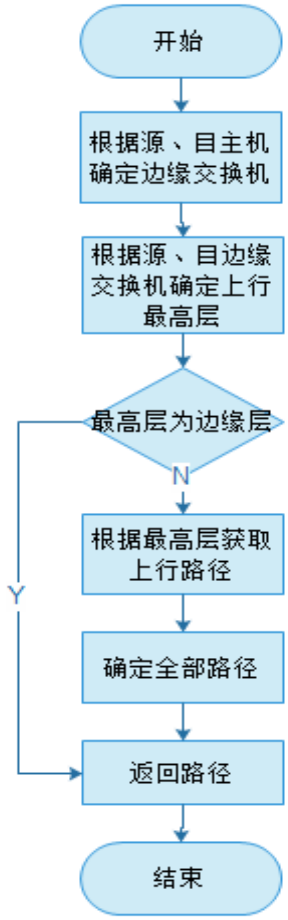

图 4-4 拓扑计算模块线程流程图

在拓扑计算模块中，通过获取链路发现模块中得到的信息可以将网络拓扑划分成一个或多个簇。簇，指的是一组强相连的 OpenFlow 交换机，遍历所有链路，若链路能与簇相连则加入该簇。如图 4-5 所示拓扑，由于存在非 OpenFlow 交换机而被划分成两个簇，一个簇中含有 OpenFlow 交换机 1 和 OpenFlow 交换机 2，另一个簇中仅含有 OpenFlow 交换机 3。两个簇通过中间的普通交换机而连通。遍历簇中的所有交换机生成一个广播树，当源、目主机所连接的边缘交换机在同一个簇内时通过广播树便可计算获得路径。

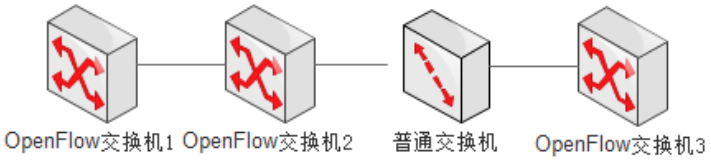

图 4-5 簇拓扑示例

4.2.4 信息统计模块

信息统计模块主要功能是实时获取网络中的链路的状态信息。在本文中需要利用获取到的网络中链路的带宽占用情况及时延完成负载均衡模块中最小花费值的计算，并且在结果分析模块中，需要通过统计模块对带宽、时延、丢包率等评价指标的统计验证算法的优劣。

信息统计模块通过发送状态统计请求信息来周期性获取交换机的状态信息，在 OpenFlow 网络中，交换机与控制器的交互可以是被动式也可以为主动式。通常情况，当流量转发时采取的都是交换机主动式查询，而统计流量一般为控制器主动式查询，即控制器周期性地向交换机发送 STATS_REQUEST 消息，交换机收到该消息后就会立即向控制器回复一个 STATS_REPLY 消息，这个消息中包含了流的字节数和流的持续时间等信息，因此，控制器就获得了特定流的信息。链路的评价指标可以通过对源端口进行监测获得，通过交换机的传回信息可以获取到端口的字节数、丢包数及持续时间等信息。因此可以求得 t1-t2 时间内链路的当前带宽为（s2-s1）/t1-t2，其中 s2-s1 为这两个时间段内传传输的字节数。

4.2.5 负载均衡模块

负载均衡模块功能是实现 LCLB 算法，寻找到最小花费路径，将最小花费路径作为转发路径下发到流表中，当数据流到来时将按此路径进行转发，从而达到负载均衡的目的。负载均衡的流程如图 4-6 所示，遍历拓扑计算模块中所获得的源、目边缘交换机之间的备选路径，通过调用信息统计模块中链路占用带宽求得备选路径中最大的链路带宽占用率，并将其作为此备选路径的路径带宽占用率δ。若路径带宽占用率大于负载均衡阈值（本文中设置负载均衡阈值为 0.7）认为这条路径处于相对拥塞状态，将路径花费值设置为$\delta * 1000$；否则将通过路径中链路的多指标综合评价后求得，并将其归一化，保证这个路径花费值远小于拥塞链路的路径花费值。这里的链路多指标是由路径的平均链路带宽占用率和路径总时延组成，由于带宽占用率对负载均衡的影响更大，因此设置平均链路带宽占用率的影响因子为 0.7，路径时延的影响因子为 0.3。根据这一方法计算出每一条备选路径的路径花费值，相同源、目交换机的路径中，选择路径花费值最小的作为转发路径下发到流表中。随着网络状态的不断改变，

路径的花费值也在不断更新，每次更新都选择最小花费值的路径，从而保证了负载均衡的有效性。

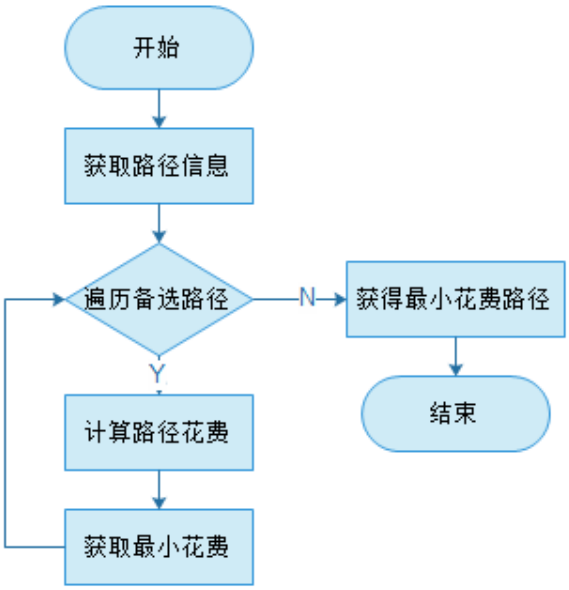

图 4-6 负载均衡模块算法过程

4.2.6 流表下发模块

流表下发模块的主要功能是获取负载均衡模块计算获得最小花费路径作为转发路径并将其下发至交换机的流表中。当控制器获取到从交换机传来的 Packet_In 消息时，根据包的类型或者决策来决定对 Packet_In 消息执行的动作。对于广播包和多播包，执行洪泛操作，对其他类型的数据包根据类型进行转发操作，对于需要丢包的动作，执行丢包操作。

如图 4-7 所示，我们将对转发进行详细的分析，首先通过这个方法获取到 Packet_In 消息中的数据包的匹配域，提取出数据包中的交换机的信息，判断源交换机和目的交换机是否存在，若都存在则继续执行，否则进行洪泛操作；当源交换机和目的交换机都时，将判断两者是否属于同一个二层网络（即同一个簇中），若属于则继续执行，否则进行洪泛操作；当二者属于同一个二层网络时，继续判断源交换机和目的交换机是否是同一个端口，若二者处于同一个端口下就进行洪泛操作，若不在同一端口，那么便调用路径计算模块中的路径转发方法，获取到从源交换机到目的交换机之间的路径。根据其中的匹配规则将路径下发到相应的交换机，完成流表下发模块的操作任务。这里的路径转发方法所转发的源、目交换机之间的路径就是经由负载均衡模块计算得出的同一源、目交换机中多条备选路径里花费值最小的一条。

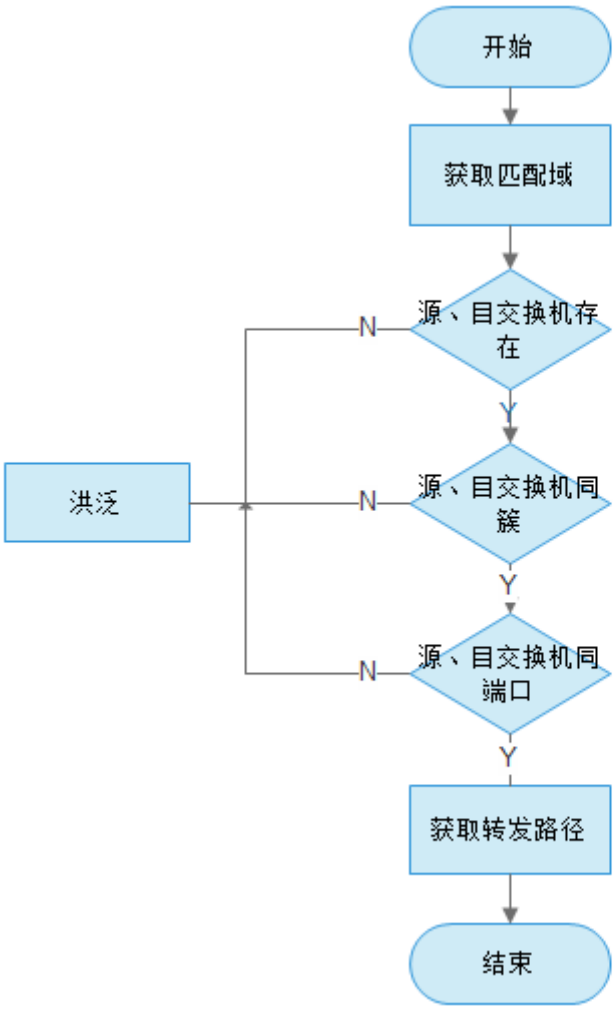

图 4-7 流表转发模块流程图

4.3 本章小结

本章主要是针对 Floodlight 控制器的分析和模块的介绍来完成本文负载均衡算法的实现。

首先，介绍了 LCLB 的算法框架，阐述了算法涉及到的模块以及如何同交换机进行交互。

接着，在 LCLB 算法框架的基础上详细的分析了涉及到的模块，并阐述了如何在相关模块上完成开发工作。

最后，介绍了实现 LCLB 算法的各个模块之间的关系与功能。通过对链路发现模块、拓扑管理模块、拓扑计算模块、信息统计模块、负载均衡模块及流表下发模块进行详细的解读与分析，对每个模块如何工作、模块之间怎样相互调用以及怎样对模块进行开发修改都做了详细的介绍。

第五章 系统测试与分析

本章对 LCLB 算法进行测试，并对实验结果进行分析。首先将介绍本文的实验环境，包括实验的拓扑、实验的系统配置以及生产流量的方式。通过对实验环境进行配置，得出实验结果并对实验结果进行详细的分析。

5.1 实验环境

5.1.1 实验环境设定

本文的实验系统是在内存为 16GB，CPU 为 Intel（R） Core（TM）i7-6700，3.4GHZ、64 位版本的 Windows 操作系统的主机上运行的 4GB 内存的 Ubuntu 16.04 系统的虚拟主机。这台虚拟机同时运行 Floodlight 控制器的软件运行平台，以及 Mininet 仿真实验平台。

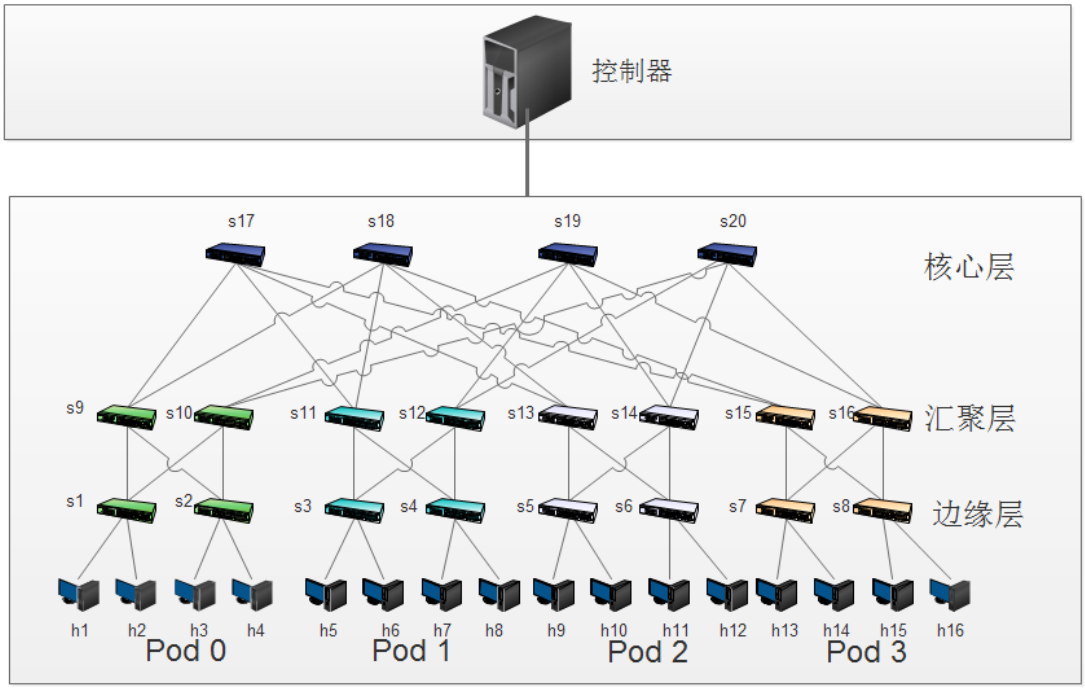

图 5-1 实验网络拓扑图

本文在 Mininet 中的交换机构造胖树拓扑来模拟数据中心网络的场景我们将选取 k=4 来模拟数据中心网络的场景，如图 5-1 所示，网络模型中含有四个阵列，每个阵列包含四个主机，两个连接主机的边缘层交换机，两个连接边缘层交换机的汇聚层交换机，除此之外还包括与汇聚层相连的四个核心层交换机，即胖树网络中共含有 20 台交换机和 16 台主机。由于 Mininet 不支持胖树拓扑，因此通过 Mininet 中的自定义拓扑实现对胖树拓扑的构建。

SDN 控制器通过与核心层交换机相连而实现对整个网络的管控。在 Mininet 的仿真平台中使用的交换机为是内核模式下的 OpenvSwitch，与 Miniet 连接的控制器为 Floodlight，通过 Floodlight 控制器对网络进行控制并实现本文的负载均衡算法策略。

5.1.2 随机流量生成

由于数据中心网络是交换机和大型网络主机的聚合地，网络中流量巨大且错综复杂，因此在仿真实验中，我们希望尽可能真实的模拟数据中心网络的流量。在文献[36][37]中通过编程模拟出数据中心流量模型，本文将按随机性模拟数据中心流量，即网络中任意一个主机向其他的一个或多个主机以等概率发送流量；

本文中使用 mininet 中的 iperf 工具生成网络中的 UDP 流量，Iperf 客户端传送数据流到 Iperf 服务器端，由服务器负责接收并记录相关的信息。虽然 mininet 提供了较为丰富的可使用命令，但却没有给出支持用户自定义命令的 API 接口。因此在诸如数据中心网络这样复杂、网络节点较多的网络环境中进行批处理就需要将自定义的 Iperf 命令添加到 mininet 中，完成拓展。

通过对 Mininet 内部文件修改，对 Iperf 的命令功能进行重新定义，实现随机流量的生成。net.py 文件是仿真平台的主体类。它通过实现了 Miniet 类来完成节点添加配置、实现网络基本功能以及一些选项功能的拓展，因此需要将自定义函数定义到此类中。首先定义一个函数在两个主机之间进行 Iperf 的测试功能，并将结果记录在服务器端。在随机流量模型中，添加一个自定义的命令 iperfmulti，依次为每一个为客户端的主机随机选择一台主机作为它的 Iperf 服务器端通过调用上述函数发送 UDP 流，并将服务器端生成的报告以重定向方式输出到文件中。cli 模块中定义了 CLI 类，提供了命令行接口，用于解析用户所输入的命令，因此自定义命令需要在 CLI 类中通过注册函数来定义这条自定义的命令。在完成了命令的注册和绑定以后，需要在 mininet 执行解释器中注册执行函数与对应命令的映射关系。

通过上述三个文件的更改可以实现自定义 Iperf 命令的功能，在完成其更改后重新对 mininet 的核心文件进行编译和安装后便可以使用自定义命令，本文中我们将上述两种流量模型进行自定义，以此来测试本文中提出的 LCLB 算法和 Floodlight 中的基于时延的 Dijkstra 算法。

5.2 实验结果与分析

经过大量重复的验证实验后，分析实验结果，可以证明在数据中心的拓扑网络中 LCLB 算法的负载均衡效果优于基于时延的 Dijkstra 算法。从实验结果

的数据中可以看出，LCLB 算法相比于基于时延的 Dijkstra 算法具有更小的丢包率及更平稳的抖动率，因此网络的整体性能更加优越。

在上文所述的实验环境中，进行实验后两种算法的路径规模的不同，通过时延试验的 Dijkstra 算法中包含以所有交换机为源、目节点的路径，因此路径的规模要远远大于 LCLB 算法中的路径规模。

链路负载抖动变化实验：使用 iperf 命令语句令 16 台主机以 3M 的带宽随机向另一台主机发送恒定的 UDP 数据流，持续时间为 200s，多次实验后得到的链路负载抖动的实验结果如下图所示：

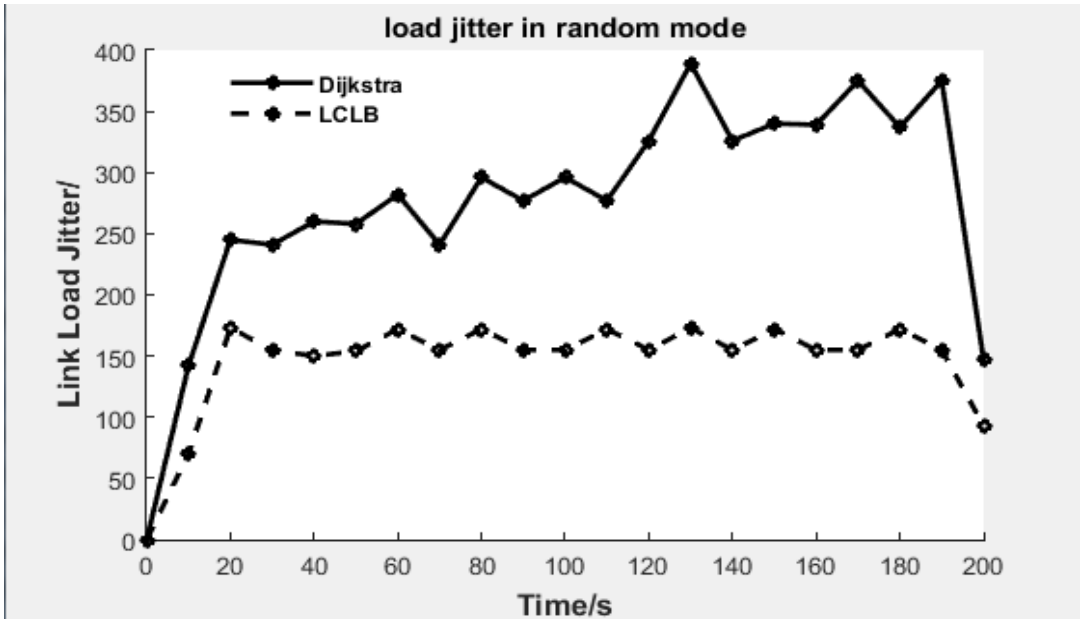

图 5-2 随机模式下链路负载抖动变化图

图 5-2 展示的是在随机流量生成下链路的负载抖动情况变化，整体上看 LCLB 的曲线位于基于时延的 Dijkstra 曲线之下，表示网络中链路负载的标准差较小，负载上更加均衡；LCLB 算法的曲线相较于 Dijkstra 曲线变化更加平缓，表示各条链路的负载值相对稳定，因此在整体性能上要更加优越。

平均丢包率实验：使用随机模式下的 iperf 命令语句，使主机分别以 0.1-0.9 的负载向另一台主机发送恒定的 UDP 数据流，持续时间为 60s，通过分析 Iperf 服务器端生成的报告得到平均丢包率。

图 5-3 展示的是在随机模式下平均丢包率的变化情况。由于实验中有 16 对主机之间会产生流量，传输这些流量的路径中很多链路为共同链路，因此两种算法在负载升高时会产生丢包的现象。随着负载的增加，两种算法的平均丢包率都在逐步升高，但在整体上看，LCLB 算法的平均丢包率要比 Dijkstra 算法更小。在负载较小时，两种算法的表现相当。当负载为 0.3-.0.5 时，LCLB 的平均丢包率好于 Dijkstra 的，其性能的优越性开始得到体现；当负载在 0.5-0.7 时，LCLB 的表现要比远好于 Dijkstra 算法，这是因为随着负载的增加，Dijkstra

算法的最小时延策略不会因为某一条链路时延的过大而拒绝该条路径，从而导致网络的局部拥塞，增大了数据的平均丢包率。当负载增大到 0.8 以后，LCLB 算法的丢包率虽小于 Dijkstra 算法，但由于此时网络中负载过大，绝大部分链路都处于过载状态，因此两算法的丢包率都较大。

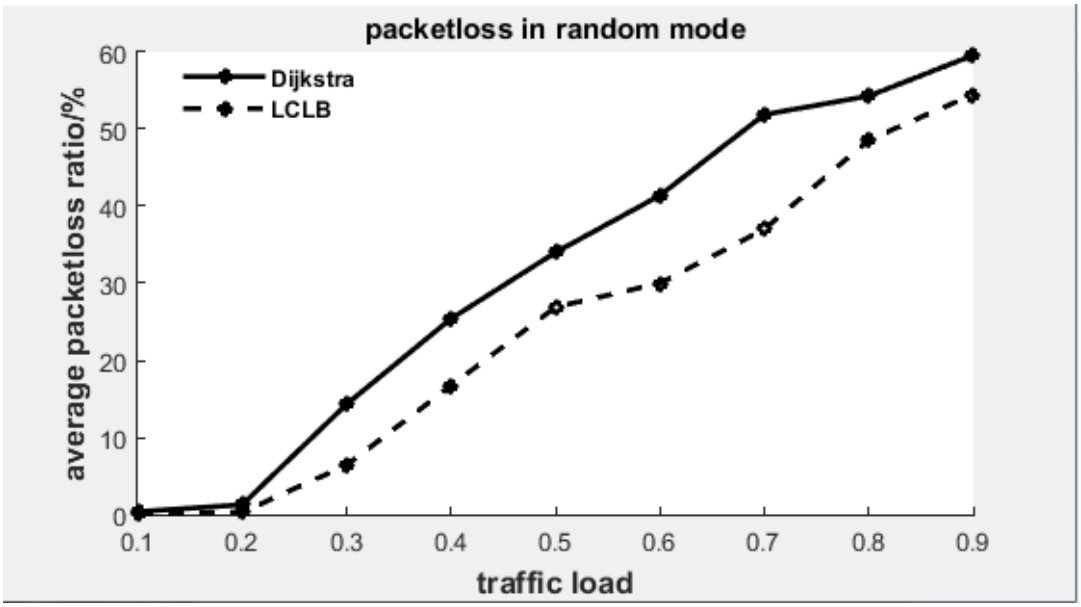

图 5-3 随机模式下平均丢包率变化图

平均路径时延实验：使用 iperf 命令语句令 16 台主机以 3M 的带宽随机向另一台主机发送恒定的 UDP 数据流，持续时间为 200s，多次实验后得到的路径平均时延的实验结果如下图所示：

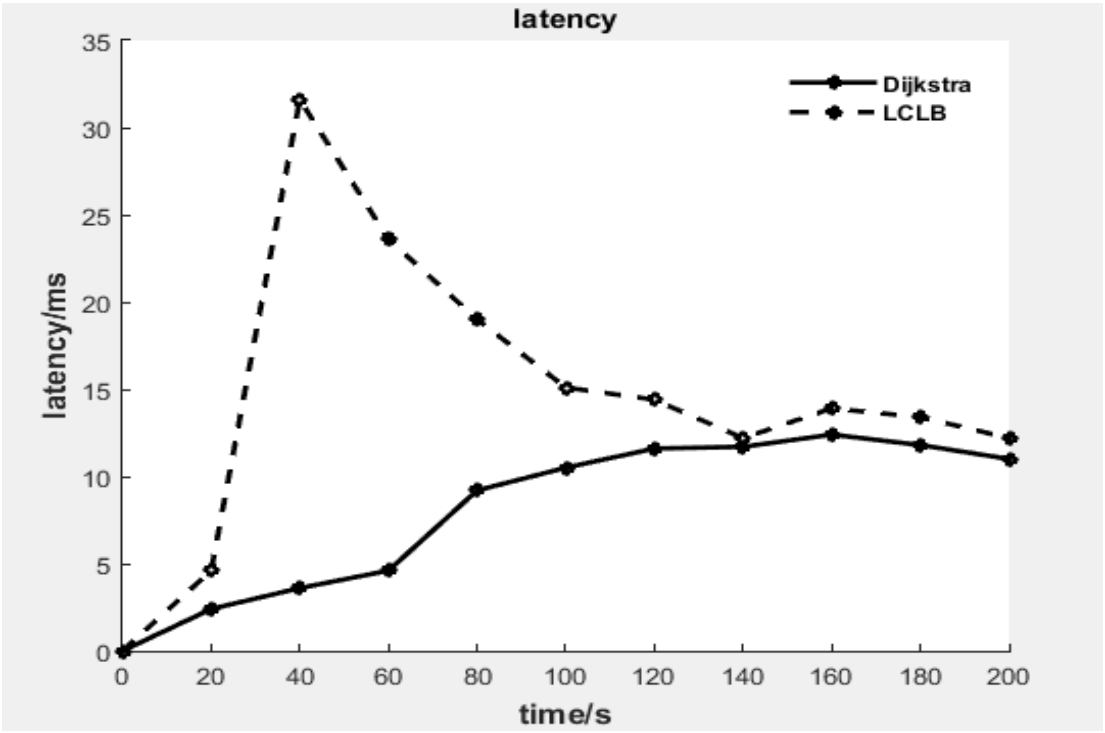

图 5-4 随机模式下平均路径时延变化图

图 5-4 展示的是在随机模式下平均路径时延的变化情况。由于 Dijkstra 算法是基于时延进行选路，因此每次选择的路径都应该是当前时延最短的路径，随着时间的变化，链路的占用率变高，因此时延也在增加。但由于其算法基于最短时延，因此总体上来看 Djikstra 的实验结果要好于 LCLB 算法。LCLB 算法是基于链路的多指标综合计算的算法，时延只是其考虑的一部分，在该算法中链路的带宽占用率所占的比重更大，因此在初始阶段其选择的路径的平均时延要远大于 Dijkstra 算法。随着时间的变化，链路的负载趋于均衡，此时所选路径的时延也随之减小。

5.3 本章小结

本章介绍了仿真实验的环境和实验中随机流量的生成。通过 Floodlight 控制器与 Mininet 相连完成多次重复实验，将 LCLB 和基于时延的 Dijkstra 算法的结果进行对比，通过实验结果可以看出相较于 Dijkstra 算法在链路抖动率和丢包率上面的表现更好，然而由于 Dijkstra 遵循最小时延，因此在时延的实验结果中，Dijkstra 表现的更为优异。由于在负载均衡策略中，丢包率和链路负载抖动更为关键，因此我们可以认为 LCLB 算法比基于时延的 Dijkstra 在数据中心网络拓扑结构中具有更好的负载均衡效果，从而验证了本文工作的有效性。

第六章 总结和展望

6.1 工作总结

本文详细分析了 SDN 的相关背景技术包括 OpenFlow 技术、控制器架构及模块等，通过对 SDN 的当前进展状况及相关技术的研究，在传统负载均衡的基础上讨论基于 SDN 的负载均衡技术的优势及可行性。在此基础上，本文提出了一种基于数据中心网络胖树拓扑的动态负载均衡算法 LCLB。根据胖树的特点，缩小选择路径的范围，并在所计算路径中综合考量整个路径中的链路指标值，根据所占权重不同计算出相应的路径花销，最后选择花销值最小的路径，更好的解决了 Dijkstra 算法中的可能引起的路径拥塞问题，为数据中心的负载均衡策略提供了新的解决方案。针对本文完成的主要任务有如下几点：

（1）本文对 SDN 技术进行了深入的研究，对 SDN 的架构及其他重要的概念做了详细的分析，介绍了现阶段存在的负载均衡算法并通过对现有负载均衡算法的对比分析提出了基于链路多指标的最小花费负载均衡算法。

（2）本文通过对控制器中的链路发现模块、拓扑管理模块、拓扑计算模块、信息统计模块、负载均衡模块及流表下发模块进行设计与开发，实现了 LCLB 负载均衡算法，经过大量模拟实验后，通过对负载均衡性能的评估指标的对比和分析，验证了算法在负载抖动率及平均丢包率上的优越性。

6.2 未来展望

本文虽然为数据中心网络中的胖树拓扑提出了一种更好的负载均衡算法，但是由于本人时间精力的不足和研究水平的限制，本文还有很多可以提高改进之处：

（1）本文进行的实验为模拟仿真实验，在后续的工作中我们将在真实网络环境中进行验证。

（2）本文是基于数据中心的胖树拓扑提出的负载均衡算法，下一步我们将验证其在其他网络拓扑中的适用性。

参考文献

[1]2016 年中国互联网产业综述与 2017 年发展趋势.http://www.miit.gov.cn/n1146290/n1146402/n1146455/c5454725/content.html

[2] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008,38(2):69−74. [doi: 10.1145/1355734.1355746]

[3] Surhone L M, Tennoe M T, Henssonow S F. Global Environment for Network Innovations[M]. Betascript Publishing, 2010.

[4] Stanford University. Clean slate program. 2006.http://cleanslate.stanford.edu/

[5] Casado M, Freedman M J, Pettit J, et al. Ethane: taking control of the enterprise[M]. ACM, 2007.

[6]Jain S, Kumar A, Mandal S, et al.B4:experience with a globally-deployed software defined wan[J]. AcmSigcomm Computer Communication Review, 2013, 43(4):3-14.

[7] Hong C Y, Kandula S, Mahajan R, et al. Achieving High Utilization with Software-Driven WAN[J]. AcmSigcomm Computer Communication Review, 2013, 43(4):15-26.

[8] Handigol N, Seetharaman S,Flajslik M,et al. Plug-n-Serve: Load-
-Balancing Web Traffic using OpenFlow[C]// ACM SIGCOMM. ACM, 2009.

[9] Wang R, Butnariu D, Rexford J. OpenFlow-based server load balancing gone wild[C]// Usenix Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. USENIX Association, 2011:12-12.

[10]Niranjan Mysore R, Pamboris A, Farrington N, et al. Portland: A scalable fault-tolerant layer 2data center network fabric[C].AcmSigcomm Computer Communication Review. 2009, 39(4):39-50.

[11] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera:Dynamic Flow Scheduling for DataCenter Networks[C].Nsdi. 2010, 10: 19-19.

[12] Li Y, Pan D. OpenFlow based load balancing for Fat-Tree networks with multipath support[C]//Proc. 12th IEEE International Conference on Communications (ICC'13), Budapest, Hungary. 2013: 1-5.

[13] Leiserson C E. Fat-trees:universialnetworks for hardware-efficient super computing[J]. Computers, IEEE Transactions on,1985,100(10):892-901

[14] 李彬先. 基于 Openflow 的虚拟交换技术的研究[D]. 山东大学, 2015.

[15]Open Network Foundation[EB/OL].2011.http://www.opennetworking.org

[16] Juniper.OpenContrail[EB/OL].http://www.opencontrail.org

[17]Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks[J]. AcmSigcomm Computer Communication Review, 2008, 38(2):69-74.

[18]OpenFlowSwitch Specication Version 1.3.0.June 26,2012.https://

www.opennetworking.org

[19]OpenFlow 入门资料汇总（OpenFlow、SDN、NOX 等，多为网络文章）《互联网文档资源（http://www.360doc.co）》

[20]张仕,黄林鹏.基于 OSGi 的服务动态演化[J].软件学报,2008.19(5):1201-1211.

[21]Nierbeck A, Goodyear J, Edstrom J, et al.Apache Karaf Cookbook[C]// Packt Publishing, 2014.

[22] Batalle J, Riera J F, Escalona E, et al. On the Implementation of NFV over an OpenFlow Infrastructure: Routing Function Virtualization[C]// Future Networks and Services. IEEE, 2013:1-6.

[23] Cai Z, Cox A L, Ng T S E. Maestro: A System for Scalable OpenFlow Control[J]. Cs.rice.edu, 2011.

[24] 黎进都, 时向泉. 基于 Openflow 的 FloodLight 控制器实现研究[J]. 中国电子商情 通信市场, 2014(1):70-75.

[25] Mininet.2014.http://mininet.org/

[26] De Oliveira R L S, Shinoda A A, Schweitzer C M, et al. Using Mininet for emulation and prototyping Software-Defined Networks[C]// IEEE Colombian Conference on Communications and Computing. IEEE, 2014:1-6.

[27] 魏祥麟, 陈鸣, 范建华,等. 数据中心网络的体系结构[J]. 软件学报, 2013(2):295-316.

[28] Wang R, Butnariu D, Rexford J. OpenFlow-based server load balancing gone wild[C]// Usenix Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. USENIX Association, 2011:12-12.

[29] Koerner M, Kao O. Multiple service load-balancing with OpenFlow[C]

// IEEE, International Conference on High PERFORMANCE Switching and Routing. IEEE, 2012:210-214.

[30] Li Y, Pan D. OpenFlow based load balancing for Fat-Tree networks with multipath support[C]//Proc. 12th IEEE International Conference on Communications (ICC'13), Budapest, Hungary. 2013: 1-5.

[31] 吴宇文. 基于 OpenFlow 的网络负载均衡算法的研究与设计[D]. 华东师范大学, 20.14

[32] 陈云. 基于 SDN 的数据中心网络动态负载均衡研究[D]. 湖南师范大学, 2015.

[33] Mahdavi J, Paxson V. IPPM metrics for measuring connectivity[J]. IetfRfc, 1999:2678.

[34] 黄正兴, 苏旸. 基于链路性能分析的网络安全态势评估[J]. 计算机应用, 2013, 33(11):3224-3227.

[35] Garzon M, Gao Y, Rose J A, et al. In vitro implementation of finite-state machines[M]// Automata Implementation. Springer Berlin Heidelberg, 1998:56-74.

[36] Benson T, Anand A, Akella A, et al. Understanding data center traffic characteristics[C]// ACM Workshop on Research on Enterprise NETWORKING. ACM, 2009:65-72.

[37] Kandula S, Sengupta S, Greenberg A, et al. The nature of data center traffic: measurements & analysis[C]// ACM SIGCOMM Conference on Internet Measurement. ACM, 2009:202-208.

致谢

两年的研究生生活匆匆而过，转眼学生时代就要到达终点。从考研到读研，一路的艰辛与快乐都历历在目，虽然在北邮的时间只有两年，但这里带给我的美好回忆和宝贵财富将令我铭记一生。在论文即将完成之际，对在我学习和生活中给予我指导和帮助的老师、家人、朋友表示由衷的感谢！

首先，特别要感谢我的导师——王东滨副教授。王老师治学严谨、品格高洁，不仅在学术方面给予了我很多指导，也在生活上给予了我很多关怀。王老师在生活中和善待人、在项目中认真负责、在科研中一丝不苟，一言一行无不为我们树立了榜样。两年来您如春风化雨般的教导我将永远铭记在心，并将在未来的工作和生活中深深的影响着我。

其次，我还要感谢李祺老师，如果没有李老师我将与北邮擦肩而过，您对我的鼓励和帮助会让我永远记在心中。

再次，我要感谢实验室的和寝室的兄弟姐妹们，感谢你们一直以来对我的帮助与包容，与你们一起学习、一起生活、一起加班的日子让我感到充实而快乐。虽然我们即将别离，但一起奋斗的岁月永远不会磨灭！

最后，特别感谢我的父母，他们给了我前进的动力和坚实的后盾，我会用努力来回报你们默默的付出。感谢姚景科同学，在任何时候都给我前进的勇气，让我无所畏惧！

最后的最后，感谢评审老师百忙之中对论文的评阅！

攻读学位期间取得的研究成果