班 级 1501013
学 号 15010130034

西安电子科技大学

本科毕业设计论文

题 目 基于 SDN 的网络资源配置方法的探索

学 院 通信工程学院

专 业 通信工程

学生姓名 樊冰超

导师姓名 尚 韬

摘　要

随着云计算、大数据、虚拟化等技术的发展以及它们在数据中心的广泛应用，网络流量急剧增加，业务种类不断丰富，为了满足不同业务对宽带、时延和可靠性等方面的需求，网络拓扑结构也越来越复杂。软件定义网络（SDN）是近年来备受关注的一种新型的网络体系结构，它打破了传统网络的技术壁垒，让网络技术走向开放，代表着网络产业的未来。

SDN 通过将网络控制与网络转发解耦来构建开放可编程的网络体系结构，具有集中管理和可编程性的优点，已成为管理包括云服务器和数据中心在内的大规模网络的一种流行模式。单一集中式控制器所面临的可扩展性和可靠性问题使得分布式控制器体系结构应运而生。分布式控制器体系结构存在的问题之一是 SDN 交换机与控制器之间的静态映射，这很容易导致控制器之间的负载分布不均匀。此外，如果存在多个过载控制器，那么在单次负载均衡操作中只能处理一个具有最大负载的控制器，从而会降低负载均衡的效率。针对这些问题，我们首先对 SDN 的相关技术知识做了详细的介绍，在此基础上，我们提出了一种基于响应时间的多个 SDN 控制器的负载均衡策略，该策略考虑了实时响应时间随控制器负载变化的特点。通过选择合适的响应时间阈值，同时处理多个过载控制器，可以很好的解决 SDN 控制平面上有多个过载控制器的负载均衡问题。然后，我们将自己所提出的算法与已有的算法进行了分析比对，比较了两种算法的优劣性。

关键词：软件定义网络 分布式控制器 负载均衡 响应时间 交换机迁移

ABSTRACT

With the development of cloud computing, big data, virtualization and other technologies as well as their wide application in data centers, network traffic has increased dramatically and business types have been constantly enriched. In order to meet the needs of different services for broadband, delay, reliability and other aspects, network topology structure has become more and more complex. Software-defined networking (SDN) is a new network architecture which has attracted much attention in recent years. It breaks the technical barrier of traditional network and makes network technology open, representing the future of network industry.

SDN builds an open and programmable network architecture by decoupling network control and network forwarding. SDN has become a popular paradigm for managing large-scale networks including cloud servers and data centers because of its advantages of centralized management and programmability. The issues of scalability and reliability that a single centralized controller suffers makes distributed controller architectures emerge. One of the problems in distributed controller architecture is the statically configured switch-controller mapping, easily causing uneven load distribution among controllers. Additionally, if there are several overloaded controllers, just one controller with the maximum load can be addressed within a single load-balancing operation, reducing load-balancing efficiency. To address these problems, we first made a detailed introduction to the relevant technical knowledge of SDN, on this basis, we propose a load-balancing strategy of multiple SDN controllers based on response time, considering the changing features of real-time response times versus controller loads. By selecting the appropriate response time threshold and dealing with multiple overloading controllers simultaneously, it can well solve load-balancing problem in SDN control plane with multiple overloaded controllers. And then, we analyzed and compared the algorithm we proposed with the existing algorithm, and compared the advantages and disadvantages of the two algorithms.

Keywords: Software-defined networking distributed controllers load-balancing response time switch migration

目录

第一章 绪论

1.1 课题背景

目前的互联网体系架构已经运行了超过 40 年，随着云计算、人工智能、大数据等科学技术在互联网领域上运用的普遍化，网络的规模在急剧的膨胀，各种业务类型也不断增加，这使得网络环境变得越来越复杂。人们开始思考如何设计新的网络体系架构，软件定义网络(SDN)的出现为解决现有的网络问题提供了新的思路。

SDN 之所以是一种新型的网络体系结构，是因为它突破了传统网络体系结构的设计思想。SDN 的设计理念是将控制平面与数据转发平面解耦，让网络可以面向业务层编程。与传统网络相比，SDN 不仅提高了网络的灵活性，还让网络的管理水平有了很大的提升。这让网络运营商不仅可以更加便捷地部署网络，而且让之后的网络维护成本有了非常显著的降低。

SDN 的集中式的控制平面体系结构可以带来许多好处，它允许网络由应用程序编程并由一个中央实体控制。然而，与其他集中式系统一样，单一集中式控制器也面临着可扩展性和可靠性问题。对于由数十万台服务器组成的大型网络场景，单个集中式控制器很难管理这些网络拓扑。并且，随着 SDN 中交换机数量的增加，集中式控制器可能无法处理来自交换机的所有请求。此外，由于单点故障，SDN 控制器的故障会导致整个网络瘫痪。因此，下一个工作是使用多个物理分布式 SDN 控制器来提高系统的可扩展性和可靠性，同时保持集中式系统的简单性。

现有多控制器体系结构存在的问题之一是 SDN 交换机与控制器之间的静态映射，使得控制平面无法适应流量变化。真实网络可能在时间维度(流量在一天的不同时间甚至更短的时间尺度内变化)和空间维度(流量在网络的不同位置变化)上都表现出巨大的变化。如果 SDN 开关控制器映射是静态的，那么巨大的变化可能导致控制器之间的不均衡，即，有些控制器超负荷，有些控制器未得到充分利用。过载的控制器将以增加的延迟响应切换请求，从而降低用户体验的质量。因此，开关和控制器之间的动态映射可以克服不均衡并减少连接设置延迟，方法是将一些开关从过载的控制器迁移到负载较轻的其他控制器。

动态交换机迁移是一种在分布式控制器之间实现负载均衡有效且简单的方法。通过动态交换机迁移，可以有效地提高分布式控制器的性能和可扩展性。Dixit 等

人[1]首先提出了一种有效的协议(ElastiCon)来支持跨多个控制器的交换机迁移。在该协议中，选择最近邻控制器来作为接收负载转移的迁移控制器。由 Zhou 等人[2]提出了一种动态自适应负载均衡（DALB）算法。通过将负载值与自适应负载收集阈值进行比较，选择最大负载控制器。然后，选择一个重负载的交换机迁移到一个低负载的控制器下。在负载均衡过程中，通常需要判断控制器的负载何时不均衡，并对如何进行负载转移做出有效的决策。因此，选择重载控制器时需要一个阈值。此外，如果我们在多个控制器之间追求绝对的负载均衡，可能会频繁的执行交换机迁移。迁移成本是不可避免的，在交换机迁移期间，消息交换成本不可忽略。

为了避免频繁的不必要操作，我们充分利用响应时间和控制器负载的变化特性，提出一种新的负载均衡方案，即基于响应时间的 SDN 多控制器负载均衡策略。如果控制器对每个流请求都有正常的响应时间，则迁移交换机的过程是不必要的和不需要的。如果它的响应时间显著增加，我们将评估为控制器负载接近其能力瓶颈。因此，应该触发有效的负载均衡方法。此外，如果有多个过载控制器，一次操作只能处理负载最大的控制器，这会负载均衡的降低效率。为了解决这个问题，我们设计了一种只需一次操作就能处理多个过载控制器的方法。

1.2 国内外研究现状

本节主要回顾了分布式控制器和负载均衡方法的最新研究成果，这些成果支持我们的研究背景及其理论基础。主要从 SDN 负载均衡方法、SDN 分布式控制器和 SDN 控制平面的负载均衡[3]三个方面进行了介绍。第一个方面介绍了传统的负载均衡方法和基于 SDN 的负载均衡方法，然后详细介绍了 SDN 控制平面的分布式体系结构及其面临的挑战，最后一个方面介绍了 SDN 控制平面负载均衡的一些研究进展。

1.2.1 负载均衡方法

在传统网络中，负载均衡有两层含义。第一层含义是将单个重负载的运算分担到多台节点设备上作并行处理，每个节点设备将各自的运算处理结束后，然后把结果进行汇总并把其返回给用户，这使得整个系统的处理能力有了极大的提升，这就是我们常说的集群(clustering)技术。第二层含义就是进行大量并发访问或将网络数据包分配给多个节点设备上进行处理，以减少用户响应时间，这主要针对的是

FTP 服务器、Web 服务器等网络应用。从网络组成来看，可以将网络负载均衡分为网络链路负载均衡和服务器负载均衡。传统的网络负载均衡技术按照实现方式可以分为硬件负载均衡和软件负载均衡。硬件负载均衡是使网络基础设备符合数据中心的网络拓扑，避免链路拥塞。软件负载均衡主要是为了提高链路带宽的利用率，合理调度网络流量。相比硬件方式，软件方式实现成本会比较低。

然而，在传统网络中，由于难以获得整个网络的状态，全局负载均衡策略的实现并不容易。此外，传统的负载均衡方法也难以适应网络状态[4]的变化和调整。SDN 作为一种新的网络模式，它的核心思想就是要实现控制平面与数据平面的解耦，并将可编程网络交换机的控制权限外包给软件控制器。SDN 控制器通过北向接口向业务层提供 API，通过南向接口控制基础层的网络设备。SDN 能够根据用户不同的需求以及全局网络的拓扑结构，对网络资源进行灵活动态的分配。对于基础层通过标准的南向协议与网络基础设施通信；对于应用层，它通过一个开放的北向接口提供对网络资源的控制。由于 SDN 的可编程性和灵活性，传统网络中通常使用控制器作为负载均衡器进行流量调度和负载管理。软件定制方法可以降低对硬件设备的要求，因此，SDN 可以用来实现传统网络中的链路和服务器负载均衡。模糊综合评价机制[5]是一种基于 SDN 的路径负载均衡解决方案，有效地均衡了流量，避免了由于链路失效而导致的意外故障，提高了网络路径的利用率和可靠性。参考文献[6]提出了一种基于 OpenFlow 的数据中心网络动态负载均衡策略，有效地利用了网络资源容量。基于服务器的响应时间[7]的负载均衡也是一种有效的服务器负载均衡方案，与传统的方案相比，该方案具有更好的负载均衡效果。

1.2.2 SDN 多控制器

在某些网络情况下，集中式 SDN 控制器(如 NOX)可以很好地用于传统的网络流量控制和管理。然而，作为网络流量的管控中心，SDN 控制器的性能也受到其处理负载的影响。随着 SDN 在云计算、大数据等应用场景中的部署，对单个 SDN 控制器的海量流量请求，使得 SDN 控制平面的性能和安全性成为整个网络的潜在瓶颈。

SDN 分布式控制器体系结构可以很好地限制包括 Onix[8]在内的集中式控制器的控制层处理能力。Beacon[9]采用多线程设计，提高了控制平面各控制器的性能。Kandoo[10]通过 SDN 节点集群实现分布式控制平面，提高了控制平面的可扩展性和可靠性。参考文献[11]为基于流的大规模 SDN 网络设计了混合分层控制平面，提高

了 SDN 控制平面的可扩展性。这些方法有效地解决了单个集中控制器的性能瓶颈问题。然而，多个 SDN 控制器可能会带来一些新的挑战，包括由于网络流量分布不均导致的多个控制器之间的负载不均衡以及控制信息同步问题。

1.2.3 SDN 控制器负载均衡

在 SDN 分布式控制平面上，OpenFlow 交换机与 SDN 控制器之间的映射是静态的，静态交换机-控制器映射会导致负载不均衡，降低总体资源利用率，导致性能不佳。

在 OpenFlow 1.3 协议[12]中，交换机可以连接到多个 SDN 控制器，允许其负载在不同的控制器间迁移。DiXit 等人[1]首次提出了一种用于 SDN 多控制器负载均衡的切换迁移协议，在该协议中，作者提出了一种分布式最近邻迁移算法，通过选择最近邻控制器来接收负载转移以节省迁移时间，但这可能会导致新的负载不均衡。Zhou 等人[2]提出了一种动态自适应负载均衡算法（DALB），该算法基于具有自适应负载阈值的分布式体系结构。Liang 等人[13]提出了一种基于交换机迁移的集群控制器动态负载再均衡算法，该算法还支持控制器故障转移，避免了单点故障问题，但是显著增加了控制层的响应时间。Cheng 等人[14]设计了一种资源利用率最大化迁移算法（MUMA）,当负载分布不均匀时，过载控制器随机选择一个交换机进行迁移，但是在迁移之后，过载控制器仍然可能是过载的。Yu 等人[15]提出了一种基于负载通知策略的负载均衡机制，每个控制器定期向其他控制器报告负载信息，以便过载控制器不再在做出本地决策之前收集其他所有控制器的负载信息。DSMA 算法是一种交换机动态迁移算法，从全局角度优化交换机与控制器之间的映射关系，把握全局网络情况，选择最为合适的交换机迁移策略，提高了负载均衡效率，一定程度上减少了系统的通信开销,然而 DSMA 算法只考虑了迁移目标优化，忽略了迁移产生的通信开销。上述研究不考虑多个过载控制器，对控制器负载没有细粒度的判断。

在 SDN 多控制器负载均衡研究中，参考文献[16][17]一直在追求交换机迁移决策的效率。SMDM[16]提出了一种迁移效率模型，在迁移成本和负载均衡率之间进行权衡。该方案精心设计以选择合理的迁移配对，但是由于负载均衡时间较长，迁移后也可能导致新的负载不均衡。参考文献[17]提出了一种基于交换机组的负载均衡方案，它根据超出的工作负载来选择迁移交换机组，从而有效减少迁移决策的数量，但是它增加了迁移的交换机数量且追求的是总体的负载均衡率，这会带来额外的不必要

的迁移成本。

1.3 本论文的主要研究成果和内容安排

1.3.1 主要工作

在本文中，我们主要通过交换机迁移来设计逻辑负载均衡策略。我们充分利用响应时间和控制器负载的变化特性，提出一种新的负载均衡方案，即基于响应时间的 SDN 多控制器负载均衡策略。

综上所述，本文的主要工作如下：

1.对 SDN 控制器负载不均衡问题的交换机迁移方案做了验证，证明交换机迁移对负载均衡是有效的。

2.验证了响应时间随控制器负载变化的特性，能够准确判断控制器响应时间随负载增加的响应趋势。

3.我们使用响应时间来测量控制器的负载，并得到一个合适的响应时间阈值，以便更好地检测过载控制器，使它们可以提前得到处理。

4.为了提高负载均衡效率，我们提出了一种新的交换机迁移算法，如果有多个控制器超载，在一次负载均衡操作中处理多个过载控制器。

1.3.2 论文结构

第一章为绪论，介绍了本课题的研究背景，阐述了相关技术的国内外研究现状，并简要介绍了本文的工作。

第二章介绍了本文所涉及的相关技术的概念，包括 SDN 架构、OpenFlow 技术、Floodlight 控制器、SDN 控制器负载均衡技术、实验环境 Mininet 以及所用工具软件。

第三章对 SDN 控制器负载不均衡问题做了描述，并对交换机迁移方案进行了验证。

第四章详细介绍了我们关于该课题的工作，主要包括模型的建立和算法的设计。

第五章是本文的方案分析与比较部分，对不同的负载均衡方案进行理论分析对比并进行性能评估。

第六章总结了本文的主要研究工作，并对未来工作做了展望。

第二章 相关工作概述

2.1 SDN 架构

根据业界的通用性理解，SDN 作为一种新型的网络体系结构，它不仅实现了数据与控制分离，而且让网络控制可以直接编程，其基本架构如图 2.1 所示。SDN 将集中式的控制平面和分布式的转发平面相互分离。控制平面利用南向接口集中式的控制基础层上的网络设备，并向应用层提供灵活的网络可编程能力。具备以上特点的网络架构都可以被认为是广义的 SDN 技术。

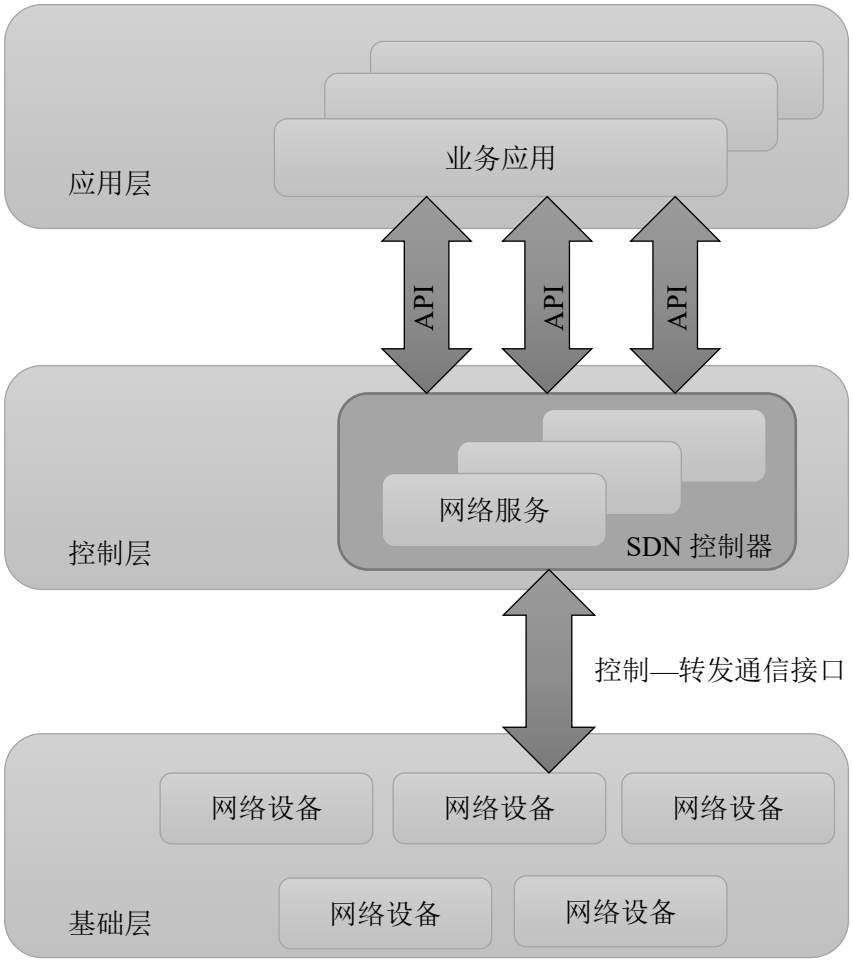

图 2.1 SDN 的基本架构

应用平面由若干的 SDN 应用（SDN Application）构成。SDN 应用作为受用户关注的应用程序，这些应用能够利用可编程的方式，完成网络请求行为并将其提交给控制器。一个 SDN 应用可以使用多种不同的北向 API，SDN 应用也可以对外提供

北向代理接口。它对自己本身的功能进行抽象、封装，封装后的接口就形成了更高级的北向接口。

SDN NBI 提供了 SDN 中控制器和开发者之间的交互能力。它主要向上层提供一些基本的、通用的网络逻辑抽象，使应用平面上的应用可以对网络的行为进行直接控制。这个接口应该是一个开放的、与厂商无关的接口。

控制平面的核心即图 2.1 中所示的 SDN 控制器。SDN 控制器是一个逻辑上集中的实体。它主要承担两个任务：第一个任务是将 SDN 应用层请求转换到 SDN 数据路径，第二个任务是向上为开发者提供网络高层的的逻辑抽象和业务模型（可以是状态，也可以是事件）。一个 SDN 控制器主要由三个部分组成，分别是 NBI 代理+、SDN 控制逻辑与 CDPI 驱动。SDN 控制器只要求逻辑上的完整，因此它可以由多个控制器实例协同组成，也可以是层级式的控制器集群。所以从地理位置上来讲，SDN 控制器实例有两种分布方式，第一种方式是将所有 SDN 控制器实例放在同一地理位置，第二种方式是将多个 SDN 控制器实例分散的放在不同的地理位置。

SDN CDPI 是控制平面和数据平面之间的接口。它应该是一个开放的、与厂商无关的接口。它提供的主要功能主要包括以下几个方面：

（1）对所有转发行为进行控制；

（2）设备性能查询；

（3）统计报告；

（4）事件通知。

数据平面由一些网元(Network Element)构成。网元是一个将被管理的资源在逻辑上进行抽象而形成的集合。一个网元不仅可以包含一个 SDN 数据路径，也可以包含多个 SDN 数据路径。每个 SDN 数据路径是一个逻辑上的网络设备，它没有控制能力，只是单纯用来转发和处理数据，它在逻辑上代表全部或部分的物理资源，可以包括与转发相关的各类计算、存储、网络功能等虚拟化资源。同时，一个网元应该支持多种物理连接类型（例如分组交换和电路交换），支持多种物理和软件平台，支持多种转发协议。一个 SDN 数据路径包含控制数据平面接口(CDPI)代理、转发引擎(Forwarding Engine)表和处理功能(Processing Function)3 个部分。

表 2.1 给出了传统网络与 SDN 的比较，更好地展示了 SDN 的特点和优势。

表 2.1 传统网络与 SDN 的比较

传统网络	SDN
物理网络	虚拟网络
硬件决定网络	软件决定网络
垂直集成，软硬件紧密耦合	水平集成，软硬件分离
智能在核心	智能在边缘
组网要对硬件进行配置	组网只需要软件编程
网络主体复杂、刚性	网络主体简单、抽象
网络新功能开发时间长	网络新功能开发时间短
网络创新难，网络封闭	网络创新容易，网络开放

2.2 OpenFlow 技术概述

2.2.1 OpenFlow 协议背景

2008 年，斯坦福大学成立了一个名为 Clean Slate 的特别工作小组，这个小组在 2009 年开发出了一个可以满足 SDN 网络转控分离架构的标准，即 OpenFlow 1.0。同时该小组还开发出了 OpenFlow 的参考交换机和 NOX 控制器。OpenFlow 标准协议允许控制器直接访问和操作网络设备的转发平面，这些设备可以是物理设备，也可以是虚拟的路由器或者交换机。转发平面则采用基于流的方式进行转发。

OpenFlow 1.0 问世后不久就引起了业界关注。2011 年 3 月 21 日，德国电信、脸书、谷歌、微软、雅虎等公司共同成立的了 ONF（Open Networking Foundation）组织，旨在推广 SDN，并加大 OpenFlow 的标准化力度。芯片商 Broadcom，设备商 Cisco、Juniper、HP 等，各数据中心解决方案提供者以及众多运营商纷纷参与。该组织陆续制定了 OpenFlow 1.1、1.2、1.3、1.4 等标准，目前仍在继续完善中。随着越来越多的公司加入 ONF，OpenFlow 及 SDN 技术的影响力也越来越大。

在表 2.2 中我们给出了 OpenFlow 协议的发布时间线。

表 2.2 OpenFlow 协议发布时间线

2009.12	v1.0	单流表、IPv4
2011.2	v1.1	流水线、组表
2011.12	v1.2	多控制器、IPv6 基本头
2012.6	v1.3	流的计量、IPv6 扩展头
2013.10	v1.4	光端口、流表满载的处理
2015.1	v1.5	出向流表、通用流表逻辑

2.2.2 OpenFlow 协议架构

OpenFlow 架构的主要组件有 OpenFlow 控制器、OpenFlow 交换机和 OpenFlow 协议，如图 2.2 所示，其特点有：

- 控制面和数据面分离；
- 采用标准化协议描述控制器与网络组件代理之间的状态。
- 通过可扩展的 API 来建立集中式视图，并基于此实现网络的可编程性。

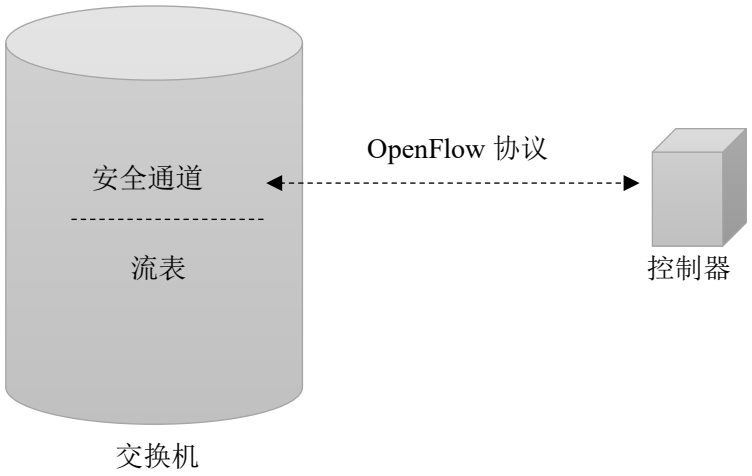

图 2.2 OpenFlow 协议架构

OpenFlow 控制器类似于大脑，用于制定所有基于业务流的智能决策，并将些决策发送给 OpenFlow 交换机。这些决策以指令的形式存在于流表中，典型的流信息的决策形式包括：添加、删除及修改 OpenFlow 交换机中的流表。通过配置 OpenFlow 交换机，将所有未知数据包发给控制器，也可以在 OpenFlow 流表中下发其他一些指令。

OpenFlow 交换机类似于现在网络中使用的典型的交换机，但不包含智能软件。OpenFlow 交换机可以分为如下两类：

（1）纯 OpenFlow 交换机：这种交换机只支持 OpenFlow 协议。

（2）混合 OpenFlow 交换机：这种交换机同时支持传统以太网协议和 OpenFlow 协议。

在 OpenFlow 交换机中，OpenFlow 控制器管理硬件的流表，交换机主要用于数据平面上的转发。控制通路有 OpenFlow 控制器管理，而数据通路则建立在由 OpenFlow 控制器编制的 ASIC 指令的基础上。OpenFlow 协议的最终目标是实现对数据通路的程序指令，但是 OpenFlow 实现数据通路指令的方法却有所不同。OpenFlow 是客户端服务器技术和各种网络协议的融合。OpenFlow 协议集目前被分为以下两部分：

（1）线路协议：用于建立控制会话，定义一个用于交换流量变动信息和收集统计信息的消息结构以及交换机的基本结构（端口和表）。1.1 版本增加了对多重表、存储动作执行和元数据传输的支持，最终在交换机中创建用于处理控制流的逻辑管道。

（2）配置管理协议：采用基于 NETCONFIG 的模型（使用 Yang 数据模型）的 OFCONFIG 协议，为特定控制器分配物理交换机端口，定义高可靠性（主用/备用）和当控制器连接失败时的行为。虽然 OpenFlow 可以使用 OpenFlow 命令/控制进行基本的配置操作，但它目前还不能启动或者维护一个网络组件。

2.3 Floodlight 控制器

2.3.1 Floodlight 控制器概述

Floodlight 是 Apache 授权并且基于 JAVA 开发的企业级 OpenFlow 控制器，它的稳定性、易用性已经得到 SDN 专业人士以及爱好者们的一致好评，并因其完全开源，这让 SDN 网络世界变得更加有活力。控制器作为 SDN 网络中的重要组成部分，能集中地灵活控制 SDN 网络，为核心网络及应用创新提供了良好的扩展平台。

2.3.2 Floodlight 控制器架构

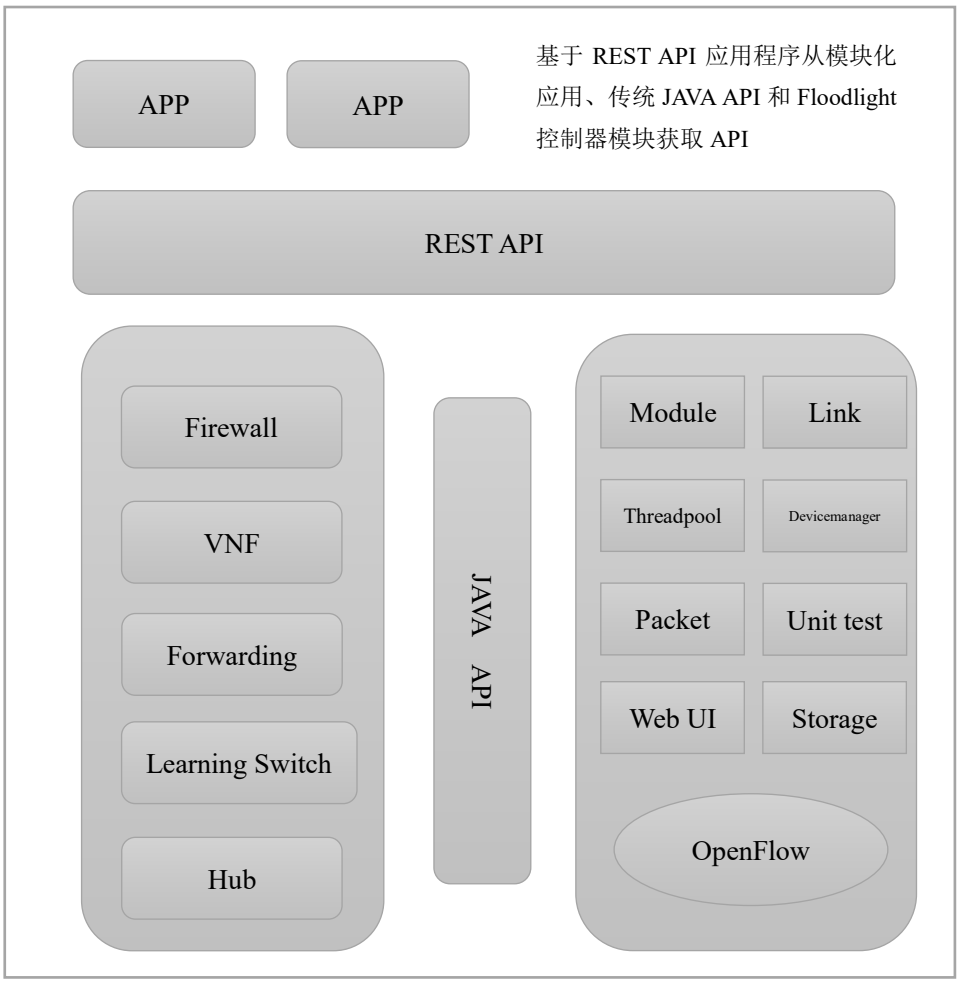

图 2.3 Floodlight 模块架构

Floodlight 整体架构主要包括各种功能模块、一些 JAVA 的 API 以及运行在其上的应用。REST 接口是控制器和应用之间的交互接口，JAVA 接口主要用于模块之间的通信。Floodlight 控制器架构如图 2.3 所示。

Floodlight 控制器的模块框架在功能上包括控制器模块（核心控制器模块）和应用模块，应用模块又可以分为普通应用模块和 REST 应用模块二个部分。模块的子模块名称和各个子模块的具体功能如表 2.3 所示。

Floodlight Provider 作为核心模块，提供两个主要功能：处理与交换机之间信息交互。另一个功能是负责确定 OpenFlow 消息在各个监听模块之间的分发顺序。DeviceManagerImpl 模块负责追踪设备在网络中移动，为新的流确定目标设备。

表 2.3 Floodlight 控制器组件功能

组件类型	模块名称	功能
核心模块	FloodlightProvider	控制器核心，驱动其他各个功能模块
	DeviceManagerImpl	管理网络中的主机等终端设备
	LinkDiscoveryManager	维护管理网络中的链路
	TopologyService	维护拓扑信息
	RestApiServer	提供 Reset API 服务
	ThreadPool	为其他模块分配线程的线程池
	MemoryStorageSource	提供数据存储及变更通知服务
	Flow Cache	流缓存，用于控制器记录所有有效流
	Packet Streamer	提供数据分组流服务
应用模块	Firewall	防火墙
	LearningSwitch	交换机的二层转发功能
	VirtualNetworkFilter	建立虚拟模块网络
	ForWarding	数据分组转发
	Port Down Reconciliation	实现在端口关闭的时候处理网络中的流
REST 应用模块	Circuit Pusher	用于创建 2 台设备之间的虚链路
	OpenStack Quantum	支持 FloodlightOpenStack 的 Quantum 插件，用来管理网络

2.4 SDN 控制器负载均衡算法

在 SDN 的发展过程中，关于交换机与控制器的负载均衡算法有 DALB 算法[2]、DHA 算法、DSMA 算法[21]等。

DALB(Dynamic and Adaptive algorithm for controller Load Balancing)算法由周等人提出，该算法对于中心控制器收到硬件条件的限制，设计出了一种新的控制器从属关系，保证了控制器集群中的每个控制器的自主性，当控制器过载的时候才收集控制器的信息，以最大幅度减少收集信息而带来的开销。该算法是将过

载控制器的交换机迁移至负载最小的控制器，这很可能会导致单个控制器过载，导致进一步的负载不均衡。

DHA(Distributed Hopping Algorithm)算法对控制器与交换机之间的关系进行了重新定义，定义了负载阈值来避免一小部分的交换机会消耗控制器的资源的现象，同时要求了一个交换机只能由一个控制器管理。从实验结果来看，该算法提升了资源的平均利用率。

DSMA(Dynamic Switch Migration Algorithm)算法从全局角度优化了控制器与交换机之间的映射关系，根据全局的网络情况，动态地选择最合适的交换机迁移策略，在一定程度上提高了负载均衡的效率，然而该算法忽略掉了交换机迁移产生的通信通信开销。

2.5 实验环境

2.5.1 Mininet 仿真平台

Mininet 是由一些虚拟的终端节点（end-hosts）、交换机、路由器连接而成的一个网络仿真器，它采用轻量级的虚拟化技术使得系统可以和真实网络相媲美。

Mininet 可以很方便地创建一个支持 SDN 的网络：host 就像真实的电脑一样工作，可以使用 ssh 登录，启动应用程序，程序可以向以太网端口发送数据包，数据包会被交换机、路由器接收并处理。有了这个网络，就可以灵活地为网络添加新的功能并进行相关测试，然后轻松部署到真实的硬件环境中。目前，Mininet 已经作为官方的演示平台对各个版本的 OpenFlow 协议进行演示和测试。

Mininet 作为一个轻量级的软件定义网络的研发与测试平台得到了学术界的广泛关注，其主要特性包括以下五个方面：

1) 可以简单、迅速地创建一个支持用户自定义的网络拓扑，缩短开发测试周期；
2) 可以运行真实的程序，在 Linux 上运行的程序基本上可以都可以在 Mininet 上运行，如 Wireshark；
3) Mininet 支持 Openflow，在 Mininet 上运行的代码可以轻松移植到支持 OpenFlow 的硬件设备上；

4) Mininet 可以在自己的电脑，或服务器，或虚拟机，或者云（例如 Amazon EC2）上运行；

5) Mininet 提供 python API，简单易用。

2.5.2 实验工具

Cbench(Controller Benchmarker)是一种用于测试 OpenFlow 控制器性能的工具，通过不断循环产生新的流(Packet-in 消息)来测试控制器的处理能力。Cbench 的基本原理是模拟连接到控制器的一组交换机，发送 Packet-in 消息并查看 Flow-mod 等消息的下发，记录相关统计信息，以计算、衡量控制器性能指标。

2.6 本章小结

在本章中，主要介绍了 SDN 的相关知识，我们首先从 SDN 架构开始，了解 SDN 到底是什么,与传统网络有什么区别。接着对 SDN 中的 OpenFlow 协议进行了介绍，了解到 OpenFlow 协议的背景、版本以及工作方法。接着对 SDN 中的 Floodlight 控制器做了介绍，对 Floodlight 控制器的各个模块及其它们的主要功能做了了解。接着介绍了几种控制器负载均衡算法，最后对我们的实验平台 Mininet 以及实验工具做了简要的介绍。

第三章 SDN 控制器负载不均衡问题

3.1 问题描述与交换机迁移

在网络技术不断的发展过程中，网络带宽与网络访问量的增长速度要比服务器 CPU 的处理速度与内存访问速度的增长速度要大很多，这个时候服务器的负载能力就成为了网络的瓶颈。

在 SDN 网络中，它由 SDN 交换机和逻辑集中式的 SDN 控制器组成。当有流进入网络后，每个 SDN 交换机对流进行解析并与存储在其中的流表进行匹配。根据流表中的规则处理和传送分组，如果匹配成功，就按照流表中的操作项对流执行转发或丢弃操作；如果匹配失败，SDN 交换机就会把流封装成 Packet－In 消息上传给 SDN 控制器，让 SDN 控制器决定流的去向并生成新的流表然后下发给 SDN 交换机进行安装，SDN 交换机根据新的流表对流进行处理。

在绪论 1.2.2 中我们提到过为了解决单个集中控制器所存在的可扩展性问题，前人提出了分布式控制器体系结构的概念。下面我们将对分布式 SDN 控制器平面的负载不均衡问题做出详细的解释。

首先，我们解释 SDN 中的响应模式行为，如图 3.1。

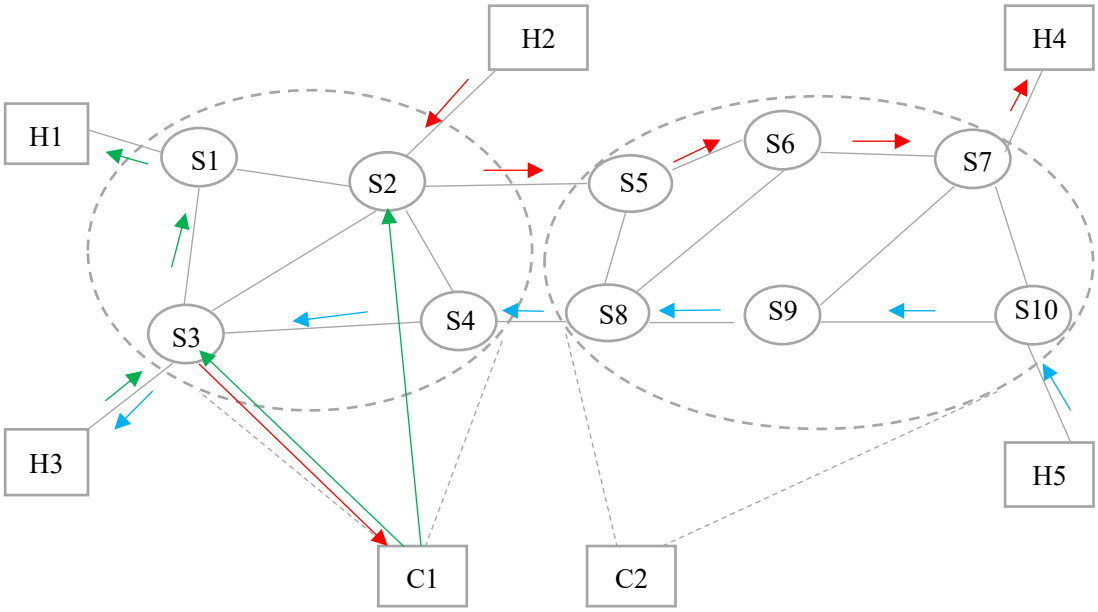

图 3.1 SDN 响应模式

在图 3.1 中，网络被划分成为了两个域，分别由控制器 C1 与 C2 控制。这时我们假设主机 H3 产生了新流 f1 到达了交换机 S3，交换机 S3 中不具有与流 f1 相

关联的流表，此时交换机 S3 就会发送一个 Packet-In 消息到达控制器 C1（图 3.1 中 S3 到 C1 的红色箭头）。然后控制器 C1 就会计算流 f1 的转发路径并将规则安装到它自己控制的交换机上面（假设流 f1 的转发路径是 S3→S2→第二个域，图 3.1 中 C1 到 S3 与 S2 的绿色箭头）。当流 f1 到达交换机 S5 的时候，交换机 S5 中也不具有与流 f1 相关联的流表，此时交换机 S5 就会发送一个 Packet-In 消息到达控制器 C2，然后控制器 C2 就会计算流 f1 的转发路径并将规则安装到交换机 S5、S6 和 S7 上面（假设流 f1 在第二个域的转发路径是 S5→S6→S7）。这就是 SDN 的响应模式行为。

下面我们假设 SDN 控制器负载不均衡的场景。假设有大量的新流到达了网络，如图 3.1：

a) 主机 H3 每秒生成 35 个新流到达主机 H1，这些新流的路径为 H3→S3→S1→H1（图 3.1 中的短绿色箭头）；
b) 主机 H2 每秒生成 40 个新流到达主机 H4，这些新流的路径为 H2→S2→S5→S6→S7→H4（图 3.1 中的短红色箭头）；
c) 主机 H5 每秒生成 25 个新流到达主机 H3，这些新流的路径为 H5→S10→S9→S8→S4→S3→H3（图 3.1 中的短蓝色箭头）；

我们假设单个流的路径计算需要 α 单位的负载，在单个交换机中安装单个流的流规则安装需要 β 单位的负载。

所以在控制器 C1 中产生的负载有：

a) 绿色的流产生了 35 α 单位的路径计算负载和（35+35）β 单位的用于 S1 和 S3 的流表规则安装负载。
b) 红色的流产生了 40 α 单位的路径计算负载和 40 β 单位的用于 S2 的流表规则安装负载。
c) 蓝色的流产生了 25 α 单位的路径计算负载和（25+25）β 单位的用于 S3 和 S4 的流表规则安装负载。

控制器 C2 产生的负载有：

a) 红色的流产生了 40 α 单位的路径计算负载和（40+40+40）β 单位的用于 S5、S6 和 S7 的流表规则安装负载。
b) 蓝色的流产生了 25 α 单位的路径计算负载和（25+25+25）β 单位的用于 S8、S9 和 S10 的流表规则安装负载。

我们考虑到用于路径计算的负载要远大于规则安装的负载，所以我们假设 α =1，β =0.1，可以得到：

L_{C1}=(35+40+25) α +(35+35+40+25+25) β =116 单位/s

L_{C2}=(40+25) α +(40+40+40+25+25+25) β =84.5 单位/s

我们假设控制器 C1 和 C2 的负载阈值为 100 单位/s，这时我们发现控制器 C1 已经过载，控制器 C1 与控制器 C2 之间负载不均衡。根据 OpenFlow1.3 协议，控制器允许自己的交换机在不同的控制器之间进行迁移。所以我们对图 3.1 中的 SDN 区域进行新的划分，我们将交换机 S2 和 S4 划到第二个域，让它们由控制器 C2 进行控制，划分后的网络拓扑如图 3.2 所示。

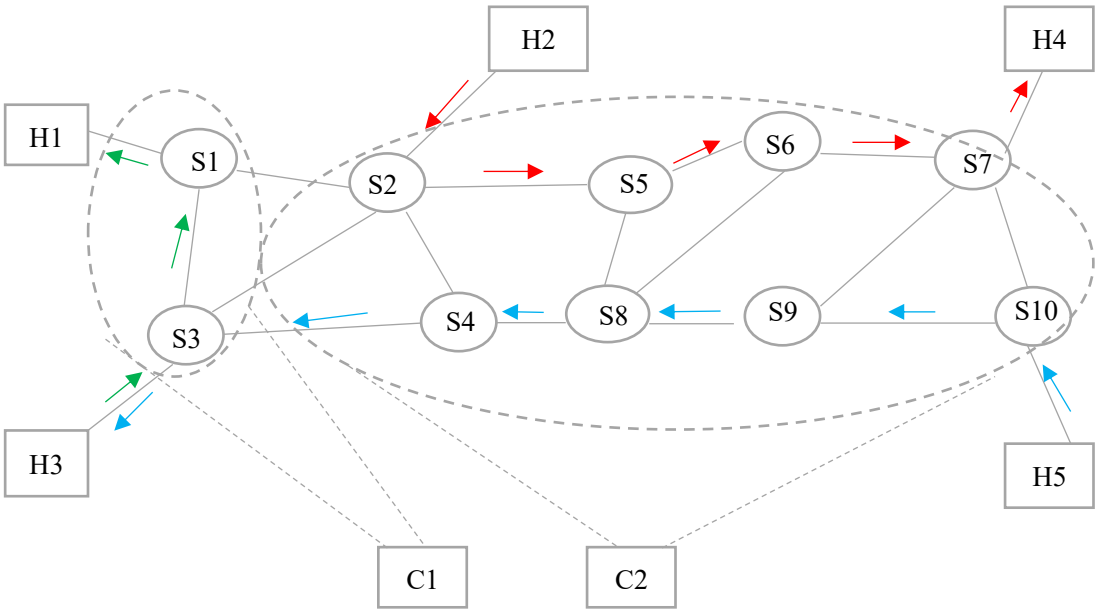

图 3.2 控制器与交换机之间的新映射

这时控制器 C1 与 C2 的负载分别为：

$L_{C1}=35\alpha+(35+35)\beta=42$ 单位/s

$L_{C2}=(40+25)\alpha+(40+40+40+40+25+25+25+25)\beta=91$ 单位/s

我们可以看到控制器 C1 处的负载显著降低了 63.79%，只有 42 单位/s，远远低于它的负载阈值，而在控制器 C2 处的负载只增加了 7.69%，并且控制器 C2 没有超过它的负载阈值。这说明在控制器之间进行交换机迁移非常有利于 SDN 控制器的负载均衡。[24]

3.2 本章小结

在本章我们对 SDN 控制器负载不均衡问题进行了详细的描述，并对已有的 SDN 交换机迁移方案进行了验证，证实了交换机迁移对于 SDN 控制器负载均衡是确实有效的。

第四章 算法设计

通常，在 SDN 网络中，交换机一旦接收到匹配非对应流表条目的包，就需要将包的头部封装到 PACKET_IN 消息中，并将其发送给主控制器进行路由和流表条目安装。PACKET_IN 消息的处理通常被认为是控制器负载中最突出的部分。因此，PACKET_IN 消息的巨大分布差异可能导致多个控制器之间的工作负载不均衡。随着响应延迟的显著增加，一些控制器可能达到性能瓶颈，而其他控制器处于正常负载或空闲状态。

在本节中，我们提出了基于响应时间的 SDN 多控制器负载均衡策略，用于在分布式 SDN 控制平面上均衡多个过载控制器，并根据响应时间对过载控制器进行细粒度判断。与其他交换机迁移方案一样，SDN 控制平面的负载均衡包括三个阶段:控制器负载不均衡的测量;选择过载控制器、主负载交换机和迁移控制器，进行迁移计划的决策;实现迁移计划，转移过载控制器的负载。

如图 4.1，我们建立一个由 N 个控制器 $C=\{C_1,C_2,\cdots,C_N\}$ 和 K 个交换机 $S=\{S_1,S_2,\cdots,S_K\}$ 组成的分布式系统模型，这些控制器具有相同的功能，将网络划分成了 N 个域。令 $Q_{C_i}\in S$ 表示由主控制器 C_i 管理的交换机集。

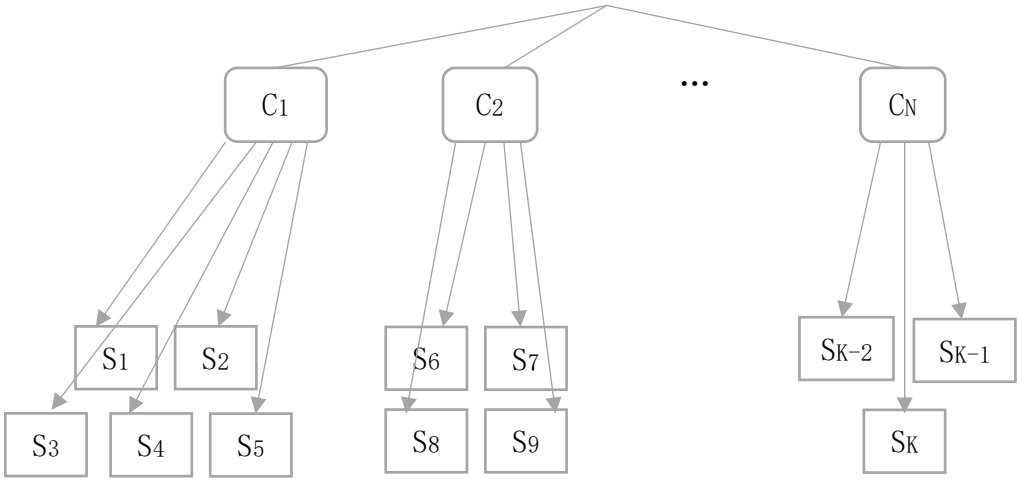

图 4.1 系统模型

4.1 响应时间采集和负载测量

我们的方案增加了一个新的模块，用于周期性统计和计算响应时间间隔 T。我们用 S_k 和 T_n 分别表示第 k 个交换机和第 n 个周期，t_{arrive} 表示 PACKET_IN 消息

到达的时间，t_{reply} 表示控制器响应 PACKET_OUT 消息或 FLOW_MOD 消息的时间。因此，单个 PACKET_IN 请求的响应时间如式(4-1)所示。

$$t_{response} = t_{reply} + t_{arrive} \quad (4-1)$$

以 $f_{S_kT_n}$ 表示 S_k 在 T_n 中的请求消息数量，并将其表示为 S_k 带来的工作负载 $Load_{S_kT_n}$，因此，可以用集合 Q_{C_i} 中这些交换机的和负载 $f_{C_iT_n}$ 来表示第 n 个周期内接收到的请求消息的总数或控制器 C_i 的工作负载 $Load_{C_iT_n}$，如式(4-2)所示。

$$Load_{C_iT_n} = \sum_{S_k \in Q_{C_i}} f_{S_kT_n} \quad (4-2)$$

以 $t_{S_{k\_response}}$ 表示 S_k 在 T_n 内发送的 PACKET_IN 请求的响应时间，因此，可以得到 C_i 在 T_n 内接收到所有 PACKET_IN 请求的总响应时间和单个 T 内中 PACKET_IN 请求的平均响应时间 $t_{C_iT_n}$，如式(4-3)所示。

$$t_{C_iT_n} = \frac{\sum_{S_k \in Q_{C_i}} t_{S_{k\_response}}}{Load_{C_iT_n}} \quad (4-3)$$

4.2 基于响应时间的适当阈值

与传统网络中的服务器负载均衡一样，合理设计和准确判断过载控制器是不可避免的，也是非常重要的。在以往的控制器负载均衡研究中，大多采用与控制器负载直接相关的指标作为阈值，如负载或平均负载。然而，这些指标的值一般为经验值或随机值，很难实现对负载不均衡的最佳检测。如果阈值过小，由于流量的暂态性，容易发生负载不均衡，导致均衡操作频繁，导致迁移成本高，网络稳定性差。然而，较高的阈值会导致较低的均衡率。也就是说，一些控制器处理能力变差，而另一些控制器仍处于相对空闲状态。因此，选择合适的阈值尤为重要，对负载不均衡的准确判断和对过载控制器的及时检测和处理具有重要意义。

从用户的角度来看，他们的请求消息的响应时间是决定网络质量的最关键因素。响应时间会影响网络延迟和网络服务质量。因此，在本小节的其余部分，我们将充分利用响应时间来选择合适的阈值，这也可以在均衡速率和迁移成本之间进行权衡。

首先，我们需要测试不同负载下的 SDN 控制器平均响应时间。我们选择一个 SDN 控制器， 通过 4.1 部分的方法得到单一 PACKET_IN 请求消息的平均响应时间，在这个测试中，选择基于 JAVA 的 Floodlight 控制器。将 PACKET_IN 到达速率从每秒 300 消息包增至每秒 3000 个消息包，间隔设为每秒 100 个消息包。通过多次

实验，选取了一些相对稳定的数据，得到了不同分组速率下 Floodlight 控制器的平均响应时间。通过对数据进行统计并拟合曲线，可以将这些数据绘制成曲线图，如图 4.2 所示。

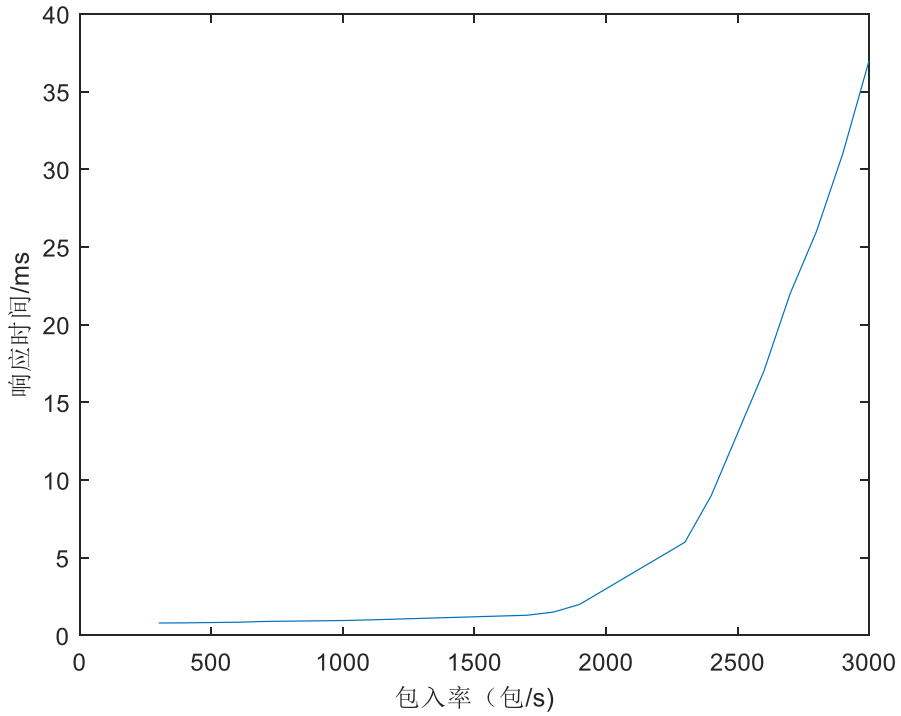

图 4.2 响应时间随控制器负载变化曲线

其次，从变化曲线可以得到响应时间随控制器负载的变化特征。一开始，控制器处于低负载状态，随着负载的增加，响应时间缓慢增加，变化范围小于 0.5ms。当控制器处于正常负载状态时，响应时间的变化加快。当控制器处于过载状态时，响应时间明显增加。

最后，根据响应时间的变化特征选择合适的响应时间阈值。随着控制器负载从正常状态增加到过载状态，响应时间增加得更快。通过计算响应时间变化的极值，可以得到合适的阈值 $t_{threshold}$。设 $g(t_{i-1})$为第$(i-1)$个周期的平均响应时间。因此，$g(t_i)$可以表示当消息包到达速率增加每秒 100 个包时，第 i 个周期的平均响应时间因此，我们可以很容易地计算出变化曲线在第 i 个周期结束时一阶和二阶的导数，如式(4-4)和(4-5)。

$$g'(t_i)=\frac{g'(t_i)-g'(t_{i-1})}{t_i-t_{i-1}} \qquad (4-4)$$

$$g''(t_i)=\frac{g'(t_i)-g'(t_{i-1})}{t_i-t_{i-1}} \qquad (4-5)$$

由上面的公式，我们可以得到变化曲线上极值点的合适阈值 $t_{threshold}$。在此之前，虽然响应时间随着负载的增加而增加，但是它的变化缓慢而轻微。也就是说，控制器负载的增加对用户消息请求的影响很小。但是在这一点之后，响应时间增长得更快，如果控制器的负载持续增加，响应时间将显著增加。因此，如果控制器的响应时间达到这个阈值，我们需要快速处理该控制器，以防止其响应时间快速增长。这样，我们就可以很好地选择合适的响应时间阈值来实现均衡速率和均衡成本之间的权衡。

4.3 检测和触发负载均衡算法

在 4.2 部分中，可以使用响应时间的突然和连续增长来选择负载均衡的最佳时间。在这里，我们需要做的是将所有控制器的响应时间与适当的阈值进行比较，以找出哪个控制器容易接近其性能瓶颈并趋向于过载状态。

设集合 $A=\{t_{C_1T_n}, t_{C_2T_n}, \cdots, t_{C_NT_n}\}$ 表示一个时间间隔内 N 个控制器的响应时间值，从中我们可以得到过载控制器。为了有效地识别过载控制器，并降低选择负载转移对的复杂性，我们使用 OM_C 和 IM_C 这两个集合来表示过载控制器集和低负载控制器集，我们用阈值检查控制器的响应时间，并确定哪些控制器可以被选为输出迁移控制器的集合 OM_C，哪些控制器可以被选为迁移控制器的集合 IM_C。如果响应时间满足 $t_{C_iT_n} > t_{threshold}$，控制器 Ci 就添加到 OM_C 中，如果 $t_{C_iT_n} < t_{threshold}$，控制器 Ci 就被添加到 IM_C 中。在时间间隔 Tn 内，我们通过得到所有控制器的平均响应时间集 A 来计算集合 OM_C 和 IM_C。在本小节中，我们决定是否需要通过生成两个控制器集来实现负载均衡。我们将负载均衡检测和触发的算法过程描述为算法 1。

算法流程图如图 4.3 所示。

4.4 交换机迁移方案

为了更好地提高负载均衡的效率，我们提出了一种基于算法 1 的交换机迁移方法，当多个过载控制器同时存在时，在一次负载均衡检测中实现多个交换机迁移操作，可以很好地实现多个控制器的负载转移，大大节省了负载均衡时间。如果激活了 4.3 部分中的负载均衡操作，我们可以得到 OM_C 和 IM_C 控制器集。下面的步骤可以详细说明交换机迁移集合 P 的决策，P 是由三元组<Cu、Se、Cv>表示的一

系列迁移操作组成的。首先，如果 OM_C 和 IM_C 都不为空，即一定有一个需要均衡的过载控制器和一个可以接受负载转移的低负载控制器，在这种情况下，需要使用交换机迁移来实现负载均衡。然后，我们需要执行以下步骤，为每个过载控制器生成迁移操作。

第一步：从 OM_C 集合中选择第一个工作负载最大的控制器 C_u,然后，选择对过载控制器影响最大的交换机 S_e 进行迁移。

第二步：在 IM_C 集合中，所有控制器的响应时间基本接近，不能根据响应时间值来判断选择哪一个控制器作为接收。因此，我们直接选择第一个工作负载最小的控制器 C_v 来接收负载迁移。此时，将三元组<C_u、S_e、C_v>添加至交换机迁移集合 P 中。然后，我们从 OM_C 中删除 C_u，从 IM_C 中删除 C_v。

第三步：接下来，我们可以得到三个更新的集合 OM_C、IM_C 与 P。然后重复第一步和第二步，直到 OM_C 和 IM_C 中有一个是空集。

算法流程图如图 4.4 所示。

最后，通过改变迁移交换机的管理角色，获得交换机迁移动作并完成负载迁移。通过算法 2 生成的交换机迁移集，我们对多个过载控制器进行了迁移决策。负载均衡的实际实现是由迁移执行模块完成的。通过完成这个实现过程，我们的方案可以真正实现多控制器的负载均衡操作。

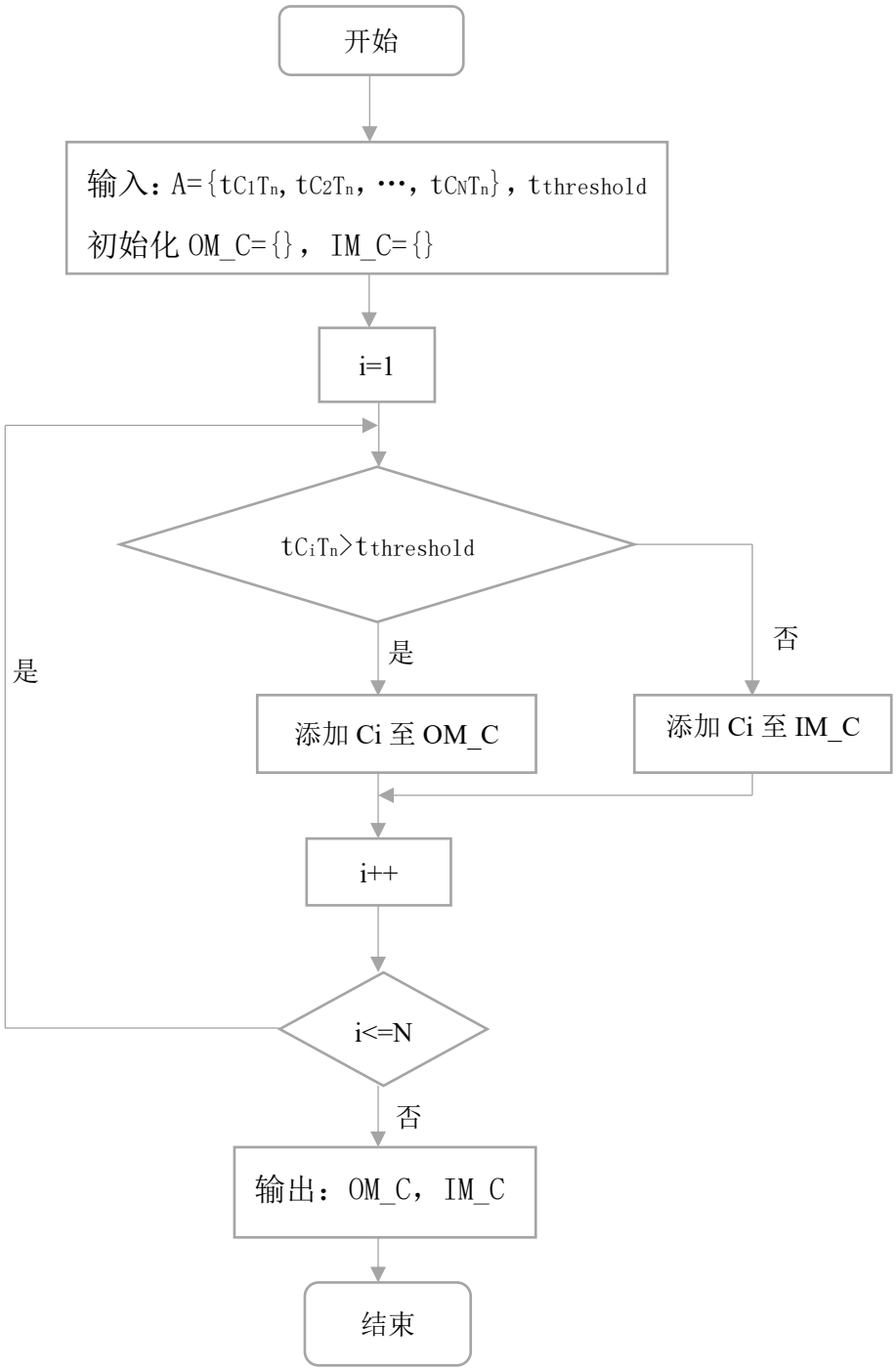
开始
输入：A={tC1Tn, tC2Tn，…，tCNTn}，tthreshold
初始化 OM_C={}， IM_C={}
i=1
tCiTn>tthreshold
是
否
是
添加 Ci 至 OM_C
添加 Ci 至 IM_C
i++
i<=N
否
输出：OM_C，IM_C
结束

图 4.3 算法 1 流程图

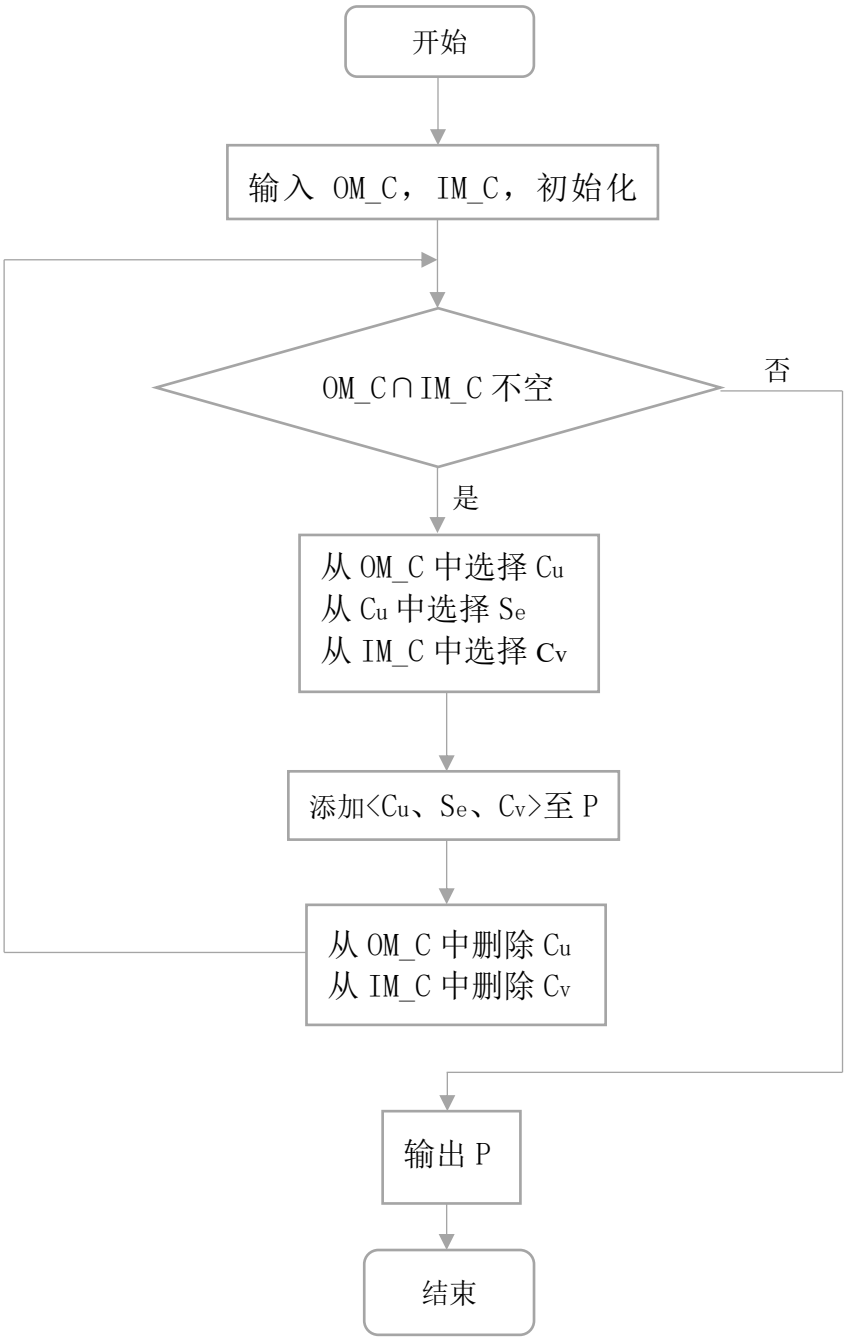

图 4.4 算法 2 流程图

4.5 本章小结

本章着重介绍了我们提出的基于响应时间的控制器负载均衡算法。最开始介绍了我们要在何种系统模型中分析并提出算法，然后我们对控制器的响应时间做出了清晰的描述。然后我们对 Floodlight 控制器的不同包入率下的响应时间进行了记录与分析，并绘制了曲线图，然后在该曲线图上得到了合适的响应时间阈值。基于此时间阈值，我们设计了基于响应时间的负载均衡算法，并针对多过载控制器

下的交换机迁移设计了一种可以在一次负载均衡操作中实现多个控制器负载迁移的算法。这两个算法也是本文的研究重点。

第五章 方案分析与比较

在本章我们将先对已有的基于预测的分散调度算法进行介绍，然后对我们所提出的基于响应时间的控制器负载均衡算法与已提出的基于预测的分散调度算法进行分析与比较。

5.1 基于预测的分散调度算法

该算法是北京邮电大学的蔡磊在他的基于 SDN 的分布式异构网络资源配置管理的研究论文中提出的，关于控制器负载均衡这方面，结合已有的算法，他提出了一种基于预测的分散调度算法，该算法主要分为两部分：

（1）负载均衡预测

这部分所说的控制器做出预测，并不是代表着下一刻控制器就会进入过载状态，而是在下一刻控制器进入过载状态后，控制器可以立刻进行负载均衡计算与负载任务分配，控制器通过广播的形式把负载均衡的消息发送到网络中，收集网络其他控制器负载信息，需要一定的数据传输时间，因此为了提高控制器负载均衡的运算效率，避免网络传输时延带来的性能的降低，需要提前做出过载预测[18]。如果收集了网络所有控制器的当前负载信息，控制器并没有出现过载情况，为了保证数据的实时性，收集到的负载值存活周期不易过长，应该符合网络对时延的基本要求，这种情况下，控制器只收集了其他控制器的负载信息，并没有进行负载值的计算，所以只消耗了一定网络带宽和控制器的少许存储空间，相比于控制器之间链路带宽资源，控制器的计算资源显得尤其重要，控制器计算资源决定了网络请求任务处理效率，所以本算法舍弃一定的链路资源，来换取控制器的计算资源[18]。

（2）负载任务调度

该部分主要是应用了注水算法，通过不断的迭代操作，分散任务，最终使整个控制器集群达到负载均衡。

整个算法的步骤分为：

a. 初始化负载表。

b. 计算各个控制器的负载均衡值，判断控制器是否过载。

c. 将过载控制器的任务分配给负载值最小的控制器，当负载值最小的控制器任务与第二小的控制器负载相同时停止分配。下次分配是对这两个控制器同时进

行任务的分配。

d. 重复步骤 c, 直至待分配的任务队列为空。

算法流程图如图 5.1 所示。

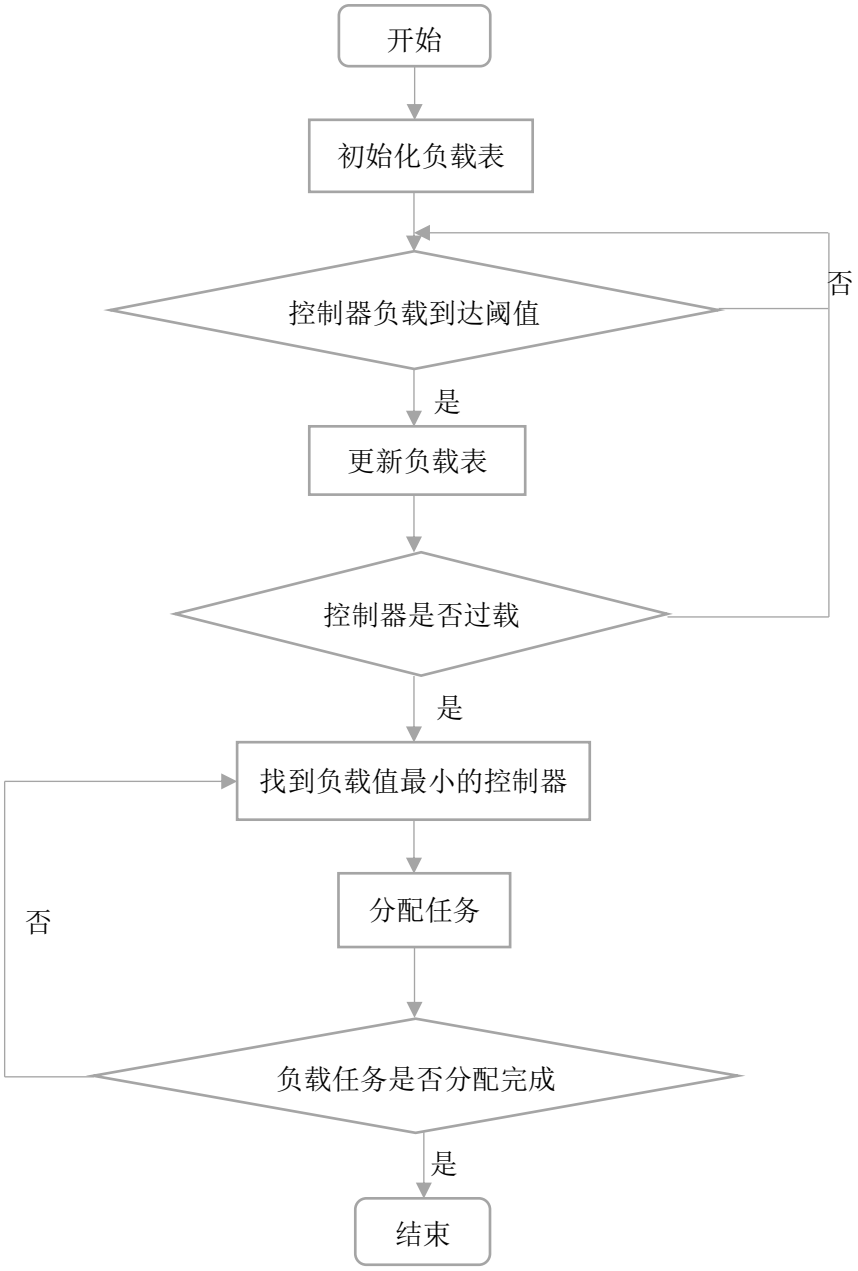

图 5.1 基于预测的分散调度算法流程图

5.2 算法分析与比较

本节将对我们所提出的基于响应时间的控制器负载均衡算法与上一节所介绍的基于预测的分散调度算法进行对比分析，分析这两个算法的优劣性。

5.2.1 算法的相似点

基于预测的分散调度算法与我们提出的基于响应时间的控制器负载均衡算法有一些相似点。

首先，这两种算法都可用于分布式 SDN 控制器结构。而分布式 SDN 控制器结构的特点就是没有中心控制器，控制器之间不分主从，这意味着每个控制器在网络中的地位是平等的。因为没有中心控制器，所以每台控制器上都要部署负载均衡算法。所以当这两种算法进行负载均衡的时候，都是主动发起负载均衡的控制器来进行负载均衡的计算，被动控制器只需要将自身的参数信息发给主动控制器，主动控制器将根据收到的参数信息进行负载均衡计算，然后做出合理的调度。因为每个控制器只负责自己的负载均衡，所以这样做的好处是不会出现相互干扰的问题，但是这样要求我们的控制器有较高的处理性能。

其次，这两种算法都带有一定的预测性质。基于预测的分散调度算法无需多述，在 4.1 节的第一部分有着详细的介绍，该算法的预测是指为了避免网络传输时延带来的性能的降低，需要提前做出过载预测，然后当控制器在进入过载后能够快速地进行反应，进行任务的调度。而我们的基于响应时间的控制器负载均衡算法的预测性质是指控制器的响应延迟在超过合适的时间阈值之后，控制器的响应延迟增长速率会极大地增加，但是这并不是真正意义上的控制器过载，这只是代表着由于响应延迟的快速增加，下一时刻控制器很可能进入真正意义上的过载，而在这时我们进行负载均衡，这样的话能够一直保持我们的控制器具有极高的处理性能，我们的算法的负载均衡地来临时间将比很多控制器负载均衡算法要提前。

但是它们的不同点更多，下面我们将对这两个算法的不一样的地方进行比对，并分析它们的优劣点。

5.2.2 算法的不同点以及优缺点

（1）本文所提出的算法的主要思想与大多数控制器负载均衡技术一样，都是将过载控制器下的交换机迁移到轻负载的控制器下。而基于预测的分散调度算法是不进行交换机的迁移，只进行过载任务的的迁移。交换机的迁移会导致网络结构发生改变，但是不会增加控制器链路之间的负荷，而只进行过载任务的迁移虽然不会改变网络结构，但会增加控制器链路之间传输的时间。因为我们没有仿真结果来验证自己的算法，所以我们对基于预测的分散调度算法的论文中的他的仿真结果

进行了分析，我们发现他的算法的仿真结果和他的算法理论所展现的效果并不相同，所以我们并不能够分析出哪种迁移方式具有更好的负载均衡速率。

（2）我们的算法与基于预测的分散调度算法最大的不同就是我们的算法是基于响应时间来实现负载均衡，而基于预测的分散调度算法是基于控制器的负载任务队列中的任务数量来实现负载均衡。基于预测的分散调度算法是基于控制器的任务队列中的任务数量来实现负载均衡，如果控制器任务队列中的任务超过阈值时，将启动负载均衡策略，在众多的控制器负载均衡算法中，就第二章第四节中的三个算法来说，这是它们的共识。而我们的算法是基于合适的响应时间阈值，完全从另一个角度出发，在此之前，虽然响应时间随着负载的增加而增加，但是它的变化缓慢而轻微，也就是说，控制器负载的增加对用户消息请求的影响很小。但是在此之后，响应时间将会增长得更快，如果控制器的负载持续增加，响应时间将显著增加。因此，如果控制器的响应时间达到这个阈值，我们需要快速处理该控制器，以防止其响应时间快速增长。

（3）这两种算法的负载均衡过程中分配任务的方式不同。我们的基于响应时间的负载均衡算法为了更好地提高负载均衡的效率，我们提出了一种新的交换机迁移方法，我们设立了一个交换机迁移集合 P，当多个过载控制器同时存在时，将它们需要迁移的交换机加入至交换机迁移集合 P，然后在一次负载均衡检测中实现多个交换机迁移操作，这样就可以很好地实现多个控制器的负载转移，大大节省了负载均衡时间。而基于预测的分散调度算法是对单个过载控制器的负载任务进行多次迭代分配，如果出现多个控制器出现过载，针对这种情况，它采用了一种负载表定时失效策略，用来解决负载均衡请求过多导致负载表一直不更新的问题，设定了负载表失效周期后，当负载表失效后，后续的负载均衡将进入等待队列，直至负载均衡完成。

5.3 本章小结

在本章中我们首先对基于预测的分散调度算法作了介绍，然后把我们的基于响应时间的控制器负载均衡算法与该算法在不同方面进行了对比，对它们的优缺点进行了分析。

第六章 总结与展望

本章主要是总结本文的主要内容，对工作任务进行简要的总结。并在此基础上，基于本文工作，对未来的工作做出展望。

6.1 工作总结

本文开始介绍了软件定义网络的研究现状及其相关技术背景知识。然后研究了多过载控制器问题，接着研究了响应时间随控制器负载的变化特征，选择变化曲线上的极值点作为合适的响应时间阈值，在负载均衡速率与交换机迁移成本之间进行权衡，提出了一种基于响应时间的交换机迁移负载均衡策略。该策略对控制器工作负载进行细粒度的判断，以便及时找到响应时间增长更快的控制器。因此我们的方案为过载控制器提供了一个很好的负载均衡点，使其能够提前转移负载。为了快速均衡多个过载控制器，我们还设计了一个交换机选择算法来同时执行不同的交换机迁移操作。

第二章主要介绍了 SDN 控制器负载均衡的相关技术工作，不仅介绍了 SDN 的架构，还对 SDN 中的 OpenFlow 协议、Floodlight 控制器以及已有的 SDN 控制器负载均衡算法进行了介绍，还介绍了实验所需的环境以及一些工具。第三章主要分析的是响应时间随控制器负载变化的变化曲线，基于该曲线，我们选择合适的响应时间阈值，并基于该合适的响应时间阈值提出基于响应时间的负载均衡算法，对我们提出算法的主要结构进行了阐述。

然后对我们提出的算法与已有的控制器负载均衡算法进行了理论上的分析与比较，分析已有的算法的主要思想方针，然后和我们的算法进行对比，比较算法的优劣，提出进一步改进的地方。

6.2 未来工作与展望

在这篇论文中，我们也有一些需要改进的问题，并在未来做更多的研究。最重要的问题就是我们没有真正的仿真结果来支撑我们的理论，所以前面的所有分析都是基于理论的。所以在接下来的工作中，我们需要做更多的仿真来验证我们的算法的可行性，我们将模拟真实的网络拓扑结构，将本文所设计的算法应用到仿真中，从仿真结果来验证本文所提出的方案的有效性。并在未来的工作中提出实时、

自适应地选择合适的响应时间阈值。我们还计划考虑迁移成本，研究一种更好地均衡多个过载控制器的方法。此外，我们计划考虑一些导致控制器工作负载快速异常增长的因素，如 SDN 控制器上的 DDoS 攻击。总之接下来的工作是复杂且繁重的，但是，我们仍然对之后的工作充满了期望。

致谢

四年的大学生活即将结束，这一刻我感慨万千，非常不舍，这四年的经历将是我人生中一笔宝贵的财富。

首先我要感谢我的导师尚老师。因为本文是在尚老师的精心指导下完成的，从论文的选题到最终的成果，他给予了我极大的帮助与指导，并给我提出了非常宝贵的意见，尚老师的专业水平、敬业精神、以及谦和的态度深深的影响着我，对我的学习以及将来的工作有着不可磨灭的影响。在毕设过程中，学长学姐也给出了非常多的宝贵意见，在不懂的问题上，细心地指导着我，对我的论文的最终完成是不可或缺的。在论文的结尾，我以诚挚的心情向尚老师与学长学姐表示衷心的感谢，感谢他们这半年时间里对我的亲切关怀、热情帮助与悉心指导。

同时我要感谢班级的老师和同学给予的关心和帮助。

最后，感谢我的家人给予我极大的支持与理解。

参考文献

[1] A.Dixit,F.Hao,S.Mukherjee,T.V.Lakshman,and R.Kompella,Towards an elastic distributed SDN controller,ACM,2013, pp.7-12

[2] 周华英，一种基于分布式决策的 SDN 控制器负载均衡策略，IEEE，2014 年，Page(s):851-856

[3] M.Koerner and O.Kao,Multiple service load-balancing with OpenFlow,IEEE, Jun. 2012, pp.210-214

[4] T.Wood,K.K.Ramakrishnan,J.Hwang,G.Liu,and W.Zhang,Toward a software-based network:Integrating software defined networking and network function virtualization,IEEE,May/Jun.2015,pp.36-41

[5] 李军，任永茂，一种有效的基于 SDN 的路径负载均衡机制，电子工程师学会学报,2014 年 9 月，Page(s):527-533

[6] R.Trestian,K.Katrinis,and G.-M.Muntean,OFLoad:An OpenFlowbased dynamic load balancing strategy for datacenter networks,IEEE, Dec.2017, pp.792-803,

[7] 钟红，崔杰，LBBSRT：一种有效的基于服务器响应时间的 SDN 负载均衡方案，未来世代，2017 年 3 月，Page(s):183-190

[8] T.Koponen et al.,Onix:A distributed control platform for large-scaleproduction networks,OSDI,2010,pp.351 – 364

[9] D.Erickson,The beacon OpenFlow controller,HotSDN,2013,pp.13 – 18

[10]S.H.Yeganeh and Y.Ganjali,Kandoo:A framework for efficient and scalable offloading of control applications,HotSDN,2012,pp.19 – 24

[11]Y.Fu et al.,基于流量的大规模软件定义网络的混合层次控制平面,IEEE,2015 年 6 月,Page(s):117 – 131

[12]B.Pfaff et al.,OpenFlow Switch Specification 1.3.0, 2012

[13]Liang et al., 基于多个开放流控制器的交换机迁移的可扩展和容错负载均衡,CANDAR,2014 年 12 月，Page(s):171-177

[14]Cheng et al.,面向 SDN 可扩展分布式控制平面的交换机迁移的决策,IFIP Netw,2015 年 5 月，Page(s):1-9

[15]Yu et al.,一种基于负载通知策略的多 SDN 控制器负载均衡机制,APNOMS,2016 年 10 月，Page(s):1-4

[16]Wang et al.,一种基于交换机迁移的 SDN 负载均衡决策方案，IEEE，2017，Page(s):4537-4544

[17]Zhou et al.,基于交换机组的 SDN 多控制器负载均衡，APNOMS，2017 年 9 月，Page(s):227-230

[18]蔡磊，基于 SDN 的分布式异构网络资源配置管理的研究，北京邮电大学，2017 年 12 月 1 日，Page(s):21-47

[19]黄韬，软件定义网络核心原理与应用实践，北京：人民邮电出版社，2014 年 9 月

[20]谭振建，毛其林，SDN 技术及应用，西安电子科技大学出版社，2017 年 9 月

[21]刘必果，基于交换机迁移的 SDN 控制平面负载均衡研究，安徽大学，2017

[22]黄韬，刘江，魏亮等，软件定义网络核心原理与应用实践 上 第 2 版，北京：人民邮电出版社，2016 年 9 月

[23]黄韬，刘江，魏亮等，软件定义网络核心原理与应用实践 下 第 2 版，北京：人民邮电出版社，2016 年 9 月

[24]Yang Xu,Marco Cello,I-Chih Wang et al., Dynamic Switch Migration in Distributed Software-Defined Networks to Achieve Controller Load Balance,IEEE,Mar.2019,pp.2-3