**班 级 1501013**

**学 号 15010130034**

****

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**

第一章 绪论

1.1 课题背景

当前正在运行的互联网体系架构已经有超过40年的历史，随着网络规模的急剧膨胀和业务类型的不断丰富，互联网的结构和功能也日趋复杂，网络管控难度日渐增加，网络新功能难以快速部署。这促使人们重新思考网络体系架构的设计，而此时SDN的提出为未来网络的发展提供了新的可行方向。

SDN之所以是一种革新性的技术，是因为它打破了传统网络架构的设计理念，一方面实现了控制平面与数据平面相分离；另一方面开放了网络可编程能力，从而提高了网络的灵活性和可管控性。此外，SDN运营建立在开放软件的基础上，可以显著降低业务部署和维护成本。

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平衡，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平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三个方面进行了介绍。第一个方面介绍了传统的负载平衡方法和基于SDN的负载平衡方法，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。首先，应该将大量并发访问或网络数据包分配给多个节点设备，以减少用户响应时间。其次，应该将单个重载操作分配给多个节点设备进行并行处理，以显著提高负载平衡[3]的整体性能。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。硬件方式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。软件方式可以通过负载均衡提高链路带宽利用率，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分离，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和南向接口协议分别与上层应用和下层转发设备实现交互，根据不同的用户需求和全局网络拓扑结构，动态灵活的分配网络资源。对于数据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，有效地平衡了流量，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控制平面的安全性和性能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，在迁移成本和负载平衡率之间进行权衡。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平衡策略。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.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。

2.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。

3. 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协议、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。

第四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。

第五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，可以将SDN定义为一种数据控制分离、软件可编程的新型网络体系架构，其基本架构如图2.1所示。SDN采用了集中式的控制平面和分布式的转发平面，两个平面相互分离，控制平面利用控制—转发通信接口对转发平面上的网络设备进行集中式控制，并向上提供灵活的可编程能力，具备以上特点的网络架构都可以被认为是广义的SDN技术。

应用层

业务应用

API

API

API

网络服务

SDN控制器

控制层

控制—转发通信接口

网络设备

网络设备

网络设备

基础层

网络设备

网络设备

图2.1 SDN的基本架构

应用平面由若干SDN应用构成，SDN应用是用户关注的应用程序，它可以通过北向接口与SDN控制器进行交互，即这些应用能够通过可编程方式把需要请求的网络行为提交给控制器。一个SDN应用可以包含多个北向接口驱动（使用多种不同的北向API），同时SDN应用也可以对本身的功能进行抽象、封装来对外提供北向代理接口，封装后的接口就形成了更高级的北向接口。

SDN NBI是应用平面和控制平面之间的一系列接口，它主要负责提供抽象的网络视图，并使应用能直接控制网络的行为，其中包含从不同层次对网络及功能进行的抽象，这个接口也应该是一个开放的、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：一是将SDN应用层请求转换到SDN数据路径(Datapath)，二是为SDN应用提供底层网络的抽象模型（可以是状态，也可以是事件）。一个SDN控制器包含北向接口(Northbound Interfaces,NB)代理、SDN控制逻辑(Control Logic)以及控制数据平面接口驱动(Control Data Plane Interface Driver,CDPI Driver)3个部分。SDN控制器只要求逻辑上完整，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。从地理位置上来讲，既可以是所有控制器实例在同一位置，也可以是多个实例分散在不同位置。

SDN CDPI是控制平面和数据平面之间的接口，它提供的主要功能包括：对所有转发行为进行控制、设备性能查询、统计报告、事件通知等。SDN一个非常重要的价值就体现在CDPI实现上，它应该是一个开放的、与厂商无关的接口。

数据平面由若干网元(Network Element)构成，每个网元可以包含一个或多个SDN Datapath，是一个被管理的资源在逻辑上的抽象集合。每个SDN Datapath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 Datapath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。

2.2 OpenFlow技术概述

2.2.1 OpenFlow协议背景

SDN控制器对网络的控制主要通过南向接口协议实现，包括链路发现、拓扑管理、策略制定、表项下发等，其中链路发现和拓扑管理主要是利用南向接口的上行通道对底层交换设备上报的信息进行统一监控和统计，而策略制定和表项下发则是控制器利用南向接口的下行通道对网络设备进行统一控制。

最知名的南向接口莫过于ONF(Open Network Foundation,开放网络基金会)倡导的OpenFlow协议。作为一个开放的协议，OpenFlow突破了传统网络设备厂商

对设备能力接口的壁垒，经过多年的发展，在业界的共同努力下，已经日臻完善，能够全面解决SDN中面临的问题。OpenFlow在SDN领域中的重要地位不言而喻，甚至大家一度产生过OpenFlow就等同于SDN的误解，实际上，OpenFlow只是基于开放协议的SDN中可使用的南向接口之一。

2009年发布了OpenFlow协议的第一个正式版本1.0.OpenFlow1.0版本的优势是它可以与现有的商业交换机芯片兼容，通过在传统交换机上升级固件就可以支持OpenFlow1.0版本，因此，OpenFlow1.0是目前使用和支持最广泛的协议版本。2011年2月，OpenFlow.org发布了OpenFlow1.1版本。2011年3月，德国电信、Facebook、谷歌、微软、NTT、Verizon和Yahoo联合成立了ONF，同年3月，ONF接管了标准的后续开发和维护。ONF成立之后，OpenFlow的发展明显加快。2011年，ONF批准了OpenFlow1.2版，并于2012年2月正式发布。ONF目前已经推出了两个可商用化的版本：OpenFlow1.0和OpenFlow1.3。2012年4月发布了OpenFlow1.3版本，成为长期支持的稳定版本。1.3版中增加了Meter表，用于控制关联流表的数据包的传送速率，但控制方式目前还相对简单。2013年最新发布的OpenFlow1.4版本仍然是基于1.3的改进版本，数据转发层面没有太大变化，主要是增加了一种流表同步机制。随着ONF组织的成立，OpenFlow协议得到了更快的发展，目前版本已经发展到1.5版。

2.2.2 OpenFlow协议架构

OpenFlow是一个开放的协议，它定义了控制器如何对控制平面进行配置和管理。通过使用OpenFlow，控制器可以管理数据包在网络中的传输。在传统网络中，交换机和路由器的信息以不同的形式（路由表、MAC表等）进行存储，需要在整个网络或一组网络中通过复杂的交换和路由协议进行计算才能得到。OpenFlow规范并集中了协议，通过创建和管理流信息表来代替其他所有的转发表，数据平面就是根据这些流信息表生成的。

OpenFlow架构的主要组件有OpenFlow控制器、OpenFlow交换机和OpenFlow协议，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。

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是一款基于Java语言的开源SDN控制器，遵循Apache2.0软件许可，支持OpenFlow协议。Floodlight的日常开发与维护工作主要由开源社区支持，其中包括很多来自Big Switch Networks公司的工程师，同时Floodlight也是该公司商业控制器产品的核心部分。Floodlight作为免费的开源控制器提供了与商业版本相同的API，使得开发者可以把Floodlight上的程序快速移植到商用版本的控制器上。

Floodlight与NOX、POX等其他控制器类似，也使用了层次化架构来实现控制器的功能，同时提供了非常丰富的应用，可以直接在网络中部署数据转发、拓扑发现等基本功能。此外Floodlight还提供了友好的前端Web管理界面，用户可以通过管理界面查看连接的交换机信息、主机信息以及实时网络拓扑信息。Floodlight通过向OpenFlow交换机下发流表等方式来实现数据分组转发决策，以达到交换设备集中控制的目的。

2.3.2 Floodli ght控制器架构

基于REST API应用程序从模块化应用、传统JAVA API和Floodlight控制器模块获取API

APP

APP

REST API

Link

Module

Firewall

JAVA API

Devicemanager

Threadpool

VNF

Unit test

Packet

Forwarding

Storage

Web UI

Learning Switch

Hub

图2.2 Floodlight模块架构

Floodlight整体架构由控制器核心功能以及运行其上的应用组成，应用和控制器之间可以通过Java接口或REST API(Representational State Transfer API,表征状态转移API)交互，控制架构如图2.2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2.1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2.1 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是一个可以在有限资源的普通电脑上快速建立大规模SDN原型系统的网络仿真工具。该系统由虚拟的终端节点(End-Host)、OpenFlow交换机、控制器组成，这使得它可以模拟真实网络，可以对各种想法或网络协议等进行开发验证。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四个方面：

* 灵活性。可通过软件的方式简单、迅速地创建一个用户自定义的网络拓扑，缩短开发测试周期，支持系统级的还原测试，且提供Python API，简单易用。
* 可移植性。Mininet支持OpenFlow、OVS等软件定义网络部件，在Mininet上可运行的代码可以轻松移植到支持OpenFlow的硬件设备上。
* 可扩展性。在一台电脑上模拟的网络规模可以轻松扩展到成百上千个节点。
* 真实性。模拟真实网络环境，运行实际的网络协议栈，实施管理和配置网络，可以运行真实的程序，在Linux上运行的程序基本上都可以在Mininet上运行，如Wireshark。

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。

Mininet可以快速地创建SDN拓扑，但是Mininet本身只提供了几种简单的拓扑类型，如需要更复杂的网络拓扑，通常需要用户通过编写Python脚本的方式来手动创建。VND(Visual Network Description,虚拟网络描述)是一个基于网页的通用网络场景图形化设计工具，它能够生成NSDL(Network Scenario Description Language，网络场景描述语言)脚本，并可以进一步地针对这些场景进行仿真和分析。该工具通过友好的图形化界面能够设置SDN的流表、Qos等信息，并可以在创建NSDL脚本的同时自动生成Mininet和OpenFlow控制器脚本。

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 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:控制器负载不平衡的测量;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3.1，我们建立一个由N个控制器C={C1，C2，…,CN}和K个交换机S={S1，S2，…,SK}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QCi∈S表示由主控制器Ci管理的交换机集。

…

CN

C1

C2

SK-1

SK-2

S7

S6

S2

S1

SK

S9

S8

S5

S4

S3

图3.1 系统模型

3.1 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Sk和Tn分别表示第k个交换机和第n个周期，tarrive表示PACKET\_IN消息到达的时间，treply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(1)所示。

(1)

以fSkTn表示Sk在Tn中的请求消息数量，并将其表示为Sk带来的工作负载LoadSkTn，因此，可以用集合QCi中这些交换机的和负载fCiTn来表示第n个周期内接收到的请求消息的总数或控制器Ci的工作负载LoadCiTn，如式(2)所示。

(2)

以tSk\_response表示Sk在Tn内发送的PACKET\_IN请求的响应时间，因此，可以得到Ci在Tn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tCiTn，如式(3)所示。

(3)

3.2 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，很难实现对负载不平衡的最佳检测。如果阈值过小，由于流量的暂态性，容易发生负载不平衡，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平衡率。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，对负载不平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3.1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3.2所示。



图3.2 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tthreshold。设g(ti-1)为第(i-1)个周期的平均响应时间。因此，g(ti)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)和(5)。

(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tthreshold。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平衡速率和平衡成本之间的权衡。

3.3 检测和触发负载均衡

在3.2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={tC1Tn,tC2Tn，…，tCNTn}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tCiTn>tthreshold，控制器Ci就添加到OM\_C中，如果tCiTn<tthreshold，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平衡。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3.3所示。

3.4 交换机迁移

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3.3部分中的负载平衡操作，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Cu,然后，选择对过载控制器影响最大的交换机Se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Cv来接收负载迁移。此时，将三元组<Cu、Se、Cv>添加至交换机迁移集合P中。然后，我们从OM\_C中删除Cu，从IM\_C中删除Cv。

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3.4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。负载平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平衡操作。

开始

输入：A={tC1Tn,tC2Tn，…，tCNTn}，tthreshold

初始化OM\_C={}，IM\_C={}

i=1

tCiTn>tthreshold

否

是

是

Add Ci to IM\_C

Add Ci to OM\_C

i++

i<=N

否

输出：OM\_C，IM\_C

结束

图3.3 算法1流程图

开始

输入OM\_C，IM\_C，初始化P={}

OM\_C∩IM\_C不空

否

是

从OM\_C中选择Cu

从Cu中选择Se

从IM\_C中选择Cv

添加<Cu、Se、Cv>至P

从OM\_C中删除Cu

从IM\_C中删除Cv

输出P

结束

图3.4 算法2流程图

3.5 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。

第四章 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。

4.1 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4.1所示。

开始

初始化负载表

否

控制器负载到达阈值

是

更新负载表

控制器是否过载

是

找到负载值最小的控制器

否

分配任务

负载任务是否分配完成

是

结束

图4.1 基于预测的分散调度算法流程图

4.2 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。

4.2.1 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。

4.2.2 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。

4.3 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。

第五章 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。

5.1 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，在负载平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。为了快速平衡多个过载控制器，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。

5.2 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，研究一种更好地平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韬教授。因为本文是在尚韬教授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，尚韬教授的专业水平、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平衡,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
19. 黄韬，刘江，魏亮，软件定义网络核心原理与应用实践，人民邮电出版社，2016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