中国科学技术大学

专业硕士学位论文

（专业学位类型）

软件定义网络中 QoS 可感知的流量管理策略研究和实现

作者姓名：　雷　经

专业领域：　软件工程

校内导师：　黄刘生教授

企业导师：　王宜怀教授

完成时间：　二〇一八年四月一日

University of Science and Technology of China

A dissertation for master's degree

（Professional degree type）

A Qos-aware Traffic Management Scheme in Software Defined Network

Author：	Lei Jing
Speciality：	Software Engineering
Supervisor：	Prof. Huang Liusheng
Advisor：	Prof. Wang Yihuai
Finished time:	April 1st, 2018

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____ 签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开 □保密（____年）

作者签名：_____ 签字日期：_____

摘 要

在云计算和大数据的广泛应用下，各类企业和机构的网络规模不断扩大，网络流量急剧增长，这对网络性能提出了越来越高的要求，网络的管理面临着巨大的挑战。目前云数据中心中依托虚拟化技术已经实现了对物理资源（包括CPU，内存和存储等资源）的灵活调度，极大地提高了资源的利用率。但是网络资源的调配仍然受到很大的限制，原因是网络对设备管控能力不足并且网络管理成本过高。传统网络往往网络缺乏对上层应用的感知，在应用业务出现变动时网络无法及时进行应对，难以保障应用业务动态变化的服务质量需求。下一代网络要求网络具备灵活地根据实际需求资源调配的能力，但传统网络的架构下难以提供较为有效的解决方案。

软件定义网络（SDN）作为一种新型网络，具有集中控制和可编程的特点，该种网络架构将控制平面和数据平面分离，使网络管理功能抽离，将网络设备的管理方式进行统一，网络能从全局视角制定资源调配方案。SDN 中网络管理变得更加智能，通过网络可编程的接口可以实现自定义功能，从而实现弹性化、细粒度的网络管理，满足不同应用业务的需求。

目前，现有大多数的 SDN 流量调度策略还不能将调度的粒度精确到特定的应用数据流上，网络不支持应用程序主动申请执行 QoS 策略。因此，本文 SDN 中 QoS 感知的流量调度问题进行研究，并实现了一种 QoS 感知的流量调度策略：通过开发控制器北向接口增加网络对上层业务的感知，使得网络调度的行为可以随着上层业务的需求进行调整，并且使对服务质量具有较高要求的数据流能按照 QoS 指标更好的路径进行转发，同时其他数据流能沿着负载较低的路径进行转发。

文中基于 SDN 开源控制器 Floodlight 进行开发，对上述方案效果进行了测试。实验结果显示，文中控制器能对 QoS 流进行灵活的配置，在网络运行良好且不出现明显拥塞的情况下，控制器为延迟，丢包或带宽敏感的数据流分配相应 QoS 指标更佳的转发路径，路径分配的准确率能达到 90%，QoS 流的传输质量得到了明显的提升。对于普通流的调度实验显示，相对于控制器原有的单路径转发方式，文中调度策略能充分利用可用转发路径，使网络主机间的吞吐量能提高超过 1 倍。

关键词：软件定义网络　QoS　OpenFlow　流量管理　Floodlight

ABSTRACT

With the widely adoption of cloud computing and big data, the scale of the network of all kinds of enterprises and institutions is expanding, and the traffic of these networks is increasing rapidly. This poses higher and higher requirements for network performance and brings huge challenges to network management. Currently, , cloud data centers have realized flexible scheduling of physical resources (including CPU, memory, storage, and other resources) by using virtualization technology, which greatly increase the utilization of resources. But due to the complexity and the high costs of network management, the distribution of network resources still lack of flexibility and effectiveness. Traditional networks often lack the awareness of network applications and services, thus network is hard to respond to the application changes and ensure the dynamic quality of service (QoS) requirements of different application services. Next generation network requires the ability to dynamic allocate network resources based on actual traffic demand. However, traditional networks cannot provide effective solutions.

Software Defined Network (SDN) has a brand new framework, which allowing centralized control and programmability. SDN decouples control plane and data plane, standardize deivce management, and provides global view of network, making network management more intelligent. The programmable northbound interface of SDN enable developers to implement custom functions, which and enable flexible and fine-grained traffic engineering, and can better meet the needs of different applications.

At present, most of the existing SDN-based network do not support applications to requrest QoS services, and network scheduling does not provide fine enough control over flows of a specific application. Therefore, this paper study the QoS-aware traffic scheduling problem in SDN, and implemented a QoS-aware traffic scheduling scheme. The scheme developing the northbound interface of the network controller and adding the application awareness to the network, which making the behavior of the network scheduling can be adjusted with the needs of applications. The scheme can offer better paths for QoS flows, and scheduling non-QoS flows to low-load paths.

The scheme is based on Floodlight controller, and the performance of the

controller is tested. The test cases shows that the controller can configure QoS flow flexibly, and can offer better paths for delay-sensitive flow, loss-sensitive flow and bandwidth-sensitive flow. The accuracy of path allocation can reach 90%, and the transmission quality of the QoS flow has been significantly improved. Moreover, compared to the single path forwarding method of the original controller, the controller can take advantage of available forwarding paths, increasing the throughput of non QoS-flows between hosts by more than 1 times.

Key Words: SDN，QoS，OpenFlow，Traffic Management，Floodlight

目 录

第 1 章 绪论

1.1 软件定义网络的研究背景与意义

计算机网络自诞生以来就对社会的发展起了十分巨大的推动作用，经过多年的发展网络技术已经深入到人们生活的方方面面。网络规模的增大和流量的增长给网络的管理提出了更高的要求。网络管理过程中应更有效地利用网络资源，以降低成本并提高系统的总体性能[1]。

传统网络由路由器，交换机，终端等多种设备组成，这些设备通常配备着由供应商独立定制的专有封闭接口。网络面临着设备繁多，配置复杂，管理成本高的特点，因而企业需要花费大量的财力来维护来网络。有调查显示，在 IT 企业的投入中，仅有 20%的投入用于软硬件更新与商业价值的提升，而 80%则投入与系统维护[2]，这很大程度上限制了网络的创新和发展。于此同时，传统网络大量分布式的协议使得网络设备缺乏全局视角，不能很好地调度网络资源和保障数据流的服务质量。

如何提供优质的网络服务质量（Quality of Service，QoS）一直是现今网络的主要探讨的议题之一[3]，QoS 参数作为衡量网络服务质的重要指标，在许多场景下发挥着重要的作用。一个典型的场景是云计算，云架构下服务器的数据处理能力强大，计算和存储资源往往不会成为系统的瓶颈，但网络传输的效率缺却是影响系统效率的重要因素。QoS 参数作为系统选择业务服务器的一项重要依据，扮演者重要的角色，云管理系统进行服务提供时需要根据链路状况进行服务器的选择，网络链接的情况会直接影响到系统的效率和服务的质量。如果将网络看作云系统下的一种服务，那么网络的 QoS 需求便可以视为系统的非功能性需求，网络的服务质量就和系统的性能息息相关。

传统网络架构下难以较好地实现 QoS 策略，原因是不同的应用业务对网络质量的需求各不相同，而网络设备运行的静态协议不能很好地处识别的流量需求。例如，视频流媒体具有流量大和持续时间长的特点[4]，数流据需要长时间保持稳定的传输速率，以保证观看者能接收到清晰流畅的图像数据，传输过程中需要带宽保障。语音交流或游戏业务需要保障低传输过程中需要低时延和少丢包，否则即使数据传递到目的主机数据也会因超时成为无效数据。另外，应用的种类繁多，即使是同一类业务的 QoS 需求也会随着时间的改变而发生变化。例如，云系统下可能运行着着成千上万款软件，提供着种类多样的云服务，应用业务的种类更迭很快，网络往往无法对上层应用的变动做出及时的应对措施。在网络资源有限，特别是在网络出现瓶颈时，就尤其需要进一步甄别来自不同应用业务的

流量，优先处理核心业务，放缓非关键业务的处理，以保障系统主要部件的正常运作，避免系统整体失效。因而，应用业务网络服务需求的动态性和网络管理的滞后性使得当前网络服务质量保障的效果不佳。

面对这个问题的一个可行方案是，将应用业务信息引入到网络中，使网络能对应用业务的重要性和业务需求服务质量进行感知，进而根据应用的特点制定转发策略。换句话说，网络应当具备获取应用信息并根据应用 QoS 需求进行自适应路由，动态制定转发路径的能力。因而应用可感知的网络（Application-aware Network）[5][6]、应用驱动的网络（Application-driven Network）[7]和 QoS 可感知的网络（QoS Aware Network）[8]的思想不断涌现。传统观点下的网络感知是网络对网络底层流量状况进行监测和分析，包括流量监测和流量分析两部分，而新型网络要求网络不仅对网络底层设备进行服务，同时还要对网络上层应用的运行状态进行一定的感知，以兼顾不同流量对服务质量的差别，充分贴近应用业务的实际需求。而这些思想都需要依托软件定义网络技术才能得以实现。

软件定义网络（SDN）是由 Nick Mckeown 教授于 2009 年提出的一种新型网络[9]，提出后迅速的受到学术界和工业界的广泛关注，同年被麻省理工大学评为“最重要的十大新兴技术之一”[10]，SDN 相关技术在随后的几年内逐渐成为了研究的热点。SDN 具有集中控制和可编程的特点，控制层和数据层分离，将网络管理功能抽离，可以从网络全局视角制定资源调配方案，实现了南向接口的统一，使网络管理更加便捷。SDN 使网络管理更加智能，通过其可编程的接口进行自定义开发，可以使得网络行为随着上层应用业务的策略进行调整和改变，以实现更加弹性化、细粒度的网络管理。SDN 能很好地控制网络运营成本，化简网络管理，并支持软件编程，因而成为最具潜力的网络技术之一。

据悉，SDN 市场已于 2013 年达到约 2 亿美元的产值,预计到 2016 年将达到 20 亿美元[11],SDN 拥有广阔的市场,具有广阔的发展前景和巨大的研究价值。目前 SDN 已经从理论研究逐步走向实际生产，并应用到了高校校园网，数据中心网络，企业私有网络等多个领域中，随着 SDN 研究的不断深入，其应用领域也将更加广泛。

综上，网络流量不断增长，网络用户希望得到更好的服务质量，而传统网络下大量运行的分布式服务限制了网络的资源调度，难以实现高效的路由和流量调控。下一代网络要求网络服务从用户或应用的需求为出发点，进行资源的合理调配，实现网络资源使用的最大化，对的网络服务质量实现有力的保障。SDN 的诞生给网络技术带来了巨大的革新，SDN 集中控制的优势可以使网络流量调度更加智能，QoS 策略实施更加灵活，网络服务质量进一步提升。SDN 是下一代网络技术的代表，其相关技术具有十分重要的研究意义。

1.2 国内外研究现状

1.2.1 服务质量 QoS

服务质量（Quality of Service，QoS）用于描述网络在带宽，时延，时延抖动，丢包率，可用性等多方面的术语。根据服务的级别，可以将 QoS 的服务级别分成刚性（Hard-QoS）和弹性（Soft-QoS）两个类别。

刚性服务：该种服务严格按照 QoS 需求参数进行网络资源的分配，在网络资源低于需求时会拒绝执行对应的任务。集成服务（Intergrated Servive）模型与该种思想类似，该种网络服务需要为数据流传输路径预留资源，能实现很好地保障数据传输的质量。但该种服务模型，容易造成资源的浪费，实现机制相当复杂并且难以扩展，在真实网络中应用得并不多[12]。

弹性服务：弹性服务分配网络资源时没有刚性服务那么严格，在可用资源不足时可适当降低服务等级。差分服务（Differential Service）就是一种弹性 QoS 服务，该服务模型把相似需求的业务流量进行归类，按不同类别提供不同级别的服务。

MPLS 模型对数据包进行标记，根据标记对数据包进行转发，加速了数据的传输效率，增强了 IP 网络实施流量工程的能力[13]，但 MPLS 模型在传统网络的分布式协议的限制下，不能充分发挥其作用，难以从网络整体上实现资源的调度。

传统网络架构下已经具备了许多较为成熟的 QoS 技术，例如：优先队列技术可以处理链路拥塞时的报文调度和处理先后问题；流量整形和拥塞避免技术可以限制特定端口的速率以更好地利用带宽资源，避免带宽滥用；负载均衡技术可以提高链路利用率，增加网络吞吐量等等。

1.2.2 软件定义网的特点

软件定义网络是一种新型网络架构，SDN 的思想源于可编程网络（Programmable Network），并在可编程网络的基础上进行革新：将控制器加入到网络架构中，使用控制器统一对网络进行管理，其核心思想是控制平面和数据平面分离，在控制平面实现对网络设备的集中管理。SDN 的特点突出表现为以下四点[14]：

（1）控制平面和数据平面分离

SDN 架构中控制功能和数据转发功能相互分离，避免了传统网络中的路由控制功能与转发功能的强耦合问题，同时可以将底层网络设备和拓扑和上层应用进行隔离，使得网络开发人员和运维人员可以避开网络底层的复杂设备，专注于网络的管理，降低了网络的运营成本。

（2）逻辑上集中控制

SDN 控制器对网络进行集中控制，控制器可以监测网络拓扑和设备状态信息，具有全局调度网络资源的能力，可以对资源的分配进行全局优化。在 SDN 中资源调度的计算过程由控制器完成，然后将指令传达到数据层上的设备，转发设备只负责指令的执行。控制器功能集中，不需要再考虑设备之间的差异，使网络管理更加简单和高效。

（3）基于流的转发策略

SDN 使用基于流（Flow-based）的转发策略，而非传统的基于目的地址（Destination-based）的转发策略。基于目的地址的转发策略通在数据的转发过程中不能很好地考虑到链路的负载状况和数据的传输质量，而 SDN 基于流的转发方式可以实现更细粒度的控制，通过处理流水线和多级流表可以动态调整网络流量，流量管理变得更加灵活和高效；

（4）可编程的开放接口

SDN 控制器开放可编程的北向接口供应用程序调用，开发人员可以通过软件可以对网络行为进行控制，根据实际需求进行自定义的开发，这个特性为智能的网络管理铺平了道路，使大规模网络实验提供了可能，进一步推动网络的创新。

SDN 的架构具有开放性和创新性，使网络可以实现的自动化管理，进行灵活的流量调度，大幅提高网络管理的效率，具有十分强大的优势。

1.2.3 SDN 和 QoS

在网络中，大量的实时业务流对传输过程中的延迟、丢包敏感，所以合理地调度网络资源，为实时业务提供 QoS 保证是流量工程的一项重要工作[1]。SDN 提供开放的控制接口，可以充分兼顾不同应用的需求，网络开发者可以按需进行自定义开发。SDN 能对网络全局资源进行调配，能使用软件进行高效而灵活的流量调度，减少网络拥塞的发生，实现网络负载均衡。利用 SDN 集中的网络视图和动态的规则配置能力，可以对网络流量进行灵活的调度. 所如何结合流量特征、网络状态和应用需求进行流量的路由转发，实现网络负载的有效均衡，是基于 SDN 的数据流调度研究工作中的重点[1]。

如何根据流量的实际需求快速而有效地分配资源，是下一代网络需要解决的主要问题之一，但目前在实现流量管理可伸缩性的同时提供严格的 QoS 保障仍是一个极具挑战性的问题。目前常见的 DiffServ，Integrated，MPLS 模型都不支持单一应用申请特定级别的 QoS 服务[15]，而 SDN 的到来使这个问题的解决更近了一步而，SDN 基于流模式管理模式可以实现对端到端的应用流量灵活而细粒度的调控，目前已经有很多论文致力于基于 SDN 的 QoS 研究。

对业务流量进行分类，能更好实现不同的层次的服务质量保障，已经有文献显示，能通过一系列的方法有效区分各类应用的流量。文献[5]使用数据包首部

字段匹配和深度数据包检测（Deep Packet Inspection, DPI）的方法识别不同应用的流量，进而可以对不同的应用流量采用不同的转发策略。文献[8]提到传统流量分类方法需要对流量的确切种类进行识别，分类效率不高，因而文献中提出了 QoS 可感知的流量分类框架（QoS-aware Traffic Classification Framwork），框架将根据 QoS 需求将应用进行分类，使用深度数据包检测和半监督学习方法，能实现流量的精确分类和细粒度的流量调度。

流量分类已经有了一定的技术支撑，因而有部分文献已经开始研究面向应用业务的服务，文献[6]提出云数据中心下的应用可感知的路由（Application-aware routing Scheme，App-RS），文章将网络内的所有应用进行分析，根据链路负载、时延和时延抖动三个 QoS 参数将应用划分为三类，对不同类别的应用流量采用不同路由方式。

多媒体流量占据着因特网总流量很大的一部分，因而也有较多文献研究多媒体流的 QoS 保证。文献[16]提出 OpenQoS 框架，将多媒体流和普通数据流进行区分，使用动态 QoS 路由算法，使多媒体流使用时延或丢包表现更好的链路进行转发，普通数据流则按最短路径照进行转发。文献[17]提到流媒体需要稳定，地抖动和无丢包的传输，因而提出了一个服务流媒体的最优化 QoS 框架，实验显示框架可以多种环境下显著提升可伸缩编码视频流的传输质量，并且不会对其他数据流产生副作用。

还有部分文献基于 SDN 开发定制网络环境，服务于专项应用业务。文献[18]提到，分布式大数据计算对网络资源的感知能力的不足，会导致任务调度的不平衡，造成网络瓶颈，进而影响系统的计算速度。因此文献中提出了程序可感知的网络（application-aware networking , AAN）以管理分布式 Hadoop 流量，该网络基于 SDN 进行开发，实现拓扑发现，流量监测，重路由，避免回路多种功能，使 Hadoop 的计算效率进一步提高。

从网络资源的分配上看，由于目前 SDN 框架并没有提供完善的 QoS 策略执行框架，文献[19]提出了名为 PolicyCop 的自动 QoS 应用框架，PolicyCop 框架提供执行 QoS 策略的接口，能监测 QoS 策略的执行情况，并可以根据数据层的实际情况自动进行策略的调整，能很好地避免传统基于服务等级协议（Service-Level Agreement，SLA）的 QoS 部署方式缺乏开放性和灵活性的问题。

SDN 可以实现对网络资源进行灵活的分配，文献[20][21]都通队列和计量表技术对带宽进行动态分配，能有效地提高链路的资源利用，实现多层次的 QoS 保障。

1.2.4 面向 QoS 的流量调度

流量工程中的主要工作是为将流量映射到网络物理拓扑上，流量工程的包含

着大量的工作，是一套系统化的有效管理网络的方法，流量工程的主要目标为优化流量的处理和资源的分配[1]。对流量进行的合理调度是提高网络性能的一个重要途径，一方面可以利用算法对路径进行合理的安排，充分利用网络资源（主要为带宽资源），提升网络的吞吐量。另一方面采用一系列方法保障业务流量的服务质量，降低传输时延，减少时延抖动和链路丢包，保障数据传输的连贯性和稳定性，从资源的使用和传输质量的保障两方面进行优化。文献[1]中对现有的流量调度方法进行分析总结，将数据层上的流量调度方式分类为：基于 ECMP 改进的流量调度，基于通配符的流量调度和面向 QoS 的流量调度。

基于 ECMP 改进的方法，通过对算法的改进避免了 ECMP 算法中具有相同 IP 数据流总是沿这同一路径进行转发导致的链路负载不均衡问题，通过为 elephant 流安排适合的路径，进一步优化网络的吞吐量，提高链路的利用率，如文献[22]中 Heder 系统可以使链路利用率达到 96%，使网络吞吐量比静态方法提高 100%以上。

基于通配符的流量调度方式，通过通配符规则提高对 OpenFlow 流表的利用效率，能有效地减少控制器的介入，增强控制器对新数据流路径的处理能力。如文献[23]通过对 elephant 流的检测和重路由，能使流表项的使用减少 90%以上，使控制信息也减少超过 90%。

面向 QoS 的流量调度方式，在制定转发策略时考虑 QoS 需求，避免对所有流量一刀切的处理方式，可以很好地服务于网络中的对延迟和丢包敏感实时业务流，特别是在网络资源有限的情况下，能保障高优先级的流量的处理，实现网络资源的合理调度。SDN 在基于流的调度方式可以实现细粒度的调控，可以将流量的调度细化到每一条具体的数据上。与基于 IP， MPLS 的常规的流量工程方法相比，SDN 中可采用集中式的流量工程系统实现更加高效和智能的流量工程机制[24]，因此软件定义网络框架更加利于 QoS 策略的实现，能更好地保障数据流的服务质量。

1.3 本文主要研究内容和本人工作

1.3.1 本文主要研究内容

本文主要研究 QoS 可感知的流量管理策略，研究 SDN 架构和 QoS 技术，分析当前流量调度方案的问题和不足，探讨可行的优化方案。提出了一种 QoS 可感知的流量调度管理方案，根据 QoS 需求将网络流量进行分类，QoS 流以实现弹性的 QoS 路径安排为目标进行路由，普通流路由则以提高链路利用率和实现网络的负载均衡为目标，从而进一步提高网络的服务质量。围绕控制器，分析实现方案需要添加和优化的主要功能，设计详细的实现方案，并在控制器原型基础

上实现所需模块。最后对系统进行功能和性能进行测试。

1.3.2 本人工作

本人工作主要为控制器的开发，即以 Floodlight 控制器为原型设计并实现上述流量管理策略，主要内容可分为以下几个部分：

（1）研读 SDN 相关文献，分析 SDN 下 QoS 技术和流量调度相关内容，研究如何利用 SDN 的优势，提升网络的服务质量，保障数据流的传输。

（2）对系统进行需求分析和设计，分析实现方案需要实现的功能。方案中需要具备 QoS 需求获取，转发设备的数据监测，拓扑的管理，QoS 流和普通数据流的路径规划等多种功能，根据功能需求明确控制器的模块设计。

（3）根据设计方案对进行编码开发。在控制器中实现应用接口，通过 REST API 进行 QoS 流配置，使用 QoS 模块对流信息进行管理，优化网络监测模块以获取尽可能精确的转发设备数据，实现 QoS 感知的路由算法，完成路径的规划和流表的构造等内容。

（4）实验验证方案效果，搭建 SDN 仿真网络环境，对控制器的功能以及流量管理策略的效果进行测试，并对测试结果进行分析。

1.4 本文的组织结构

本文分析了当前 SDN 下 QoS 技术的研究和应用情况，指出了存在的部分突出问题，本文以 SDN 中 QoS 流量管理策略的为研究点，设计并实现了 QoS 可感知的路由方案和流量调度方案，基于 Floodlight 控制器进行开发实现文中方案，并对控制器各个模块的设计和实现进行了详细的阐述。文章最后通过实验验证控制器的功能，以及上述方案的效果，并对实验结果进行分析。本文研究内容总共分为 6 章，具体结构安排如下：

第 1 章首先概括性介绍当前网络架构下 QoS 技术的现状和主要挑战，接着引出 SDN 技术的优势，并介绍了 SDN 研究背景和应用场景，然后进一步阐述了本文的研究内容，同时还介绍了国内外相关研究现状。

第 2 章主要介绍文中 SDN 的发展历程及其架构，并详解阐述 OpenFlow 协议的原理、基本部件、处理流水线 pipeline 以及流表等主要技术内容，同时还介绍了的 OpenFlow 网络下的队列和计量表等 QoS 技术。

第 3 章对系统进行需求分析和概要设计，阐述了系统的功能性需求和非功能性需求，介绍了文中流量调度方案的主要目标，对 Floodlight 控制器及其整体功能进行介绍，并对实现所述方案需要增加的功能进行了设计，对控制器需要改动的模块和部件需要实现具体功能进行了说明，并对相关技术指标进行了介绍。

第 4 章对系统的设计和实现进行了详细的介绍，分模块讲述了各个模块的功能和运行流程，基于 SDN 的 QoS 流的路由方案，基于计量表 Meter 实现的流量调度方案。

第 5 章对系统进行测试，首先对 SDN 仿真技术进行介绍，然后介绍了实验环境，测试的条件等。实验中对 QoS 路由策略进行测试，列出了控制器端和客户主机端的流量监测数据，并系统运行结果数据进行分析。

第 6 章对文中 QoS 感知的流量调度方案进行了总结，指出了实现过程中的不足，结合 SDN 相关技术的对网络服务质量发展进行了展望。

1.5 本章小结

本章讲述了论文的研究背景，传统网络下网络管理的缺点，网络难以贴近应用服务需求实施网络服务质量保障的问题，然后介绍了 QoS 感知的网络的思想和通过软件定义网络实现细粒度和弹性服务质量的思路。进而介绍了软件定义网络的特点和优势，说明了软件定义网络的广阔的前景和巨大的研究意义。最后对本文的主要研究内容和主要工作内容进行介绍。

第 2 章 相关技术介绍

本章将介绍软件定义网络的基础理论和知识，介绍常见的控制器，软件交换机等内容，并详细介绍 OpenFlow 规范及其原理以及其 QoS 技术等内容。

2.1 软件定义网络架构

2011 年开放网络基金会（Open Network Foundation，ONF）成立，ONF 致力于推动 SDN 架构和技术标准，随后成为了 SDN 标准化和产业化的核心组织。ONF 在白皮书《Software-Defined Networking :The New Norm for Networks》[25] 中提出了 SDN 的经典体系架构，如图 2.1 所示。该架构将网络划分为控制层，基础设施层和应用层。控制层则负责整个网络的集中管理，基础设施层根据控制层的策略进行数据转发，应用层开放可编程接口由用户按需进行自定义开发。

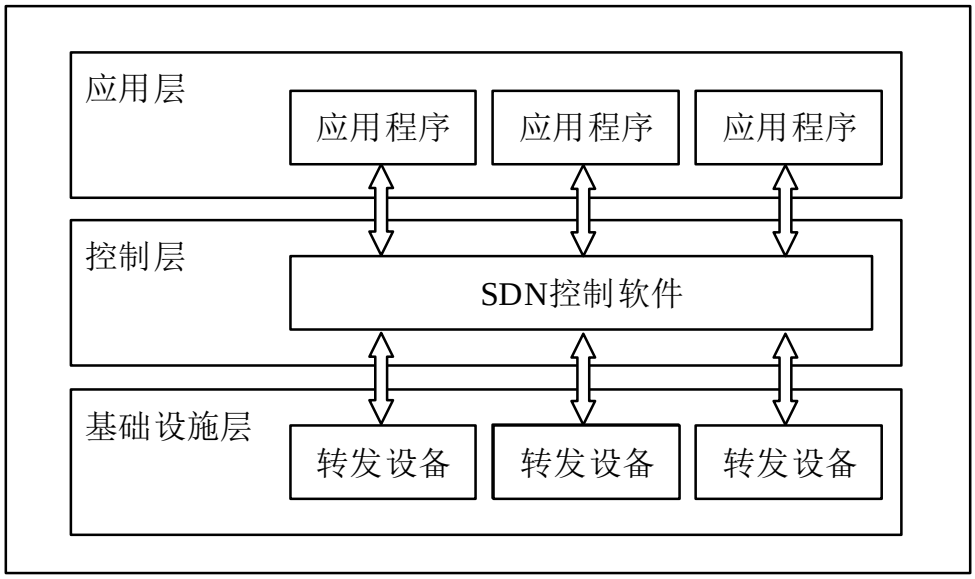

图 2.1 ONF 定义的 SDN 架构图

下面对软件定义网络的关键概念进行介绍：

（1）控制平面

控制平面（Control Plane）又称控制层，是整个网络的中心。SDN 中所有的控制逻辑都集中在控制层，网络通常使用一个或多个控制器对网络进行管理。控制器向上提供编程接口，并通过统一的方式对数据层上的设备进行管理。控制器需要将应用层软件调用的功能“翻译”成数据层转发设备的具体行为，并传达到对应的设备上。控制器集中控制的特点使得 SDN 可以全局掌控网络行为，实施有效的资源调度和分配策略。

（2）数据平面

数据平面（Data Plane）又称数据层，数据层的一个原则是保持相对简单的架构，以保证高效的数据处理。数据平面上最常见的转发设备为交换机，SDN 交换机没有路由功能，路径规划的功能由控制层进行，交换机按控制器制定的转发逻辑对数据进行转发。这种结构避开了传统网络下运行的大量分布式协议，使用同一形式的转发设备，使网络架构更加简单。

（3）应用平面

应用平面（Application Plane）又称管理平面，通常由一部分专门制定的网络管理软件组成，这些软件通过控制器的开放北向接口进行功能调用，实现对网络的管理。常见的软件有路由软件，防火墙，负载均衡软件和网络监测软件等，这些软件实现可以实现网络管理的自动化和智能化，使网络的功能更加完善。

（4）北向接口

北向接口（Northbound Interface）是控制层和应用层间的接口，可以为上层应用提供底层网络的信息抽象，该接口服务于上层应用，其设计与应用业务类型密切相关。由于应用业务种类繁多，需求复杂，北向接口的通用性就显得尤为重要，因为这将直接决定网络管理的便捷性和网络的可扩展性，良好的北向接口能为应用层业务提供高效的网络管理。一些组织正在推动北向接口的标准化，如 ONF 提出了 NBI 标准，OpenDaylight 的 REST 标准等，但这些标准并没有得到进一步的推广。因而，目前北向接口的制定标准仍没有得到统一，而是由各厂商或网络管理者按需开发。

（5）南向接口

南向接口（Southbound Interface）是数据层和控制层间的接口，可以为 SDN 控制器传递网络控制指令，南向接口采用统一的通信标准，目前最常见的南向接口为 OpenFlow[26]。南向接口的统一屏蔽了不同厂商设备的差异，避免设备供应商对设备的接口限制，有利于化简网络结构。

2.2 SDN 常见应用场景

目前，SDN 常见的应用场景包括有校园网络、企业网络、数据中心网络、电信网络等。

（1）校园网络

SDN 诞生于高校科研课题，首先应用到校园网络中。由于校园网络结构的复杂度低，新型网络架构的部署会比较顺利，科研人员可以在实验室中对网络性能进行测试，相比企业网络灵活性更大。目前斯坦福大学已经完成了 SDN 校园网的部署，其他一些高校也逐步开始部署 SDN。

（2）企业网络

不同企业的网络间差异巨大，企业需要根据业务的变动及时对网络架构进行调整。SDN 架构具有强大的管控能力和良好的可扩展性，网络规模变动时需要只需要调整数据层设备，网络控制逻辑不发生改变，并且控制器中可预先配置多种管理方案，方案在数据层设备扩展完毕后可以直接采用。部分大型企业已经在企业内部网络中应用了 SDN。

（3）数据中心网络

数据中心网络流量巨大，需要充分利用带宽资源。因此提高链路利用率、减少能耗，提高系统性能等问题是数据中心网络中的重要问题。SDN 架构保障了数据中心网络可以实现集中的自动化控制，完成路径的智能安排和灵活的数据转发，同时利用 SDN 的开放接口制定网络监测和数据分析策略，能满足多租户按需分配资源，虚拟机部署和迁移等多种应用场景。Google 的 B4[27]项目就是一个极为成功的例子，在部署 SDN 后使得数据中心间链路利用率从 30%~40%提高到接近 100%，该项目是 SDN 在数据中心中应用的典范。

（4）电信计费网络

电信运营商网络结构十分复杂，覆盖地区广，并且流量需要精细规划。SDN 的应用能帮助运营商降低运营成本，提高资源的使用效率。目前在电信网络中通常采用逐步过渡的方法，在一些关键节点上将交换机替换成 OpenFlow 交换机，使得控制器能够管理核心链路资源，其他节点间仍保持原有方式，在保障网络稳定性的同时逐步更新网络架构。

除了上述提到的领域，SDN 在物联网，车载网络和军用网络等诸多方面都有相关应用。

2.3 OpenFlow 协议

OpenFlow 是 SDN 的首个标准化协议，最初 OpenFlow 只是为了给科研人员提供一个创新网络实验平台而被应用到校园网中，随后由于其强大的优势迅速得到了工业界的肯定，得到了广泛的关注。OpenFlow 协议从 2009 年的 1.0 版本到现今的 1.5 版本历经多次修改优化，成为了 SDN 最主要的实现标准，OpenFlow 网络也成为 SDN 最常见的实现形式。

OpenFlow 中定义的 SDN 基本组件有控制器，OpenFlow 交换机和控制通道。

（1）控制器

控制器是管理网络的核心软件部件，能通过 OpenFlow 协议控制交换机的行为，调整数据转发方式，并向上层应用提供网络接口。目前业界常见的开源控制器有：OpenDayLight，NOX/POX，RYU，Floodlight 等等。

SDN 控制器需要具备一定的功能：首先，控制要支持的特定南向接口协议

（如 OpenFlow 协议），使得控制器可以对使用统一的方式对转发设备进行管理。其次，控制器需要支持软件编程，这是 SDN 的一大特点之一，网络的管理效率很大程度上取决于控制器的软件功能，完备的软件功能充分发挥 SDN 的优势。再次，控制器需要具备一定的扩展性和开放性，考虑到网络的架构和网络用户的需求往往是会随而时间变动，在使用过程中控制器的功能需要根据网络实际运行情况进行功能的调整，需要充分兼顾到不同用户及开发人员的使用。最后，控制器应该具备良好的稳定性和一定的故障恢复能力，以保障网络的稳定运行。

（2）控制通道

控制通道是连接控制器和 OpenFlow 交换机的通道，控制器可以通过该通道将配置消息传递到交换机的并接收来自交换机的各类事件，通常在控制器会和所有交换机建立控制通道。SDN 中控制平面和数据平面进行分离，控制信息通过控制通过进行传达，因而控制通道必须具备一定的处理能力，否则容易成为网络的瓶颈。一个典型的例子是，网络中短时间内涌现大量数据流，交换机无法在短时间内将信息传达到控制器，控制器也无法下达命令更新转发路径，由于交换机无法自行实现流表调整，致使新数据流无法正常转发。所以，开发人员需要关注控制通道带来的延迟和管理开销。

（3）OpenFlow 交换机

OpenFlow 交换机按照 OpenFlow 标准和控制器进行通信，主要负责数据平面上的数据的转发，能根据控制器的指令执行一系列动作。

OpenFlow 物理交换机使用三态内容寻址存储器（Ternary Content Addressable Memory, TCAM）实现流表的高效的报文匹配，但 TCAM 的造价是较为昂贵，导致了现今物理交换机流表容量较为有限，因而已经有较多文献研究了流表的聚合（Flow Aggregation），通过单个流表项处理多条数据流的转发以有效地减少流表的占用[28]。在大型网络中，控制器在路径规划时也尤其要注意流表的使用，避免因 TCAM 容量不足导致流表加载失效。

而软件交换机的引入则可以从一定程度上弥补物理交换机不足，软件交换机实际上是安装在服务器上的软件，软件具了备物理交换机的功能。从性能上看，软件交换机的数据转发能力要弱于物理交换机，数据转发时延更高，软件交换机的内存更大，因而其控制通道的处理能力和流表存储能力要比物理交换机更强，并且服务器主机可以对软件交换机的计算和存储资源进行灵活地分配，使用更加灵活。所以数据中心中常常将两种交换机搭配使用，在网络边缘节点上使用软件交换机处理新增流量，非边缘节点上使用物理交换机进行高速的数据转发。

2.3.1 流水线 Pipeline

OpenFlow 交换机使用流水线 pipeline 对数据报文行处理，pipeline 中包含一

个或多个流表（Flow Table），每个流表由多个流表项（Flow Entry）组成，Pipeline 控制着数据包通过流表的逻辑和处理的方式。对于含有多级流表项的 pipeline，数据包会先经过标号为 0 的流表进行处理，随后被转移到输出端口进行转发，或者按照流表项标号的顺序转移到下一级流表进行处理。Pipeline 会将数据包和流表项中的匹配域（Match Field）进行匹配，若匹配成功则执行流表项中的动作域（Instruction）；若数据包与流表中的所有表项都不匹配则会触发 table miss 事件，此时 pipeline 根据表项的配置可以将报文封装并通过控制通道传递给控制器，或将报文丢弃，抑或是将报文传递到下一个表项进行处理。整体流程如图 2.2 所示。

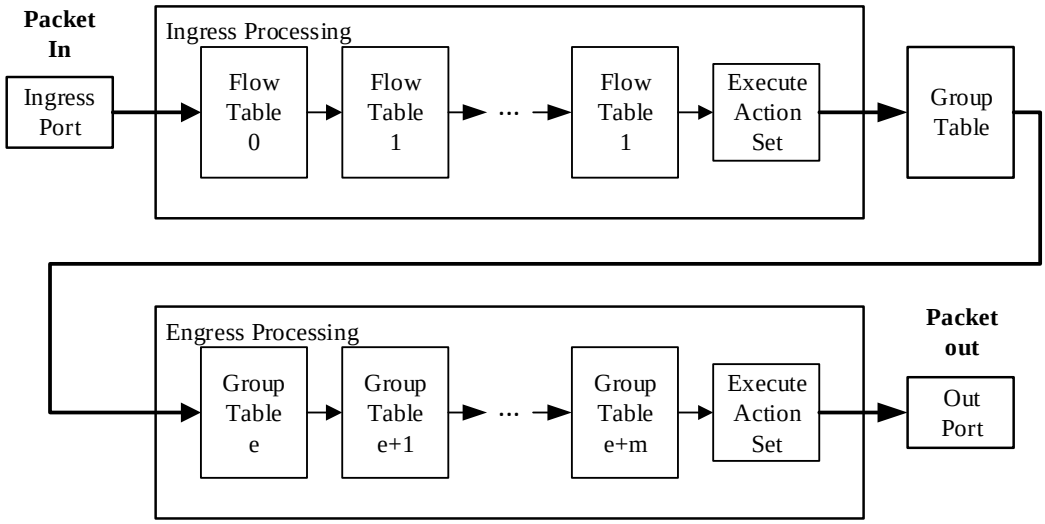

图 2.2 pipeline 处理流程图

2.3.2 流表 Flow Table

SDN 中以流（Flow）为单位进行数据转发，流表的默认处理流程为：

（1）如果一条流与流表中的所有表项都不匹配，将数据流的首个报文发送至 SDN 控制器；

（2）控制器接收到报文后，计算数据流在网络内的转发路径，生成路径信息，将流表更新信息下发到转发路径上的所有交换机上；

（3）交换机更新流表信息并开始转发数据，同一条数据流的报文使用相同表项进行转发，后续转发过程不需要控制器在进行介入。

OpenFlow 交换机的流表（Flow Table）包含多个流表项（Flow Entry），流表项的结构如图 2.3 所示。交换机在进行数据转发时，会将根据数据包匹配域（Match Field）和流表项进行匹配，找到一条最高优先级 Priority 的流表项，并执行流表项中动作域 Instructions 中设置的动作。

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

图 2.3 Flow Entry 结构图

（1）匹配域 Match Fields：匹配域定义了识别数据流的字段，具有相同匹配字段值的报文被视为同一数据流的报文，交换机选取与报文具有相同匹配字段的流表项，并执行流表项中规定的动作。OpenFlow 1.3 标准中定义了 13 个必须支持的匹配域，如表 2.1 所示。

表 2.1 OpenFlow 常见匹配域表

匹配字段	描述
IN_PORT	汇聚端口
ETH_DST	以太网帧目的地址
ETH_SRC	以太网帧源的地址
ETH_TYPE	以太网帧类型
IP_PROTO	IP 报文协议
IPV4_SRC	IPv4 报文源地址
IPV4_DST	Ipv4 报文目的地址
IPV6_SRC	IPv6 报文源地址
IPV6_DST	Ipv6 报文目的地址
TCP_SRC	TCP 报文源地址
TCP_DST	TCP 报文目的地址
UDP_STC	UDP 报文源地址
UDP_DST	UDP 报文目的地址

此外，还有其他匹配域如 Metadata，IP_DSCP，IP_ECN 等

（2）优先级（Priority）：流表项的优先级，若数据包与多个流表项匹配，则执行优先级高的流表项。若存在多条优先级相同的流表项，执行顺序为未定义。

（3）计数器（Counters）：对流表项处理的数据包数量进行计数。

（4）指令域（Instruction）：流表中包含一系列需要执行的指令集，支持的指令有执行动作域（Apply-Actions），清空动作域（Clear-Actions），通过计量表（Meter meter-id），转到下一流表进行处理（Goto-Table next-table-id）等，动作的含义见表 2.2。

（5）动作域（Actions）：当指令域中的动作执行结束后，流表设置的动作集（Action Set）将会被依次执行，一个动作集可以包含多个动作。常见的动作有：添加 MPLS 标签（push-MPLS），添加 VLAN 标签（push-VLAN），减少 TTL 字段（decrement TTL），设置特定字段值（set-field），设置队列（enqueue），设置组表，和转发数据包等多种动作，详细描述见表 2.3。

（6）超时时间（timeouts）：每条流表项都有生存时间，设置 idle-timeout 值会使流表项空闲时间超过时限后会被删除，设置 hard-timeout 值会使流表项在固定时限之内被删除。

表 2.2 OpenFlow 指令表

指令类型	参数	描述
Meter	meter_id	动作将数据包转移到特定的计量表中进行处理。
Apply-Actions	actions	不对动作域进行改动，立即执行动作域中的所有动作。
Clear-Actions	无	清除动作域中的所有动作。
Write-Actions	actions	将特定的动作写入动作域中，
Write-Metadata	metadata/mask	将特定元数据信息替换报文原有信息，
Goto-Table	next-table-id	将数据包转移到下一个流表中进行处理，下一个流表的标号必须大于前一个流表的标号

表 2.3 OpenFlow 动作表

动作类型	参数	描述
Ouput	port_id	将报文转发到指定端口。
Set_Queue	queue_id	将报文转移到指定队列中进行处理。
Drop	无	丢弃数据包。
Group	Group_id	将报文转移到指定组表中进行操作。
Push-Tag/ Pop-Tag	VLAN_header/MPLS_header/PBB_Header	将特定元数据信息替换报文原有信息，
Set-Field	filed_name value	将报文特定字段值进行修改。
Change-TTL	Change_type TTL	对生存时间 TTL（time to live）字段进行修改。

2.3.3 组表

组表增强交换机的转发行为。和流表一样组表中包含多条组表项，组表由组标记，组类型，计数器和动作筒集构成，如图 2.4 所示。

图 2.4 组表结构图

（1）组标记（Group Identifier）：一个 32 位的唯一标记。

（2）组类型（Group Type）：组类型有四种：all，select，indirect，fastfailover。组类型设置为 all，组表项会执行所有筒中的动作，常用于报文的多播和广播。组类型设置为 select，会执行表项中其中一个筒的动作，可以对不同的筒设置权值，使得数据报文按比例执行不同筒中的动作。组类型设置为 indirect 时，动作筒集中只能包含一个筒，可以将多条流表关联到同一条组表并执行相同的组动作。组类型设置为 fastfailover，执行其中一个可用的筒。

（3）计数器（Counter）：统计表项处理的报文数

（4）动作筒集（Action Buckets）：动作筒集中包含多个动作筒，每个动作筒内包含一些列动作，一旦某个筒被选中，筒中的全部动作都会被执行。

2.4.4 OpenFlow 消息的结构

OpenFlow 协议规范了 SDN 控制层和数据层的通讯，OpenFlow 消息都有着相同的头结构，包含一个 8 位的版本号信息，一个 8 位的类型信息，16 位的消息长度信息以及一个特有 32 位的数据包标记信息，如图 2.5 所示：

```
/* Header on all OpenFlow packets. */
struct ofp_header {
    uint8_t version;    /* OFP_VERSION. */
    uint8_t type;       /* One of the OFPT_ constants. */
    uint16_t length;    /* Length including this ofp_header. */
    uint32_t xid;       /* Transaction id associated with this packet.
                           Replies use the same id as was in the request
                           to facilitate pairing. */
};
```

图 2.5 OpenFlow 消息头结构图

OpenFlow 消息有三种类型：

（1）控制器-交换机（Controller-to-switch）消息

控制器-交换机消息是由控制器发起的，用于管理交换机或查询交换机状态的消息，消息的内容如表 2.4 所示。

表 2.4 控制器-交换机消息表

消息类型	描述
Features	用于查询交换机特性的消息，包含交换机数据通道 ID，支持的流表数量，（packet-in）数据包的处理能力以及交换机表的部分描述信息。
Configration	用与查询交换机具体配置参数的消息，该消息可以设置发送到控制器报文的小大，和 IP 分片的处理。
Modify-State	用于修改交换机状态的消息，支持增加，删除和修改三种动作。该消息可以对交换机行为进行控制，常用于对交换机流表，组表，计量表等表项的修改。

续表 2.4 控制器-交换机消息表

消息类型	描述
Read-State	用于监测交换机运行状态的消息，常用于网络状态的监测，可以查询到表项的报文处理数量，端口处理的字节数和报文数等多种信息。
Packet-Out	PakcetOut 消息将 PacketIn 消息承载的原始报文信息进行封装后返回给交换机，通常在新数据流的处理过程中使用。
Barrier	用于检测交换机是否完成特性行为的消息。消息只含有头部信息，消息体为空，交换机在全部完成上一个（批）指令后，回应此消息。
Role-Request	用于调整交换机（主/从/同等）角色的消息，可用于多控制器网络。
Asynchronous-Conguration	用于异步配置交换机的信息，常用于多控制器网络。

（2）异步（Asynchronous）消息

异步消息是由数据层设备发起的，用于通知控制器具体设备的配置或运行状态发生改变的消息，消息是单向的。消息类型如表 2.5 所示：

表 2.5 异步消息表

消息类型	描述
Packet-In	用于通知控制器新数据流到达的消息。该消息通常在数据流的首个数据报文与交换机流表中所有表项都不匹配时产生。
Flow-Removed	用于通知控制器流表已经删除的消息，可用于监测流表项是否已经超过了其存活时限的场景，只有经过特定标记的数据流被删除才会产生此消息，
Port-Status	同于通知控制器交换机的端口发生变化的消息，该消息保证了交换机在收到主机进行端口的增加，删除和修改时，控制器能及时地收到改动信息。
Errors	用于通知控制器交换机运行出现错误的信息，常用于提示控制器发送了错误的命令的情形。

（3）对称（Symmetric）消息

消息类型如表 2.6 所示：

表 2.6 对称消息表

消息类型	描述
Hello	用于控制器和交换机建立链接时的试探信息，可以相互通知支持的 OpenFlow 版本信息。
Echo	用于检测控制器和交换机链接状态的信息，接收到 Echo 信息的一方必须回应一个 reply 信息，可用于检测链接是否正常，也可以在 Echo 信息内加入时间戳以检测链接的时延。
Experimenter	同于传递 OpenFlow 规定消息之外的其他信息的消息。

OpenFlow 的框架内，所有信息都需要封装到 OpenFlow 消息体内，并通过控制层和数据层的专用通道进行传输。

2.4 SDN 中的路由技术

目前，SDN 路由选择算法的研究主要基于 QoS、能量、资源偏好、应用程以及权重等需求进行路径规划[29]， SDN 中常见的路由算法有：

（1）基于最短路径的路由算法

目前较为常见的最短路径算法通过 Dijkstra 算法计算最短路径，网络路径的代价用节点间的跳数（Hop Count）计算，求出源节点到目的节点的累积代价最小的路径。该路由算法在数据转发过程中没有考虑链路负载，在多条数据流的最短路径重合时容易造成某些链路的拥塞。

（2）ECMP 算法

ECMP 算法通过链路权值的调整，可以将数据流映射到多条路径上。该算法通过哈希计算生成数据流映射表，表中包含着数据流与路径的映射信息，数据流通过查表进行转发。ECMP 方法在路径确定后，后续的数据流都将沿着相同路径进行转发，算法缺乏动态调节路径能力，在应流量的不确定性问题上时效果不佳。

（3）QoS 路由算法

该种算法将时延，丢包，带宽等 QoS 参数作为路由的因素，综合链路的情况对数据进行路由。通常，QoS 的路由算法可以定义为一个多约束选路问题，根据 QoS 参数为网络内的每一条链路设置一个权值，应用业务对网络链路有一定的约束值，求在满足约束条件下的可行路径的问题。该算法需要精确的网络运行状态信息，并需要进行大量的计算，因此要求控制器或服务器具有较强的处理能力。

2.5 SDN 中的 QoS 技术

OpenFlow 标准中提供多种机制以实现 QoS 策略，最初在 OpenFlow 1.0 版本定义了队列功能用于控制端口速率，匹配域中加入 VLAN 优先级和 ToS 字段两项，然后在 OpenFlow 1.1 版本中增加了 MPLS 标签和 MPLS 流量类别字段，进而 OpenFlow 1.2 版本中增加了设置队列最大速率的功能，随后 OpenFlow 1.3 版本中引入了计量表，使 OpenFlow 能对单一流的速率进行限制，能实现更加精细的流量管理。

2.5.1 队列

队列（Queue）是交换机输出端口上的数据包调度技术，队列输出需要和端口绑定，队列可以保障数据流的最小速率和限制数据流的最大速率，在流表项中使用动作 enqueue 可以将数据包关联到队列上。

队列技术常用于带宽分配，如文献[20]中，使用队列配合计量表对数据流带宽进行保障，使 QoS 流在传输过程中可以接近零丢包，同时通过合理地带宽分配使 best-effort 流尽可能的利用到链路带宽，能够实现高效而弹性的 QoS 保障。

2.5.2 计量表

OpenFlow1.3 版本中引入了计量表 Meter，用来测量和控制数据流的速率。计量表可以监测单条数据流或多条数据流的速率，并在其速率超过设定值后执行特定的动作。计量表结构如图 2.6 所示，Meter Table 中每条表项包含标记（Meter Identifier），Meter Band，和计数器（Counter）三项内容。其中 Meter band 会监测数据流速率，当速率超过设置的 Rate 值时，将执行 Meter Band 中设置的动作（Type specific arguments），OpenFlow1.3 中支持的动作有 dscp remark 和 drop 种，dscp remark 动作可以更改 IP 报文的 dscp 字段值，可以用作 IP 报文服务等级的调整，drop 动作会将超过速率值的数据包丢弃，可以作为流量整形的一种手段。

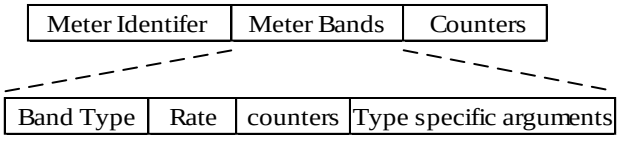

图 2.6 Meter Table 和 Meter Band 结构

文献[30]提出通过计量表和组表来应对流量的不确定性问题，文献中控制器默认为数据流安排两条路径，用计量表监测数据流速率，数据流速率低时只使用其中一条路径进行传输，速率超过计量值后使用两条路径传输，同时通过组表调整转发到两条路径上数据的比例。过程中由计量表和流表对转发路径进行控制，不需要控制器的参与，因而可以减少控制层和数据层的控制代价，能很好地应对

流量突增问题。计量表能对单一流或多条流的总速率进行控制，是一种十分灵活而有效的流量调控机制。

2.6 本章小结

本章讲述了文中涉及主要 SDN 技术的知识和原理。首先介绍了 SDN 的发展历程，SDN 的三层架构及其控制平面和数据平面分离和集中控制的特点，以及 SDN 相对于传统网络的突出优势，然后介绍了 SDN 最常见的实现形式 OpenFlow 网络和 OpenFlow 协议的相关内容，接着详细阐述了 OpenFlow 基于流的转发方式，并介绍 OpenFlow 中的 QoS 技术。

第 3 章 需求分析与概要设计

针对 QoS 可感知的流量调度策略实现过程中控制器需要具备的功能进行分析，对 Floodlight 控制器原型进行介绍，对控制整体功能进行概述，并对流量管理方案所需实现的功能结构进行概要设计。

3.1 总体目标

从网络资源管理的角度上看，网络可以分为专用网络和通用网络。专用网络如电信计费网络，金融网络或政府网络等网络，这些网络中的数据十分重要，传输过程中需要严格保障数据流的传输质量，因此该种网络主要关注如何进行资源的分配和故障恢复等方面，网络在必要时能根据确切的 QoS 参数定制专用路径，安排备用方案等等。而在校园网和企业网等通用网络中，QoS 的指标往往并不十分严格，并且现今网络应用种类很多，新旧业务变更迭代速度快，业务见无法形成适当的 QoS 需求度量标准，维护固定的 QoS 应用需求列表的方式往往不太现实，网络维护成本决定了通用网络往往难以实行严格的 QoS 策略。

文中流量管理策略应用的适用范围也主要为通用网络，策略更注重 QoS 服务的灵活性和便捷性，流量调度考虑了不同数据流对服务质量之间的差异，结合网络实际情况进行有效的资源分配和链路规划，充分利用 SDN 的优势，服务于不用类别的应用业务。流量调度策略的主要目标为：在路径规划时能充分考虑到应用流量的 QoS 需求，使得 QoS 流按适合其需求的路径进行转发，同时普通流按照按低负载链路进行转发，必要时可限制普通流的速率，使得网络中的 QoS 流得到优先处理并且网络链路利用率得到提高。

3.2 功能性需求

网络控制器面向的使用人员为网络管理人员或者网络运维人员，系统角色定义为网络管理员，网络管理员对负责 QoS 数据流进行管理，对控制器的运行进行控制。

图 3.1 为网络整体用例图，图中展示了网络管理员主要的动作实例。模块管理功能为对控制内部的模块进行控制。拓扑管理负责对网络结构形态进行抽象并提供信息呈现，并监测网络设备和链路的变动。设备管理负责管理网络设备信息，维护控制器和交换机间的链接等内容。路由管理内容包括数据流的转发方式，路径的计算以及，是控制器最核心的功能。QoS 管理主要是对 QoS 数据流进行管

理，控制器需要维护详细的应用业务 QoS 流信息，为 QoS 路由提供依据。应用接口管理负责对控制器和应用程序之间的接口进行管理。

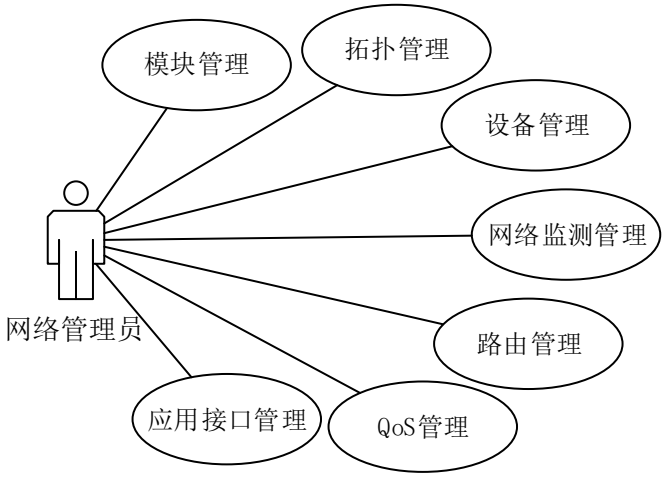

图 3.1 系统整体用例图

其中，模块管理，拓扑管理和设备管理等功能原型控制器已经具备，文中实现的主要功能为 QoS 管理，网络数据监测管理和路由管理。

3.2.1 QoS 管理需求

文中将网络数据流划分为 QoS 流和普通流，控制器需要为 QoS 流提供时延，丢包或带宽更佳的转发路径，控制器需要维护 QoS 流的信息。QoS 的配置可以由应用程序通智能配置，也可以由网络管理员手动设置，因而控制器需要提供数据流的查询，更新和删除功能。图 3.2 位 QoS 流的管理用例图。

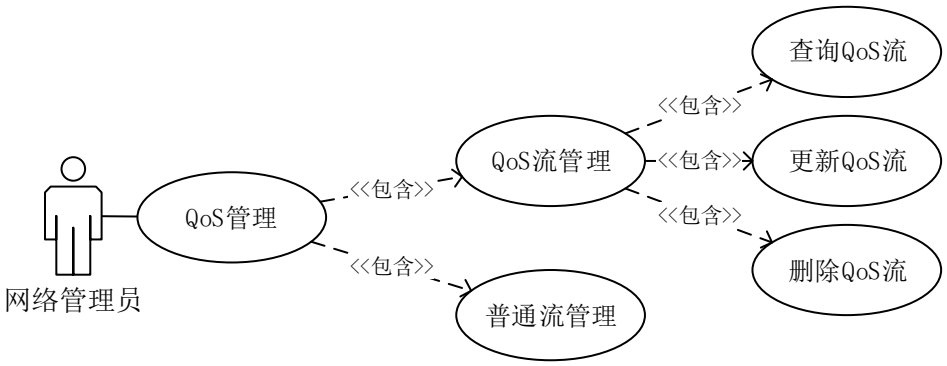

图 3.2 QoS 流管理用例图

3.2.2 网络数据监测需求

为了掌控网络的运行信息和细粒度的流量调度，网络运行过中控制器需要对交换机和数据进行监控，监测的数据包括交换机流和计量表信息，端口的发送和接收数据和链路的延迟信息等信息。因而需要在控制器内制定一套信息采集方案，对采集到的数据进行分析和处理，保证控制器在路由时查询到的网络设备信

息为有效信息。从监测功能上看，还应该设置。网络管理员网络监测操作的用例如图 3.3 所示。网络管理员可以根据网络实际需要调整监测的数据类型，并调整数据采集的时间频率。

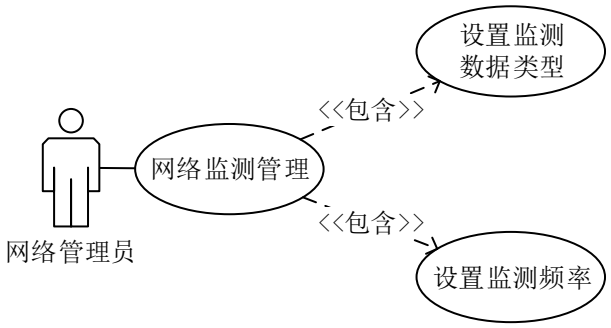

图 3.3 网络监测管理用例图

3.2.3 路由计算和流量调度需求

路由功能是控制器的核心功能。为了实现 QoS 感知的路由，除了具备基本的路径计算功能，控制器还需要具备如下功能：

（1）QoS 流识别功能：控制器需要对交换机提交的新数据流解析，识别数据流的类型，从而实现对不同需求数据流的针对性寻路功能。

（2）链路代价计算功能：控制器在收集到链路数据后，需要根据 QoS 指标对链路代价进行计算，链路代价需要对链路的表现有较好的。

（3）路径规划能：模块需要通过算法为数据流选择特定 QoS 指标表现更好的路径，减少 QoS 流该流在传输过程中受到的干扰。

（4）流量调度功能：为了进一步保障 QoS 流的传输质量，充分利用网络带宽资源，控制器需要对普通数据流进行一定的调控，让 QoS 没有特殊要求的数据流使用低负载路径进行传输，提高链路的利用率。

路由管理用例图如图 3.4 所示。

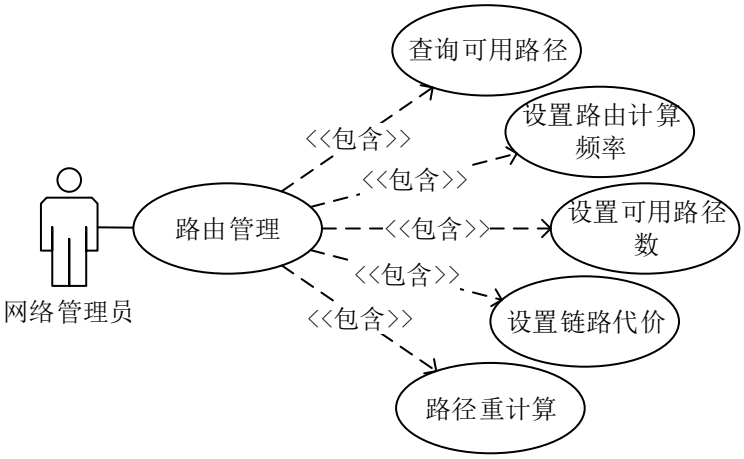

图 3.4 路由管理用例图

路由模块中大部分功能为控制器内部功能，同时模块还开放了一部分路由调整功能，包括查询可用路径，设置路由计算频率，设置链路代价种类，设置最大可用路径数，路径重计算等几个动作。网络管理员可以根据网络的运行情况和实际需要路由模块进行调整。

3.3 非功能性需求

系统非功能性需求是衡量系统表现的重要。与功能性需求的注重点不同，非功能性需求关注的是软件的质量，性能，可靠性以及扩展性等诸多方面，虽然非功能性需求通常不会设置硬性的评价指标，但不意味着这些方面不需要实现，低质量的代码会使系统效果大打折扣，极大地降低用户体验，因而非功能性需求也是需求分析中的重要步骤。

（1）易用性

网络控制器不仅要具备完善功能，同时还要保证控制器具有良好的易用性。由于网络管理涉及到底层硬件，网络管理软件需要提供良好的接口，让网络管理员能避开网络底层设备细节而专注于网络的控制逻辑，使得网络管理更加高效。

（2）准确性

为了保障控制器能够实现高效的流量管理，控制器需要获取到精确而有效的数据层数据。滞后的数据不能很好地反应网络情况，而频繁的数据更新和运算又会浪费系统资源网络，因而网络监测需要在保证数据竟可能准确的同时不给控制器带来过大的处理压力，控制器需要对数据采集频率进行适当的控制。

（3）可靠性

网络传输具有不稳定性，控制器在监控网络时有可能接收到错误数据或重复数据，因此控制器需要对接收到的数据进行验证，避免失效数据导致控制器出现错误的行为，造成流量调控失效或控制器崩溃。

3.4 控制器概要设计

3.4.1 Floodlight 控制器简介

Floodlight 控制器是由 Big Switch 公司开发的一款基于 Java 实现的企业级控制器，是目前最主流的控制器之一。控制器核心代码经过了严格的测试，具有较强的可靠性，支持多种 Openflow 版本。Floodlight 控制器在企业和学术研究中都比较常见，得到了较为广泛的认可。控制器使用模块化开发，依据 OpenFlow 规范实现了控制器的核心功能并为应用程序提供了丰富的接口，支持 REST API，具有良好的扩展性。Floodlight 使用者可以根据需求开发自定义模块。

图 3.5 为 Floodlight 控制器的架构图，可以看到控制器内的功能相当完备，结构层次清晰，下面将介绍控制器主要模块的功能：

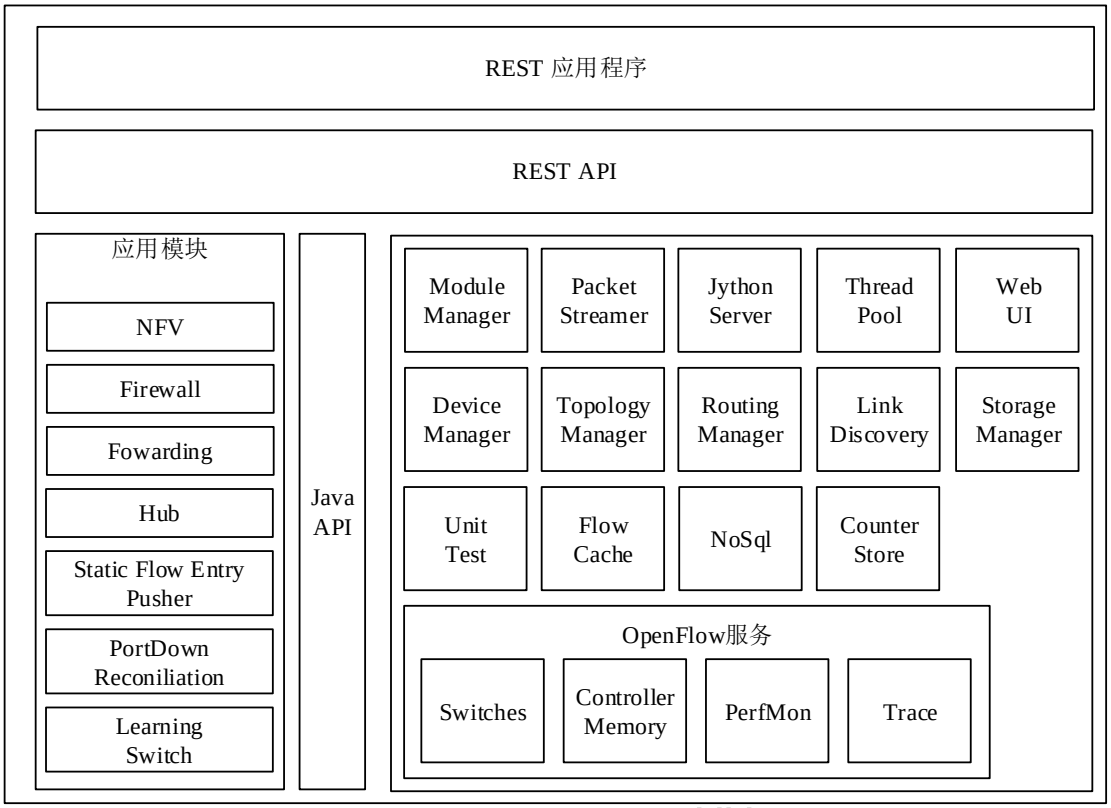

图 3.5 Floodlight 架构图[31][32]

（1）基础管理模块

Floodlight 提供了专门的部件用于模块的管理，通过配置文件设置模块的加载，控制器可以判断各个模块的依赖项，按需加载模块。

（2）拓扑管理模块

拓扑管理模块维护这整个网络的拓普信息，使用链路发现模块实现对链路的发现，通过定时的广播数据包进行链路的变动进行快速监测。一旦某条链路或某个节点出现变动，模块会立即进行拓扑信息的更新，以保证路由模块等其他模块能依据最新信息做出正确决策。拓扑管理模块向其他模块提供查询网络链路状态，端口状态，拓扑的更新时间等多种功能。

（3）设备监控模块

控制器提供了模块对交换机的信息进行收集，支持对端口带宽的计算和查询。控制器对给出了监测网络设备信息的基本样例，开发者可根据其样例进行开发，对流表，组表，计量表等多种信息进行监测。

（4）转发模块

转发模块负责数据流的识别和数据转发策略，处理过程中主要围绕 PacketIn 和 PacketOut 消息的解析和构造，一般流程为：新数据流到来时解析交换机发送

的 PacketIn 消息，调用路由模块接口得到结果，根据 OpenFlow 配置信息生成 PacketOut 消息，并转交给通讯模块。其中 PacketIn 消息中包含了数据包的原始信息（原地址，目的地址，ToS 字段等），对数据流的识别和分类也将在该模块进行。

（5）路由模块

路由提供网络内各节点的路径，管理数据流的路由方式，支持数据的转发，范洪，多播和抛弃动作。模块默认采用的基于目的地的选路，是一种 best-effort 的路由方式，能为数据流快速找出源节点到目的节点的路径，同时也提供了多路径的路由方式供开发人员进行使用。

（6）Rest API 服务模块

Floodlight 控制器通过 REST API 向外部应用提供交互的接口，REST API 能通过 HTTP 的形式为应用提供底层网络的信息呈现，控制器中已经实现了基本的 web 信息展示界面。RESTFul 风格的架构是目前比较成熟的网络应用程序 API 架构，符合该架构的接口称为表述性状态转移-应用程序编程接口（REST API），该架构将数据抽象为资源实体，用资源换标识符 URI 标志数据，并通过 Http 协议的简单动作（GET/ POST/ PUT/ DELETE）对资源进行操作。REST API 以其灵活易用性在 SDN 接口设计中得到了广泛的应用[33]，它接口极大地化简了控制接口，提供了便捷的网络管理方式。Floodlight 控制器已经具备了 RESTful 风格的应用接口，可以对网络资源(设备，拓扑和流量等)进行简单的监测，网络拓扑，交换机，流表和组表等信息都可以通过该接口进行查询，并且还可以通过该接口改变控制器的部分行为，如开启/禁用模块，设置访问控制列表 ACL，改变寻路方式等诸多功能。REST API 使用 Restlets 库实现，新增加功能类需要实现 RestletRoutable 接口完成 URL 的映射，然后定义若干个 Resource 资源类来调用控制器模块的接口，执行对应的动作。

应用通过 HTTP 请求向控制器特定 URL 发送请求内容，接口映射路径并完成对应动作后，将执行结果返回，结果通常为 Json 格式的字符串。图 3.6 展示了通过 REST API 查询交换机信息的过程。

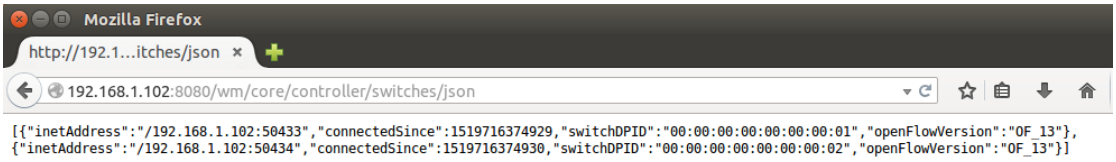

图 3.6 REST API 查询网络信息

通过浏览器输入查询 URL，提交后返回了 2 个交换机的信息。控制器接收到特定 URL 的请求后，对应的处理资源类内会有如@Get、@Put 等注解对实际的处理函数进行映射，并对请求内容进行解析，然后调用控制器的具体功能实现

URL 的请求。

（7）通讯模块

通讯模块是对交换机管理模块和底层通讯模型概括出的模块，交换机管理模块中会维护所有交换机的实例，控制器会和每一个交换机建立 OpenFlow 通道，通道由 socket 链接，所有数据层 OpenFlow 消息的传递都由模块中的链接进行传递。模块还能维护交换机链接信息，当网络增加新交换机，交换机端口状态出现变化或出现链接丢失事件的时候会及时更新模块信息。模块底层使用 Netty 框架，Netty 是一款高效的异步通讯框架，使用的是一种消息驱动的通讯模式。OpenFlow 通道的链接使用 Netty 进行维护，链接中出现对应的读、写事件后线程会通知控制器对应的消息处理函数，随后消息处理函数会根据 OpenFlow 消息的类型提交给响应的模块。

（8）线程池模块

线程池模块动态管理着控制器内线程的使用，数据采集模块中的线程都由线程池进行分配，使用线程池可以很好地减少线程创建和销毁的开销，提高系统的效率。

Floodlight 控制器具备了基本的 SDN 控制器功能，但却没有提供更多的 QoS 服务功能，开发人用可以根据实际需要对控制器功能进行定制开发。

3.4.2 控制器概要设计

前面介绍了原型控制器 Floodlight 的基本架构和功能，为了实现文中所述流量管理策略，需要在原有控制器的基础上进行开发，增加新的功能。总结前面需求分析的结果，得出系统整体功能结构如图 3.7 所示，接下来将分模块介绍各部分功能。

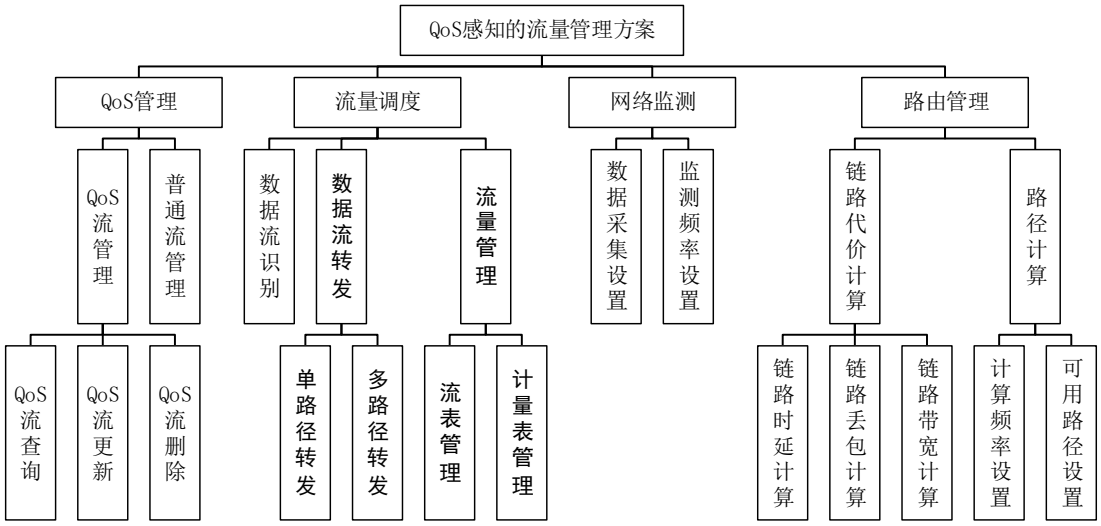

图 3.7 控制器功能结构图

（1）控制器-应用交互接口的设计

图 3.8 为 Floodlight 控制器与应用程序数据交互的时序图。控制器需要具备一个灵活的应用接口以配置 QoS 流信息，而控制器使用 REST API 形式接口设计具备了灵活和特点，因而文中沿用了原有控制器接口模型，使用 REST API 实现 QoS 流配置的接口。配置流程为，应用通过接口将 QoS 流申请信息发送到控制器，接口需要对申请信息进行解析，然后通知控制器 QoS 管理模块，模块完成 QoS 流审核和更新并将处理结果返回。

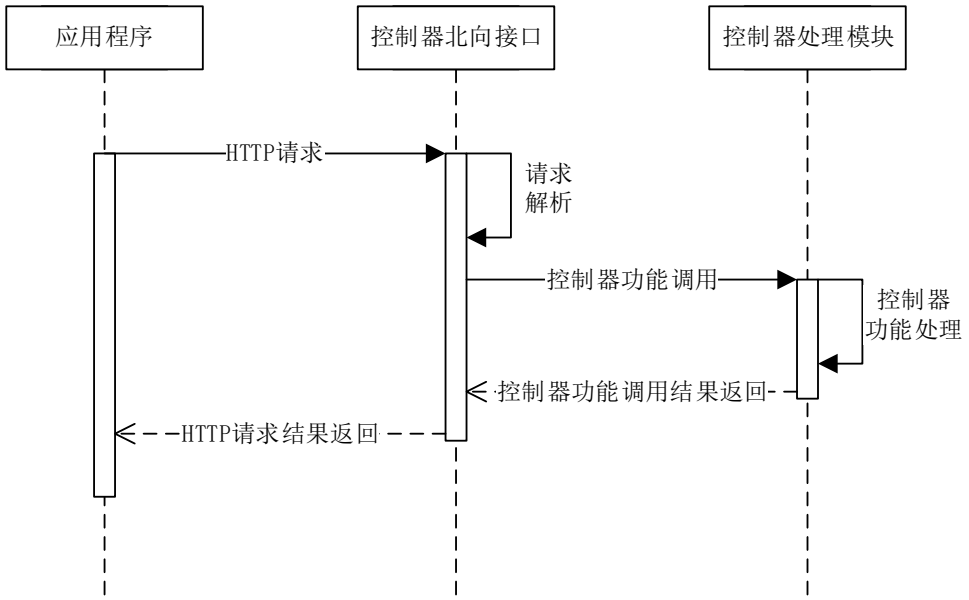

图 3.8　控制器–应用接口数据交互时序图

（2）网络监测功能的设计

控制器需要较为精确的数据平面设备信息为细粒度的 QoS 数据流调度提供支撑，控制器原有的数据采集模块功能较为简单，数据采集的时序图如图 3.9 所示，因而需要对模块进行改造，实现完整的网络数据监测功能。控制器使用的是主动测量模型，主动测量需要控制器制定数据采集，会产生额外的探测流量，因而也要注意控制数据采集的代价，避免控制通道成为系统的瓶颈。被动测量模型如 NetFlow 和 SFlow 在交换机端口进行数据采样，将收集到的信息发送给收集者。

OpenFlow 消息在控制通道的传输具有一定延迟，主动进行数据采集的方法并不能避免控制通道带来的传输延迟。同时数据采集会产生额外的数据，给控制器增加负担，需要对流量测量的准确性和控制消息的代价进行权衡，因而采集频率不宜设置过高，频繁的数据采集也使控制器短时间内解析大量数据，导致数据的错误率提高。另外文中控制器需要对网络全局信息进行监控，同时还要维护全节点的路由表，需要消耗较大的计算和存储资源，因而需要适对控制器的路由模型进行调整。该模块中主要需要对流表和交换机端口的信息进行监测。

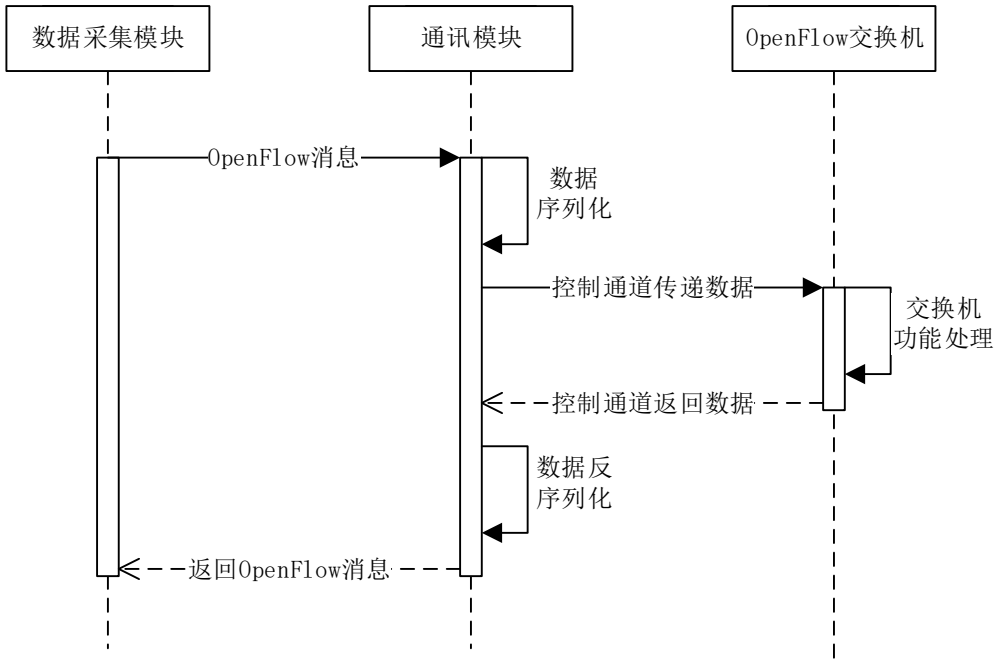

图 3.9　控制器数据采集时序图

（3）QoS 管理模块

为了有效的 QoS 策略，需要增加专用模块对 QoS 流的信息进行管理。一方面用于管理应用接口设置的 QoS 流信息，另一方面在路由时提供 QoS 的查询服务。

（4）转发模块

控制器原有转发模块实现了完整的数据流的转发功能，包括 packetIn 消息的处理，路径的查询和流表的生成和下发等工作。在此基础上转发模块需要实现数据流的识别，不同类别数据里流的转发和数据流的调度功能。数据流的识别通过解析 packetIn 消息提取原始报文的首部字段并向 QoS 管理模块发起查询，可以得知数据流类型，进而根据数据流类型向路由模块查询可用路径。数据流转发时，若数据流为 QoS 流，按照查询到的最优路径进行转发，若数据流为普通流，按找寻到的负载最低的路径进行转发，普通流的速率超过了一定的速率，模块会启用多条路径进行转发。

（5）路由模块

路由模块中将流量划分为两类：QoS 流和普通流。QoS 流需要预先通过 REST-API 向控制器进行配置，没有进行配置过的数据流皆视为普通流。QoS 流又分为时延敏感流，丢包敏感流和带宽敏感流，每种数据流需要根据其服务质量需求的类型。同时由于该方法实现的是弹性的 QoS 模型，没有进行严格的网络资源分配策略，为了减少普通流给 QoS 流带来的影响，需要对普通流进行一定的调控。

3.5 本章小结

本章对系统的整体需求进行分析并对需要实现的功能模块进行概要设计，首先介绍了所述流量调度方案的总体目标，然后对系统的功能性需求和非功能性需求进行了分析。概要设计首先对 Floodlight 控制器进行介绍，讲述了控制器的主要模块和功能，然后结合流量调度方案的需求，对控制器需要增加的功能进行设计，需要增加 QoS 流配置的应用接口，增加 QoS 模块，增强网络监测模块的功能，改动转发模块支持不同流的转发，根据流量调度目标重新设计路由方式，在多个方面的功能进行设计以实现所述流量调度策略。

第 4 章 流量调度方案详细设计与实现

文中实现的 QoS 感知的路由，使应用程序能通过 SDN 北向接口配置 QoS 流，使网络能对应用业务的服务质量需求进行感知，制定转发策略考虑 QoS 参数，使网络行为随着流量的种类而改变，可以为不同的需求的业务流量针对性地制定合适的路由方式和流量调度策略。

4.1 应用接口

REST API 形式的应用接口提供了丰富而简洁的网络信息呈现，文中应用接口按照原有控制器的北向接口框架，使用 REST API 形式的接口设置。文中在该模块中添加了 QoS 流的配制接口，使网络可以获取到上层应用业务 QoS 需求参数，如图 4.1 所示。配制内容包含 QoS 数据流的 ToS 字段设置，延迟标志，带宽标志，丢包标志等信息。QoS 流的配制信息经过解析处理后会提交给 QoS 管理模块进行处理。

REST API 使用 Restlet 库实现，配置 QoS 流的资源类为 QoSFlowResource，资源类需要实现 RestLetRoutable 接口，接口中定义了 basePath()和 getREstLet()两个方法，basePath()函数定义了接口的 URL 地址，getREstLet()方法则将 URL 请求映射到具体的处理资源类。配置 QoS 流的 UR 设置为：controller_ip:8080/wm/qos/flow/，QoSFlowResource 类中定义了 addQoSFlow()，updateQoSFlow()，deleteQoSFlow()以及 retrieveQoSFlows()四个方法对 QoS 流进行配置，方法支持 GET、POST 和 DELETE 方式的 HTTP 申请。关系类图如图 4.1 所示，数据发送的格式如图 4.2 所示。

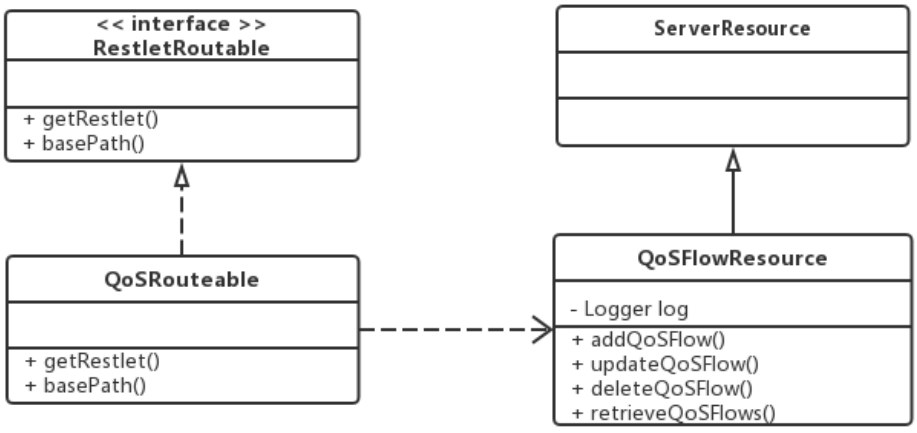

图 4.1 配置 QoS 流的 REST 接口关系类图

```
{
        "Application": "TestApp",
        "QoSType": "delay",
        "ToS": "00001100",
        "IP_PROTO":"IPv4",
        "IPv4_SRC":"10.0.0.101",
        "IPv4_DST":"10.0.0.102",
}
```

图 4.2 QoS 流设置信息样式图

4.2 QoS 管理模块

QoS 管理模块维护着网络 QoS 流的信息，应用程序可以通过 REST API 进行设置，表 4.1 列出了 QoS 管理模块中保存的数据流的主要信息，其中应用名、QoS 类型和 ToS 字段为必须项，用作最基本的流识别信息，协议、源地址和目的地址为可选项，用作更细粒度的调控。

表 4.1 QoS 流信息表

字段名	数据类型	备注
应用名	text	标记应用 id
QoS 类型	enum	QoS 流类型{delay, loss, bandwidth}
ToS 字段	byte	数据包的标记字段
协议	Text	数据流使用的协议
源地址	text	数据流源 IP 地址
目的地址	text	数据流目的 IP 地址

QoS 流匹配时加入了 ToS 字段，因此源地址和目的地址相同但 ToS 字段不同的数据流将视为不同的数据流，使用不同的流表进行转发。图 4.3 显示了 QoS 模块与其他模块的交互示意图，转发模块解析 PacketIn 消息后提取 ToS 字段，若字段值不为零，则向 QoS 管理模块查询流信息。QoS 流信息返回后，转发模块再根据数据流类型向路由模块查询路径。

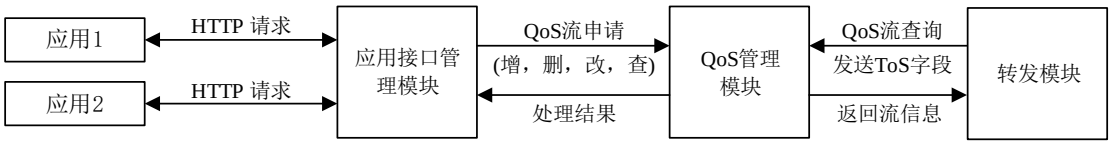

图 4.3 模块交互图

4.3 数据监控模块

数据监控模块负责对数据层设备进行监控，为转发模块和路由模块提供必要的数据流处理信息，模块间的交互示意图如图 4.4 所示。数据采集首先要设置 OpenFlow 消息类别 OFStatsType，然后构造消息体并通过控制通道下发到交换机

上，交换机执行命令后将对应的状态信息封装到 OFStatsReply 消息体并传送回控制器，控制器对消息体解析后可获得对应设备状态信息。控制器支持的常见 OpenFlow 查询消息类别和对应的状态信息如表 4.2 所示。数据监控模块主要监测的消息类型有端口信息，流表信息，和计量表信息。

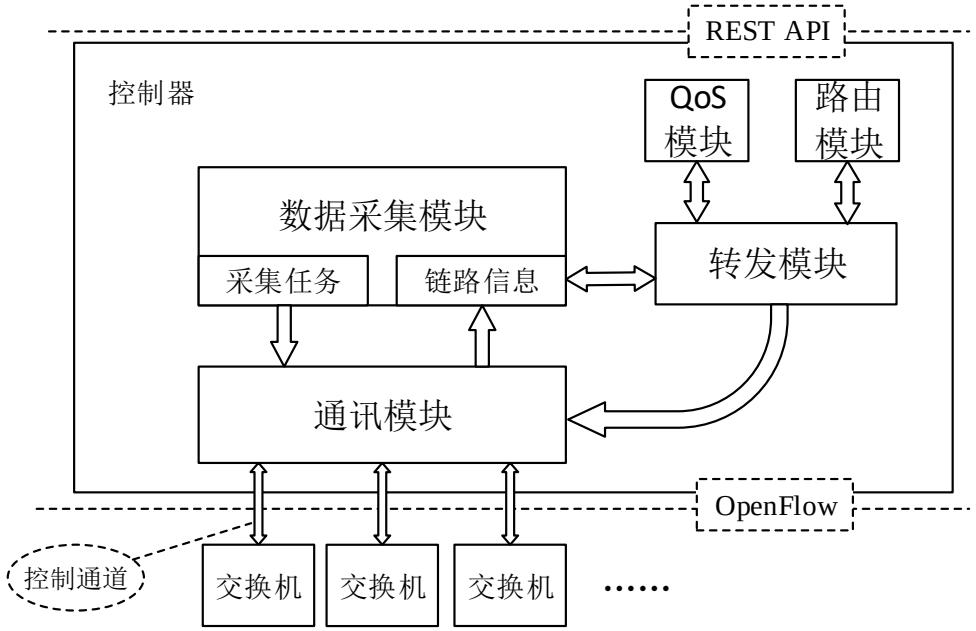

图 4.4　数据采集模块交互示意图

表 4.2　交换机端口数据表

消息类型	描述
Flow	交换机流表项信息
Table	交换机流表信息
Port	交换机端口信息
Group	交换机组表信息
Meter	交换机计量表信息
Queue	交换机端口队列信息
Desc	交换机信息

端口状态信息体 OFPortStatsReply 的主要内容如表 4.3 所示，可以看到该消息包含端口名，数据发送字节和报文数和持续时间等多项信息。模块中链路链路可用带宽值可以通过 rxBytes，txBytes 两项进行计算获取，而目前控制器还无法直接 OpenFlow 交换机还无法获取丢包值 rxDropped，txDropped 两项信息，因而需要通过端口的发送字节数和接收字节数的差值计算获得端口丢包的估计值。

表 4.3 交换机端口数据表

字段名	描述
PortNo	端口名
rxPacket	端口转接收报文数
txPacket	端口转发报文数
rxBytes	端口接收字节数
txBytes	端口转发字节数
rxDropped	接收端口丢弃的报文数
txDropped	发送端口丢弃的报文数
durationSec	持续时间（秒）
durationNSec	持续时间（毫秒）

模块使用主动测量的方法，模块将上述数据采集动作封装到一个任务实体内，并将多个实体任务交与线程池并发执行。数据采集线程的通讯方式为同步阻塞模式，单条线程在执行完“写”动作后会将自身阻塞，等待交换机返回。对于批量采集任务的执行，程序会等待所有线程全部完成或在等待超时后，一次性将所有线程的结果全部返回。数据采集的基本流程如图 4.5 示：

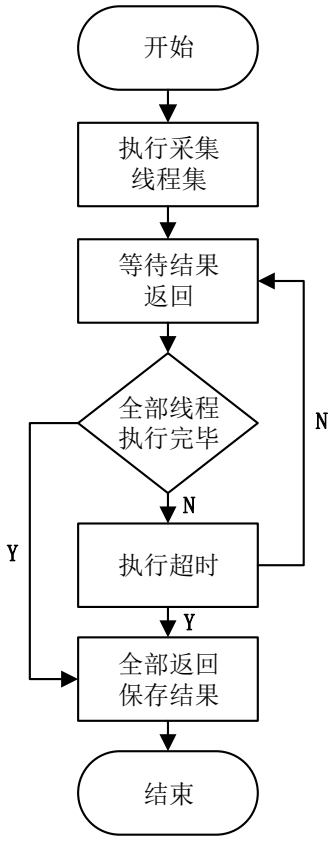

图 4.5 同步数据采集流程图

对于该种线程采集模型，如果由于交换机繁忙或故障导致单条线程出现长时间的等待，必然会影响到其他所有线程的结果返回。文中流量调度方法需要获取最新的交换机端口和链路的数据，而原有控制器方案上述情况下表现不佳。了改善这个问题，文中将线程采集模型改为异步非阻塞模式，以减少采集数据的等待时间，改动后的采集流程如图 4.6 所示。改动后的模型中，数据线程和结果的收集不在同一线程内进行，数据采集的动作后执行完毕后即停止工作，结果收集由专门线程进行处理。

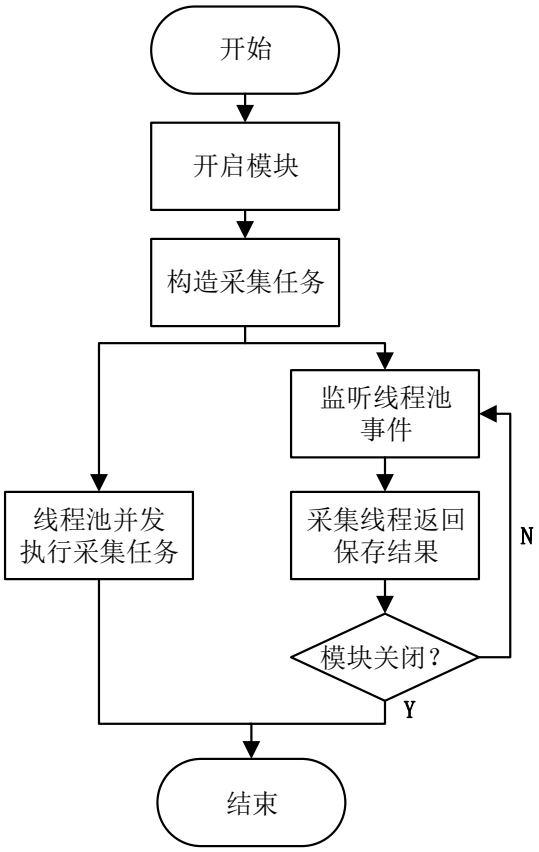

图 4.6 异步数据采集流程图

4.4 转发模块

转发模块负责管理各设备间的数据包转发，工作主要围绕 PacketIn 和 PacketOut 消息的处理和流表的生成与下发。控制器流在转发模块中识别不同数据流的报文，对 QoS 流和普通流使用不同的转发策略。

4.4.1 流的识别

SDN 使用匹配域对流进行区分，不同的匹配域的报文被归类为不同的流，通常源地址和目的地址相同的报文可以看作是同一条流的报文。文献[16]中提到，可以使用额外的字段对特定数据流的报文进行识别，例如 MPLS 中的 Traffic Class

字段，IP 报文中的 ToS 字段，IPv6 报文中的 Traffic Class 字段，端口字段等。文中使用 ToS 字段来识别数据流类型，模块解析 PacketIn 消息通过原始数据包首部字段的内容，以判断报文归处于 QoS 流或普通流。互联网工程任务组 IETF 定义的 RFC 791[34]标准中规定 IP 协议的 ToS 字段用作服务等级控制，通过优先级，延迟，吞吐量，可靠性等几个参数来制定数据包的服务等级。ToS 字段共有 8 个位，前 6 位为差分服务代码点（Differentiated Services Code Point, DSCP），后两位为显式拥塞通告（Explicit Congestion Notification，ECN），如图 4.7 所示。

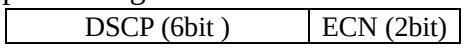

图 4.7 ToS 字段结构

由于传统网络中提供着 best-effort 式的服务，没有实现 QoS 服务，因而 ToS 字段通常都被弃用，因而使用该字段一般不会对数据包传递造成影响。

4.4.2 流的转发

转发模块中对数据的转发流程如图 4.8 所示，转发流程为：

（1）通讯模块接收到 OpenFlow 消息，调用转发模块的 receive()函数，转发模块判定其为 PacketIn 消息后，首先查看路由模块对数据流执行的动作：转发（Forward），多播（Multicast），丢弃（Drop）或不执行操作（None），随后使用 PacketIn 消息处理函数 processPacketInMessage()进行处理。

（2）若执行的操作为转发，则为数据流生成唯一的 cookie，并对数据包首部信息进行解析。

（3）若报文使用 IP 协议且报文 ToS 字段值为 0，则判定数据流为普通流，向路由模块发起普通流寻路申请；若报文使用 IP 协议且 ToS 字段不为 0，则向 QoS 管理模块查询数据流类型，根据 QoS 类型信息向路由模块发起 QoS 寻路申请。

（4）根据路由模块返回的路径信息生成流表信息，并下发到路径上的所有交换机上，同时将报文封装成 PacketOut 数据包，发送到源交换机。

控制器使用构造者模式生成 OpenFlow 消息，开发人员可以便捷的生成特定格式和内容的 OpenFlow 消息。如，使用 sw.getOFFactory().buildFlowAdd()函数得到流表构造器，构造器支持的动作如表 4.6 所示，调用对应的函数并提供参数就可以生成流表的特定内容。

匹配域的构造方式和流表相似，都是先通过模块获取对应的构造器，再调用构造器的设置函数，并添加对应的参数。该种方式可以使开发者避开 OpenFlow 协议复杂的细节，专注于需要实现的部分，通过类似部件装配的形式完成按步骤有条理地构建所需内容，使得编码更加高效。

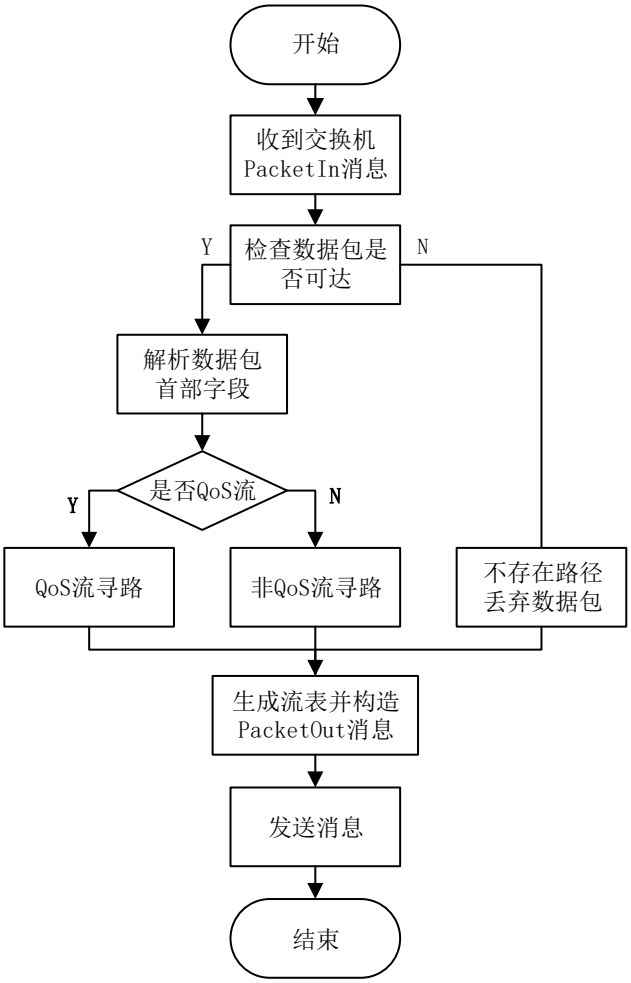

图 4.8 转发模块流程图

表 4.6 流表构造器支持的动作

函数	参数	描述
setTableId()	table_id	设置流表的标号
setMatch()	match_field	设置数据流的匹配域
setPriority()	priotity	设置流表的优先级字段
setActions()	actionSet	设置流表的动作域
setInstructions()	instructions	设置流表的指令域

4.5 路由模块

路由算法需要考虑到各条路径的代价，综合各参数选出最佳路径。

4.5.1 Floodlight 的路由策略

Floodlight 控制器中默认使用 best-effort 方式的单路径路由方案。Floodlight 采用的路由策略，没有考虑数据流 QoS 需求，并且不能很好地利用网络带宽。

根据路径的时机可以将选路模式分为两种：

（1）普通选路方式：该种方式在控制器接收到交换机发送的 PacketIn 消息后开始为数据流计算路径，因此新数据流的处理会受到控制器计算能力的影响，在控制器繁忙时数据流的处理能力下降，处理时间会明显增多。

（2）快速选路方式：快速选路方式预先计算路径，数据流到达时直接读取计算结果，因而可以有效地避免路径计算的等待时间。由于控制器无法对数据流的出现进行预知，控制器需要提前计算网络所有节点之间的路径，以保证数据流到来时可以直接从计算结果中查询路径。同时为了保证可用路径集中的路径数据的准确性和有效性，该方法需要间隔一段时间进行络路径集的重计算，因而需要控制器具备较强的计算能力。

通常来说，网络结构较为简单时可以统一采用慢选路方式，对于中型和大型网络则需要进一步衡量路由计算的时间代价。目前商用计算机 CPU 处理能力较强，而网络处理能力往往受限，网络更成为系统的瓶颈，同时处理延迟的增加又会极大地降低用户体验。因而为了提高 QoS 流的服务质量，减少数据流寻路的等待时间，路由模块采用的是快速选路方式。

使用 Dijkstra 算法[35]寻找任意两个节点的最短路径，使用 Yen[36]算法计算网络内各节点间的最短路径集合，下面对两种算法进行介绍：

（1）Dijkstra 算法

模块中通过 Dijkstra 算法来计算各节点的单源最短路径，Dijkstra 算法采用贪心策略，路径搜索时对链路设置特定的权值来进行约束选路，算法可以下列步骤进行描述：

①对网络内的所有节点标记为未访问，并放入到未访问节点集中。对链路权值进行初始化：对于任意两个节点，若节点相邻则链路权值为链路的代价；若两节点不相邻，则链路代价设置为最大权值。

将源节点设置为当前节点，从未访问节点集中删除。

③对于当前节点，查看未访问节点集中所有相邻节点，选出相邻节点中链路权值最小的节点，记录当前节点到选中节点的距离，更新路径集。

④将第②步选中的节点标记为已访问，将其从未访问节点集中删除。

⑤若目的节点被标记为已访问，或者未访问节点集中的所有节点的权值为最大权值，算法停止。

⑥从未访问节点集中选择一个权值最小的节点，将其设置为当前节点，返回步骤③。

（2）Yen 算法

模块中使用 Yen 算法计算源和目的之间的多条最优路径， Yen 算法首先使

用 Dijkstra 算法计算源节点至目的节点的最优路径，将目的节点以外的所有节点视为偏离节点，并计算偏离节点到目的节点的最优路径，使其与源节点到偏离节点的最优路径拼接即可得到候选路径。

Yen 算法计算 k 条最优路径的过程可以用以下步骤进行描述：

①使用 Dijkstra 算法计算源节点和目的节点间的 k 条路径集。

②选中路径中的一个非目的节点，记为偏离节点，使用 Dijkstra 算法计算偏离节点到目的节点的最短路径。

③将源节点到偏离节点的最短路径和步骤 4 中计算出的最短路径拼接成新路径，放到候选路径集中。

④重复上述步骤求得 k 条候选路径后，对候选路径集进行排序，得出源节点至目的节点的最优路径集。

Yen 算法从算法复杂度上来看并不是最优的，但由于其算法思路清晰，便于编码和调试，因此在实际工程应用也较常见。模块中使用 maxPathsToCompute 值来描述计算的最大可用路径数，参数默认值为 3，若需要计算更多路径，可以通过 REST API 对该值进行更改。

4.5.2 链路权值的设置

模块将根据网络监测模块到的链路数据分析和评价，链路的时延，丢包，和带宽参数可以通过下面的方法进行计算：

（1）时延和测量和计算：

网络链路时延数据的测量是通过链路层发现协议（Link Layer Discover Protocol，LLDP）来实现的，LLDP 协议是用于链路发现的协议，网络设备可以通过协议通告其他设备自身的标识和性能等信息。

交换机链路时延的具体测量方法如图 4.9 所示，可以用以下步骤描述：

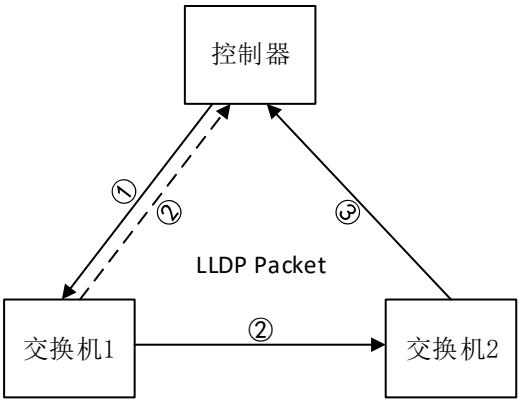

图 4.9 LLDP 协议的传输过程

①控制器向交换机 S_1 发送 LLDP 报文，要求 S_1 向所有端口发送该报文。

②交换机 S_1 接收到报文后将报文发送给控制器和交换机 S_2。

③相邻的交换机 S_2 接收到 LLDP 报文后，由于其流表没有对应的报文处理表项，因此将数据包发送回控制器。

④控制器接收到报文后，通过报文时间差值计算传输过程中的时延值。

控制器要获取多组（默认为 10 组）LLDP 时间数据后才开始链路的时延的计算，时延值的计算精确到毫秒级别。

（2）丢包测量和计算：

丢包值将根据链路两端交换机端口的发送和接收值进行计算。假设特定时间内链路一端发交换机端口发送为 Tx，另一端接收速率为 Rx，则该链路丢包率 R_{loss} 可用 4-1 式表示，如图 4.10 所示：

$$R_{loss} = \frac{R_{Tx} - R_{Rx}}{R_{Tx}} \tag{4-1}$$

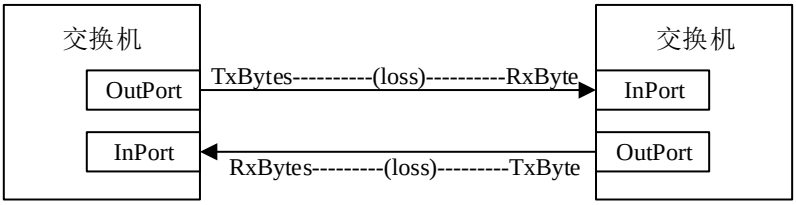

图 4. 10 端口丢包示意图

（3）交换机端口速率测量和带宽的计算：

交换机端口的发送的速记为 R_{Tx}，端口理论最大发送速率记为 R_{max}，在 t 秒内则链路两次测得端口数据包处理值分别为 Tx_1 和 Tx_2，则该时间段内的端口速率 R_{Tx} 可用公式 4-1 表示，链路利用率 U_{link} 可以用公式 4-2 表示:

$$R_{Tx} = \frac{Tx_1 - Tx_2}{t} \tag{4-2}$$

$$U_{link} = \frac{R_{Tx}}{R_{\max}} \tag{4-3}$$

$$B_{avaliable} = 1 - \frac{T_x}{T_{\max}} \tag{4-4}$$

Floodlight 使用枚举类型参数 PATH_METRIC 来描述链路的代价的类别，PATH_METRIC 可以设置为跳数（HopCount），时延（Latency），避开隧道的跳数（Hopcount_avoid_tunnle），链路利用率（Utilization）以及链路速度（LinkSpeed）五种类型。控制器支持对 PATH_METRIC 的动态修改，可以通过 REST API 对其进行设置，但同一时间内只持支持一种参数的路径计算。如，需要获取基于时延的最短路径时，将 PATH_METRIC 设置为 latency，此时路由模块计算的是基于时延的最优路径，此时将若可以 PATH_METRIC 设置为 huoCount，控制器将会计算基于跳数的最优路径集，若要基于其他值的计算，则

还需要对 PATH_METRIC 进行更改，再一次触发最优路径集的重计算。上述过程中每一次对 PATH_METRIC 的改动就会触发一次路径集的重计算，因而在网络内存在较多不同种类的数据流时需要频繁的进行更改参数和路径计算的过程，消耗大量的计算资源。

因此需要对原有控制器路由计算方式进行修改，由于文中支持三类 QoS 流的优先调度，因而路由模块需要同时支持三种 QoS 参数的寻路，加上基于跳数的路由，模块中将可以同时支持基于时延，丢包，带宽和跳数的四种参数的路由计算，具体的处理过程将在下个小节中进行介绍。

4.5.3 QoS 流路径的调度

Floolight 控制器中实际上是在拓扑管理模块的实例中实现路由的计算，但文中为了表述更加清晰，将由拓扑管理模块实现的功能放到本小节中进行讲解。

模块首先要根据 QoS 参数计算路径代价集，上小节中已经阐述了 QoS 参数的测量方式，链路权值 LinkCost 的计算分别依据 4-3 式至 4-6 式：

$$LinkCost_{hopCount} = 1 \tag{4-5}$$

$$LinkCost_{bandwidth} = Rate \tag{4-6}$$

$$LinkCost_{latency} = \begin{cases} latency, & if \quad latency < maxLinkWeight \\ maxLinkWeight, & if \quad latency \geq maxLinkWeight \end{cases} \tag{4-7}$$

$$LinkCost_{loss} = \begin{cases} loss, & if \quad loss < maxLinkWeigh \\ maxLinkWeight, & if \quad loss \geq maxLinkWeight \end{cases} \tag{4-8}$$

其中 latency，loss，Rate 分别为数据监控模块返回的实际值时延值，丢包值和链路速率值，maxLinkWeight 为链路权值的最大值。

QoS 流路径集的计算方式如图 4.11 所示。

通过 Yen 算法分别计算 4 种参数下的加权最优路径，并将计算结果分别保存到 4 个独立的 PachCache 结构体（基于时延的最优路径集 PathCache_latency，基于丢包的最优路径集 PathCache_packetLoss，基于可用带宽的最优路径集 PathCache_bandwidth，基于跳数的最优路径集 PathCache_hupCount）中，QoS 流在进行寻路时将根据其类型在不同的路径集里进行查询。

模块中使用 Dijkstra 算法和 Yen 算法实现对单源最优路径和多源最优路径计算的计算，Dijkstra 算法计算过程中对路径中各条链路的权值进行累加，逐次选出源节点至各个节点的最优路径，路径权值可用公式（4-9）表示：

$$PathCost = \sum LinkCost_{link}, \quad link \in path \tag{4-9}$$

由于路径的可用带宽由其链路的最小可用带宽决定，因而对于基于链路利用率的路径 path，其链路代价 PathCost 可以由 3-10 式计算：

$$PathCost_{\text{bandwidth}} = \min\{LinkCost_{link}\}, \quad link \in path \tag{4-10}$$

模块中 QoS 流默认按使用单路径转发，多路径传输作为一个备用方案。

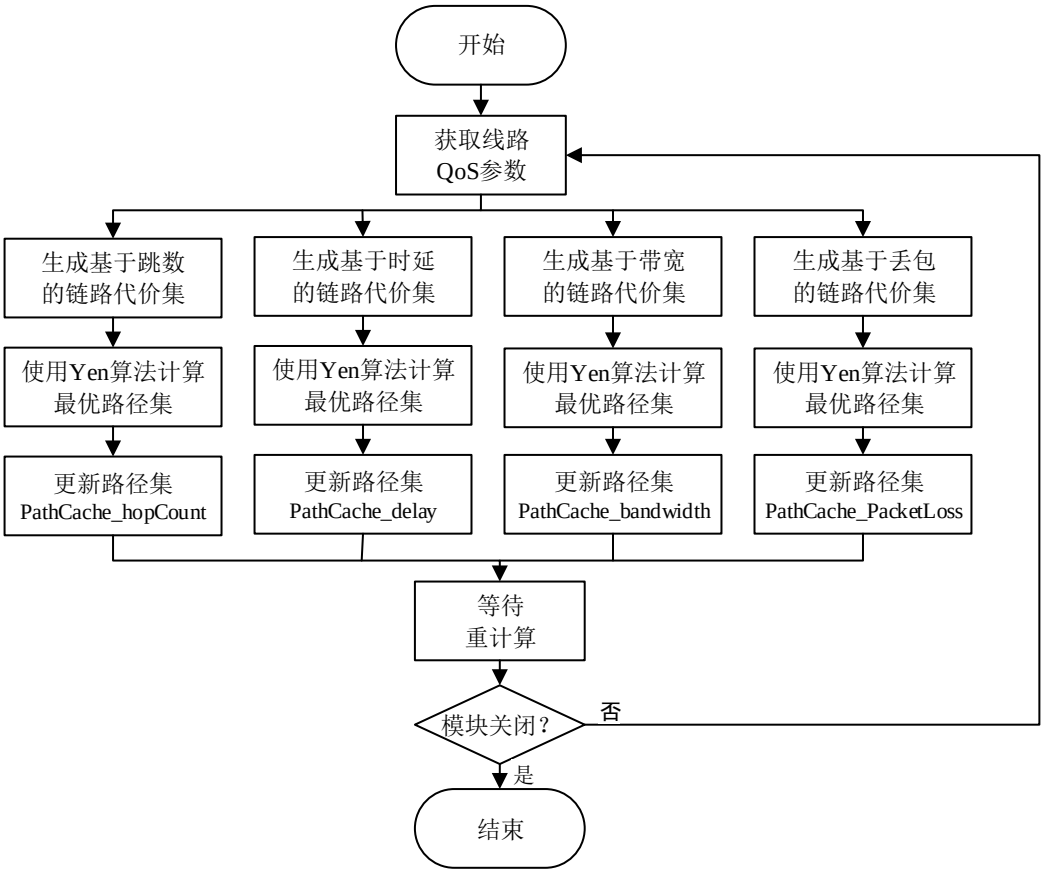

图 4.11 QoS 流路径计算流程图

路由模块会定时更新路径集信息，在拓扑发现模块监测到链路的变动时也会触发路径集的更新，控制中提供了 forceRecompute()函数对路径集进行重计算，模块中将设置计时器定时调用该函数。由于控制器维护四个路径集会消耗较多的 CPU 资源，因而路径的计算频率不宜设置过高，模块中重计算的间隔为 2 秒。

（3）QoS 流路径的查询

路径查询过程如图 4.12 所示。

转发模块在监测到数据流为 QoS 流后，向路由模块发信路径查询申请，申请需要提交源节点，源节点端口，目的节点，目的节点端口，QoS 流类型五个参数。路由模块查询路径集，将路径（集）返回，转发模块得到最优路径后再根据策略生成流表下发到路径上的交换机中。

总结来说，路由模块中维护着 4 张路由表，表中分别保存着以长度，时延，丢包，链路负载为链路代价的最优路径集，QoS 流进行寻路时直接从对应路径集中进行选取。

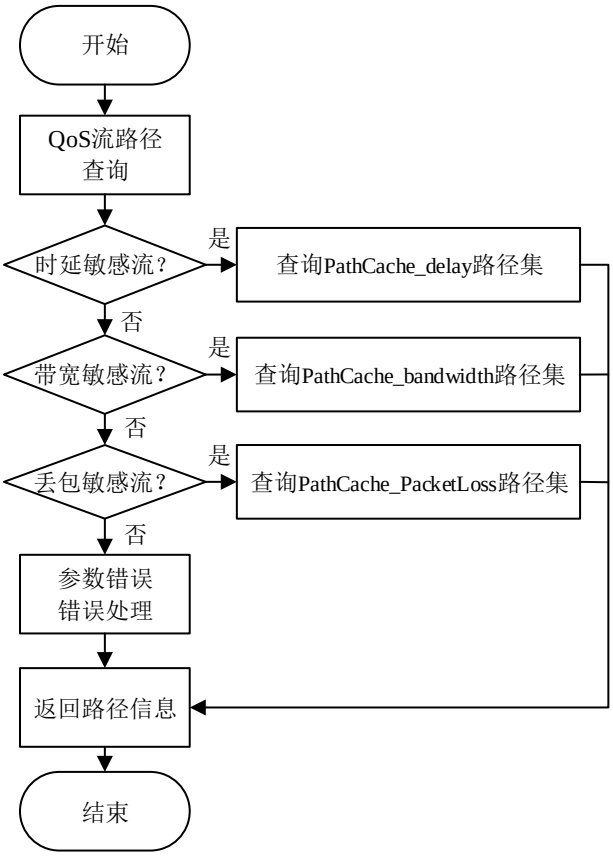

图 4.12 QoS 流路径查询流程图

4.5.4 普通流的调控

普通流的调控旨在将充分利用网络的空闲链路，尽可能的减少链路拥塞的发生，减少普通流对 QoS 流的影响。文献[37]统计数据显示，在数据中心网络链路上的大部分带宽通常被少数 elephant 流（流量大且持续时间长的流）所消耗，绝大部分的数据流数据量少，持续时间短。因此，在进行流量调度的时候对大象流进行调度的效果明显，调控代价小，文中流量调控也主要针对普通流中的 elephant 流进行。

OpenFlow 中以流为单位进行调控，可以实灵活、细粒度的流量调控。第 2 章中已经介绍了 OpenFlow 计量表 Meter 的功能和原理，文中流量调控方案就是基于 Meter 实现的，调控的基本思想为：最初普通流只按照单路径进行转发，当数据流增长 elephant 流规模后，根据的其速率大小再安排其他转发路径，具体步骤描述如下：

（1）为数据流设置若干个基准速率阈值和一个最大速率阈值；

（2）新数据流进入网络，控制器从路径集中查找最低负载的路径进行转发；

（3）监测数据流流速率，每当数据流速率超过一个基准速率阈值，检查从路径集中选取一条路径。

（4）若数据流速率超过最大速率值，丢弃超过速率的数据包。

上述过程中，由于一开始控制器并不能明确数据流是否为大象流，因而在步骤 2 中先采用单路径进行转发。在步骤 3 中监测到数据流速率超过阈值，确定了数据流为 elephant 流，此时再根据其流速的大小安排多条路径进行转发。

接下来举例说明该方法，网络内设置主机 H_1、H_2，交换机 S_1~S_5，H_1 需要将数据传送到 H_2，计量表的设置如表 4.13 所示，2 个基准速率阈值分别为 3Mbps，6Mbps，最大速率阈值为 9Mbps，网络拓扑如图 4.13 所示，整体过程描述如下：

（1）主机 H1 开始发送数据，此时链路空闲，控制器选择低路径 S_0-S_1-S_4 进行转发；

（2）当速率超过 3Mbps 后，控制器接收到交换机数据，发现该数据流为 elephant 流，随即安排第二条低负载链路 S_0-S_2-S_4，将流表下发到交换机中。速率超过 3Mbps 的这部分数据包组成了一条新的数据流，采用新的流表进行转发；

（3）当速率超过 6Mbps，控制器安排第三条路径 S_0-S_3-S_4 转发超过速率的数据包，处理过程同步骤 2。

（4）当速率超过 9Mbps，流速率超过最大限制，超过速率的部将被丢弃。

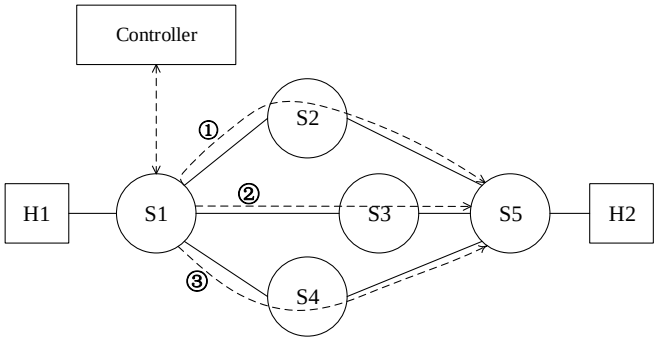

图 4.13 流量调度例图

上述方法需要使用到两级流表，流表 table0 中的流表只进行数据流速的测量，不进行数据转发，数据包在经过计量表的处理后转移到流表 table1 中，table1 根据数据包 ToS 字段对数据包进行转发，总体流程为：

（1）数据包进入交换机 pipeline 后首先经过流表 Table0 处理如图 4.14 所示，流表进行数据流匹配，执行匹配表项的动作，流表中对应表项的指令域中将数据包先交给计量表处理。

（2）计量表根据流速执行表项动作：若数据流速率低于 3Mb/s 时不执行动作；若速率超过 3Mbps，将超过速率的数据包 DSCP 字段从 0 修改为 1；若速率超过 6Mbps，超过速率的数据包 DSCP 字段将修改为 2；若速率超过 9Mbp/s，超过速率的数据包会被抛。然后未被抛弃的数据包会被转移到流表 Table1 中进行处理。

（3）流表 Table1 一开始只包含标号为 1 的表项，数据包全部由端口 port1 转发。若数据流速率超过 3Mbps，则 Table1 中会出现 dscp 值为 1 的数据包，但

此时无匹配表项，交换机会将数据包封装到 PacketIn 消息发送到控制器，控制器进行选路后下发标号为 2 的流表项，此时 dscp 值为 1 的数据包由端口 port2 进行转发；数据流速率超过 6Mbps 时同理，由端口 port3 进行转发。

若 elephant 流的传输速率 R 可预知，且存在 n 条可用路径，那么将计量表中的速率阈值设置为 R/n 时，可将数据流均匀转发到各条路径上。

该方法通过计量表监测数据流速率，在速率超过阈值后交换机将信息发送到控制器，通知控制器 elephant 流的存在，然后控制器可以迅速将备用路径下发到交换机。通过设置流量阈值，在数据流达到一定速率后再分配备用路径，选路方式更加灵活，同时可以减少对交换机流表的占用。将原始数据流分割成多条支流进行转发，可以避免单条链路上存在过大的数据流，同时能限制普通流的最大速率，避免带宽的滥用。

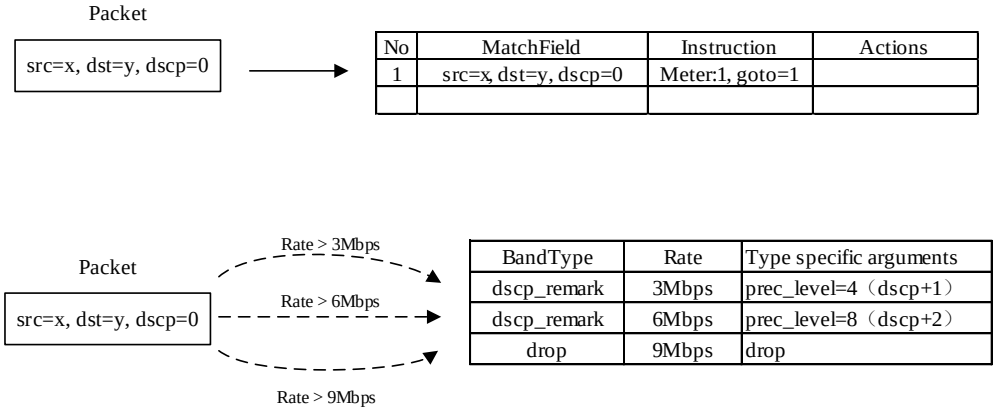

图 4. 14 普通流处理流程示意图

模块中采用快选路方法在流量突发时有可能导致路径集失效，对流量剧增时控制器在短时间内不能更新链路信息，因而路径集内的信息不能准确的反应链路实际状况，此时控制器若使用失效路径进行转发有可能会使多条数据流集中转发到同一条路径上，导致网络拥塞。为了应对这个问题，控制器中引入了版本号和临时流集合用于辅助判断控制器是否出现突发，以及判断路径集是否失效。

- 版本号（PathCacheSerialNum）：用于判断路径集是否已经更新，转发模块和路由模块中分别保存着一个路径集的版本号值。路由模块在每次路径重计算后更新其版本号，转发模块在发起寻路时对比本模块版本号和路由模块中的版本号是否相同，若不同则表示路径集已经更新。
- 临时流集合（FlowCache）：用于辅助判断是否出现流量洪峰，记录新数据流的源地址，目的地址和 ToS 字段值信息，若转发模块检测到源地址和目的地址相同但 ToS 值不同的 PacketIn 报文时，则表明交换机上数据流速率已经超过计量表值，数据流为大象流。flowCache 保存的数据流信息会在每次监测到路径集更新时清空。

若控制器检测到网络内出现 elephant 流并且路径集已失效，则转发模块会向路由模块查询数据流的 n 条可用路径，并根据 dscp 值将新划分的子数据流安排到路径集中的序号为 dscp % n 的路径上，尽可能地在流量突发时将流量转发到多条可用路径上。

总结普通流的调度方法，通过计量表将大象流进行切分成多条支流，在流量平缓变化时期支流使用最空闲的链路进行转发，在流量突发时期支流使用则按序依次使用路径集中的可用路径进行转发。

4.6 本章小结

本章详细阐述了系统的详细设计和实现，详细讲述了应用接口模块，QoS 管理模块，转发模块，路由模块的结构和功能，以及各模块之间的交互协作方式。系统通过 REST API 配置 QoS 流，在转发模块中识别 QoS 流和普通流，路由模块对两类数据流采取不同的寻路方案，对 QoS 流采取弹性的 QoS 寻路服务，使用 Dijkstra 算法和 Yen 算法计算最优路径，使用计量表 Meter 配合流表对普通流进行调度，一方面保障 QoS 流的传输，另一面提高网络整体链路利用率。

第 5 章 系统测试和验证

前面已经对流量调度方案的设计和实现进行了详细的介绍，本章将对该系统运行环境进行介绍，并对流量调度方案进行测试，以验证该方案的效果。本章首先介绍测试的环境，对 SDN 仿真软件 mininet 和软件交换机 ofsoftswitch 进行了说明，然后对控制器的功能进行测试，使用流量生成工具模拟真实流量，对 QoS 感知的流量调度策略进行测试，以验证系统是否达到预期效果。

5.1 硬件环境

实验中控制器和网络转发设备运行在不同系统上，Floodlight 控制器的运行环境如表 5.1 所示，SDN 虚拟机运行环境如表 5.2 所示。

表 5.1 控制器系统环境表

资源名称	描述
系统	Windows 10 专业版 64 位
处理器数量	4 核
处理器主频	2.40 GHz
内存	12 GB

表 5.2 SDN 虚拟机运行环境表

资源名称	描述
系统	Ubuntu 14.04, 64 位
处理器数量	4 核
处理器主频	2.40 GHz
内存	4 GB

5.2 软件环境

（1）网络环境仿真软件 Mininet

Mininet[38]是由斯坦福大学 Nick MeKeown 带领的小组研发的一款轻量级 SDN 仿真软件，通过 Mininet 可以方便地搭建网络测试环境，完成网络的部署。Minnet 提供 python API，开发人员能灵活地实现自定义的网络拓扑，支持大规模的网络模拟。Mininet 还具有良好的兼容性，它支持 OpenFlow 协议，常见的控制器和软件交换机都能集成到 Mininet 中，代码开发可以实现与真实网络的无缝切换。

Mininet 网络内的元素分配节点 Node 和链路 Link 两种类型，节点 Node 又可分为虚拟主机节点 Host，虚拟交换机节点 Switch 和虚拟控制器节点 Controller，对于每一种节点都设置有对应的函数进行配置和操作。例如虚拟主机 Host 可设置为 CPULimitedHost，对主机的 CPU 使用率进行限制。而链路类 Link 则加入功能强大的实现类 TCLink，将链路类型设置为 TCLink 可以设置链路的时延，带宽，丢包和队列的具体信息，这使得网络开发者可以通过代码和参数的设置模拟真实网络的链路场景，进行大规模实验。

（2）软件交换机

软件交换机具有和物理交换机相同的功能，通常在数据处理的性能上略低于物理交换机。但由软件交换机作为应用软件，不受物理设备和空间的限制，具有使用灵活和接近零成本的特点，因此在 SDN 的研究和实验中得到了广泛的使用。常见的软件交换机有 Open vSwitch（OVS）[39]和 Ofsoftswitch13[40]等。软件交换机为大规模的网络测试提供的极大的便捷，SDN 研究人员可以对软件交换机的数量和性能进行制定，模拟真实网络场景，并对交换机其运行状态进行实时记录。

实验中采用的 Ofsoftswitch13 交换机是一款运行在内核态的软件交换机，并对 OpenFlow 1.3 版本具有良好的支持。实验中使用 ToS 字段对不同的数据流进行区分，因而摒弃了 RFC 791 中设定 IP 协议服务等级的方式，同时对 Ofsoftswitch13 交换机的源码进行了修改。控制将 dscp_remark 设置动作“pre_level=n”的行为改动为使 dscp 字段值直接增加 n（n 为小于 256 的自然数），例如 dscp 原始值为 0，执行 pre_level=1 动作后 dscp 字段值变为 1。改动前后的代码对比如图 5.1 所示，其中 ipv4 指向数据报文内容，ipv4->ip_tos 指向 ToS 字段值。

交换机原始处理方式：

```
struct ip_header *ipv4 = (*pkt)->handle_std->proto->ipv4;
if (((old_drop == 0x8) && (band_header->prec_level <= 2))
    || ((old_drop == 0x10) && (band_header->prec_level <= 1))) {
    uint8_t new_drop = old_drop + (band_header->prec_level << 3);
    uint8_t new_tos = new_drop | (ipv4->ip_tos & 0xE3);
    uint16_t old_val = htons((ipv4->ip_ihl_ver << 8) + ipv4->ip_tos);
    uint16_t new_val = htons((ipv4->ip_ihl_ver << 8) + new_tos);
    ipv4->ip_csum = recalc_csum16(ipv4->ip_csum, old_val, new_val);
    ipv4->ip_tos = new_tos;
}
```

改动后的处理方式：

```
struct ip_header *ipv4 = (*pkt)->handle_std->proto->ipv4;
uint8_t new_tos = ipv4->ip_tos + band_header->prec_level;
uint16_t old_val = htons((ipv4->ip_ihl_ver << 8) + ipv4->ip_tos);
uint16_t new_val = htons((ipv4->ip_ihl_ver << 8) + new_tos);
ipv4->ip_csum = recalc_csum16(ipv4->ip_csum, old_val, new_val);
ipv4->ip_tos = new_tos;
```

图 5.1 Ofsoftswitch13 交换机代码改动对比图

（3）流量生成工具 Iperf

Iperf 是一软功能强大的网络性能测试工具，可以生成 TCP 和 UDP 流量对网络的吞吐量，时延抖动，丢包等表现进行测试。Iperf 支持多种参数的设置，表 5.3 中为工具常用的参数介绍。Iperf 工具还可以配合系统脚本进行使用，能实现智能而便捷的网络测试。

表 5.3 Iperf 常见参数表

参数	描述
-s	设置主机为服务端模式
-c	设置主机为客户端模式
-t	设置数据流的持续时间
-u	设置数据流为 UDP 数据流
-b	设置发送带宽
-S	设置发送数据的 ToS 字段值

图 5.2 为网络主机吞吐量测试的一个示例，图中主机 H_1 向主机 H_2 发送速率为 100Mb/s，持续时间为 10 秒的 UDP 数据流，传输结束后两端主机都可以看到传输的数据信息。

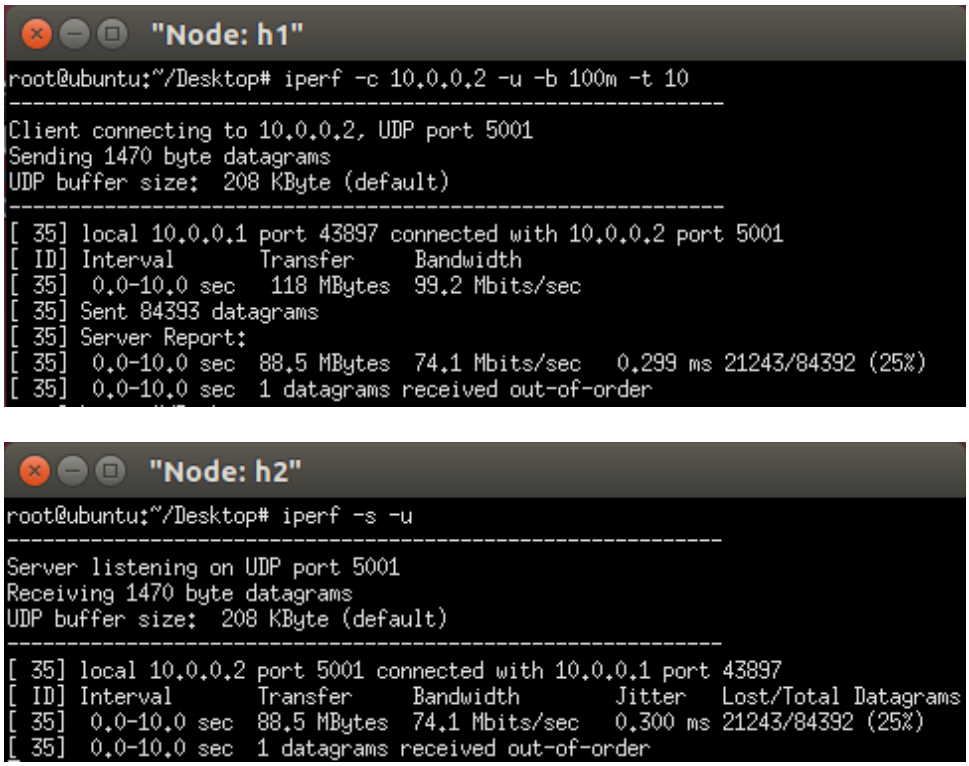

图 5.2 Iperf 测试示例图

系统测试使用 Mininet 搭建网络环境，软件交换机使用 Ofsoftswitch13，使用 OpenFlow1.3 作为通讯标准，使用 Iperf 作为流量生成工具，相关软件版本表 5.4 所示：

表 5.4 测试系统软件工具

工具类型	描述
网络模拟器	Mininet 2.3.0d1
软件交换机	Ofsoftswitch13
流量生成工具	Iperf 2.0.5

5.3 系统测试和结果分析

5.3.1 网络环境测试

首先测试控制器和各网络设备是否正常运作。实验拓扑选自 Abilene 骨干网[41]，拓扑图中共有 11 个交换机节点（S_1~S_{11}），每个交换机链接一个主机（H_1~H_{11}），链路共有 14 条，链路设置了不同的时延值和丢包值，拓扑图和链路数据如图 5.3 和表 5.5 所示。

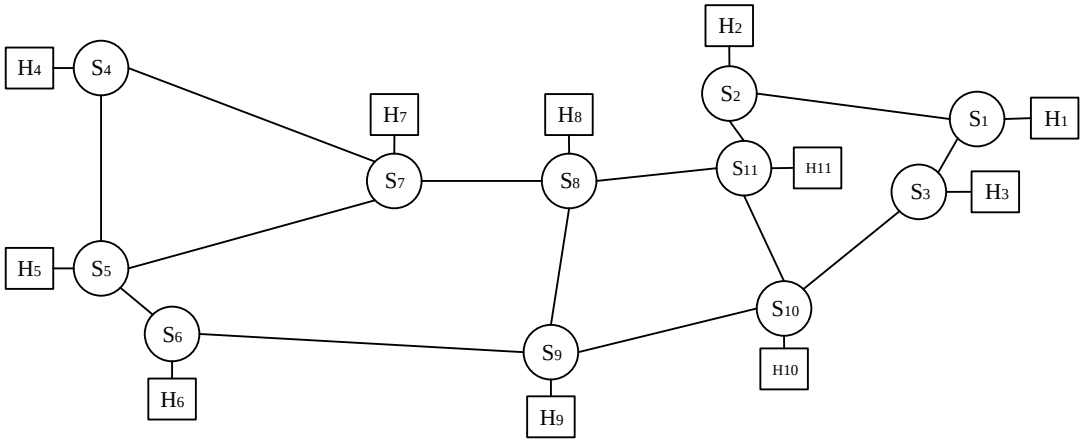

图 5.3 实验网络拓扑图

表 5.5 测试链路数据表

链路	带宽(Mbps)	时延(ms)	丢包率
Link(S1,S2)	10	10	0.3
Link(S1,S3)	10	2	0
Link(S2,S11)	10	2	0.2
Link(S3,S10)	10	8	0.1
Link(S4,S5)	10	8	0
Link(S4,S7)	10	13	0.1
Link(S5,S6)	10	4	0.3
Link(S5,S7)	10	13	0.1
Link(S6,S9)	10	16	0.5
Link(S7,S8)	10	5	0
Link(S8,S9)	10	8	0
Link(S8,S11)	10	5	0.3
Link(S9,S10)	10	10	0.1
Link(S10,S11)	10	5	0.1

实验首先在 Mininet 内执行 Pingall 指令（主机之间相互发送 Ping 报文），

测试控制器是否能正常下发流表，以及各交换机链接是否正常。从图 5.4 可以看出所有主机链接正常，Ping 报文没有丢失，同时控制器转发模块输出链路信息（如图 5.5 所示），可以看到数据流路径的长度，丢包率，链路利用率等信息。

从图中信息可以看出，控制器可以和各交换机正常运作，接下来将对数据流调度进行测试。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Results: 0% dropped (110/110 received)
```

图 5.4 Pingall 指令测试结果图

```
INFO  [n.f.f.Forwarding] [数据流]  HOPCOUNT RouteId [src=00:00:00:00:00:00:00:08 dst=00:00:00:00:00:00:0
INFO  [n.f.f.Forwarding] [路径1] 长度: 4 丢包率: 6% 链路利用率: 0% 时延: 32ms
INFO  [n.f.f.Forwarding] [路径2] 长度: 3 丢包率: 10% 链路利用率: 0% 时延: 28ms
INFO  [n.f.f.Forwarding] [路径3] 长度: 5 丢包率: 5% 链路利用率: 0% 时延: 41ms
INFO  [n.f.f.Forwarding] [路径4] 长度: 5 丢包率: 15% 链路利用率: 0% 时延: 48ms
INFO  [n.f.f.Forwarding] [路径5] 长度: 8 丢包率: 14% 链路利用率: 0% 时延: 69ms
INFO  [n.f.f.Forwarding] [数据流]  HOPCOUNT RouteId [src=00:00:00:00:00:00:00:06 dst=00:00:00:00:00:00:0
INFO  [n.f.f.Forwarding] [路径1] 长度: 2 丢包率: 26% 链路利用率: 0% 时延: 22ms
INFO  [n.f.f.Forwarding] [路径2] 长度: 5 丢包率: 5% 链路利用率: 0% 时延: 53ms
INFO  [n.f.f.Forwarding] [路径3] 长度: 6 丢包率: 4% 链路利用率: 0% 时延: 65ms
INFO  [n.f.f.Forwarding] [路径4] 长度: 7 丢包率: 10% 链路利用率: 0% 时延: 73ms
INFO  [n.f.f.Forwarding] [路径5] 长度: 8 丢包率: 8% 链路利用率: 0% 时延: 85ms
INFO  [n.f.f.Forwarding] [数据流]  HOPCOUNT RouteId [src=00:00:00:00:00:00:00:09 dst=00:00:00:00:00:00:0
INFO  [n.f.f.Forwarding] [路径1] 长度: 2 丢包率: 20% 链路利用率: 0% 时延: 21ms
INFO  [n.f.f.Forwarding] [路径2] 长度: 5 丢包率: 5% 链路利用率: 0% 时延: 43ms
INFO  [n.f.f.Forwarding] [路径3] 长度: 6 丢包率: 4% 链路利用率: 0% 时延: 52ms
INFO  [n.f.f.Forwarding] [路径4] 长度: 7 丢包率: 10% 链路利用率: 0% 时延: 57ms
INFO  [n.f.f.Forwarding] [路径5] 长度: 8 丢包率: 8% 链路利用率: 0% 时延: 66ms
```

图 5.5 控制器转发模块输出信息图

3.5.2 QoS 流路由测试

实验对 QoS 的调度进行测试，验证控制器是否能为不同数据流选择合适的转发路径，该实验通过两个部分进行验证。

（1）基本 QoS 流路由的测试

该部分验证 QoS 感知路由的基本功能。实验生成 3 条从主机 H_1 到主机 H_{11} 的数据流（记为 $Flow_1$~$Flow_3$，见表 5.6），为了从主机端获取更准确的时延和丢包信息，该实验中数据流均设置为 UDP 流，实验将多次重复发送数据流，记录数据流传输的链路信息。

表 5.6 实验数据流设置表

数据流	数据流类型	QoS 类型	发送速率	持续时间
$Flow_1$	UDP	时延敏感	1Mbps	5s
$Flow_2$	UDP	丢包敏感	2Mbps	5s
$Flow_3$	UDP	带宽敏感	5Mbps	10s

在应用层使用 CURL 工具生成 HTTP 请求，通过控制器 REST API 设置 $Flow_1$ 为时延敏感流，$Flow_2$ 设置为丢包敏感流，$Flow_3$ 设置为带宽敏感流，配置过程如图 5.6 所示。

```
sdn@ubuntu:~/Desktop$ curl -X POST 172.30.75.29:8080/wm/qos/flow -d "{"Application":"app1","QoSType":"
latency","ToS":"00100000","IP_PROTO":"IPV4","IP_SRC":"10.0.0.1"}"
{"status" : "OK."}
sdn@ubuntu:~/Desktop$ curl -X POST 172.30.75.29:8080/wm/qos/flow -d "{"Application":"app2","QoSType":"
loss","ToS":"01000000","IP_PROTO":"IPV4","IP_SRC":"10.0.0.1"}"
{"status" : "OK."}
sdn@ubuntu:~/Desktop$ curl -X POST 172.30.75.29:8080/wm/qos/flow -d "{"Application":"app3","QoSType":"
bandwidth","ToS":"10000000","IP_PROTO":"IPV4","IP_SRC":"10.0.0.1"}"
{"status" : "OK."}
```

图 5.6 QoS 流设置和结果图

实验中在控制器和终端主机监测数据流的传输情况，图 5.7 为控制器输出的部分 QoS 流路径信息。从图中可以看出，时延敏感流 Flow1 传输路径长度为 3，对应路径 S_1-S_2-S_{11}，路径总体时延 28ms，丢包参考值为 91，丢包敏感流 Flow2 传输路径长度为 4，对应路径 S_1-S_3-S_{10}-S_{11}，路径总体时延值为 36ms，丢包参考值为 56。由此可以看出，控制器为 $Flow_1$ 安排了延迟较低的路径，为 Flow2 安排了丢包较低的路径，与数据流 QoS 类型相符。在 $Flow_1$ 和 $Flow_2$ 传输过程中，路径 S_1-S_2-S_{11} 的实际带宽占用为 10%，路径 S_1-S_3-S_{10}-S_{11} 的实际带宽占用为 20%，此时控制器为 $Flow_3$ 安排了路径 S_1-S_2-S_{11}，给出的带宽计算值为 13%，由此可见控制器为 $Flow_3$ 安排了可用带宽较高的路径。

图 5.8 为主机 H_{11} 终端显示的数据接收信息图，图中显示主机 H_{11} 接收到来自主机 H_1（10.0.0.1）的三条数据流，第一条数据流 $Flow_1$ 传输数据量为 1.19Mbytes，传输为 995Kb/s，丢包率为 0.47%，第二条数据流 $Flow_2$ 传输数据量为 2.38Mbytes，占用带宽为 1.71Mb/s，丢包率为 0.24%，第二条数据流 $Flow_3$ 传输数据量为 5.94Mbytes，占用带宽为 3.37Mb/s，丢包率为 0.28%。

```
INFO  [n.f.t.TopologyManager] Recomputing topology due to: forced-recomputation
INFO  [n.f.f.Forwarding] [QoS流] LATENCY-SENSITIVE  RouteId [src=00:00:00:00:00:00:00:01 dst=00:00:00:00:00:00:00:0b]
INFO  [n.f.f.Forwarding] [路径] 长度：3 丢包：91 带宽占用：0% 时延：28ms
INFO  [n.f.f.Forwarding] [QoS流] LOSS-SENSITIVE  RouteId [src=00:00:00:00:00:00:00:01 dst=00:00:00:00:00:00:00:0b]
INFO  [n.f.f.Forwarding] [路径] 长度：4 丢包：56 带宽占用：0% 时延：36ms
INFO  [n.f.f.Forwarding] [QoS流] UTILIZATION-SENSITIVE  RouteId [src=00:00:00:00:00:00:00:01 dst=00:00:00:00:00:00:00:0b]
INFO  [n.f.f.Forwarding] [路径] 长度：3 丢包：98 带宽占用：13% 时延：28ms
```

图 5.7 控制器输出的 QoS 路径信图

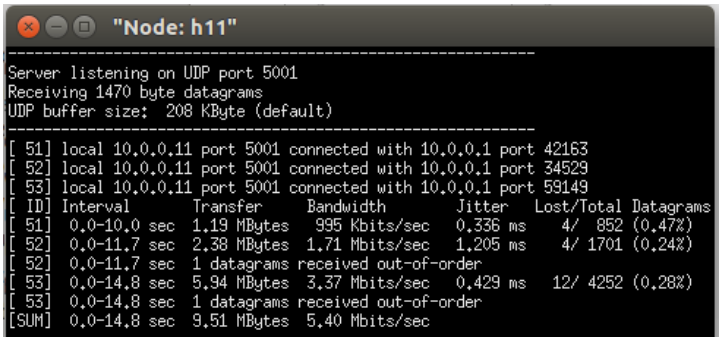

图 5.8 主机 H11 的数据流发送/接收信息图

该部分实验结果表明，控制器可以监测网络链路的运行状况，根据 QoS 流的配置为 3 种 QoS 流选择合适的转发路径。将控制器和主机两端数据进行对比，可以看到虽然控制器端的数据和主机端的数据实际上是有一定的误差的，但控制器仍可以根据测量值对路径的相对优劣进行判别，并找出相对最优路径进行数据的转发。

（2）QoS 流调度性能测试

为了进一步验证控制 QoS 感知路由效果，实验中将生成多组数据流进行测试。该部分实验进行 10 组数据流，每组中都包含三种 UDP 数据流，数据流发送速率都设置为 4Mb/s，数据流持续时间都为 30s。每组实验随机选中网络内的两个主机作为源节点和目的节点，流量生成时首先发送时延敏感流，随后发送丢包敏感流，最后是带宽敏感流。

实验结果数据如表 5.7 和表 5.8 所示，其中路径的时延数据值为控制器端的检测值，丢包值和传输速度值为主机端的输出值。从实验数据中可以看出，时延敏感流的平均时延值为 25.5ms，要远小于数据流的平均时延值 34.4ms，时延敏感流和丢包敏感流的丢包值要稍低于总体丢包均值。

从实验数据上还可以看出，三种发送速率相同的数据流的传输速率受到不同程度的影响，时延敏感流由于最先发送，且发送时的网络链路空闲，因此平均传输速率接近能够发送速率，而带宽敏感流在发送时由于网络链路内已经存在 2 条数据流，网络内多条链路的利用率已经达到了 40%以上，因此其速率受到了一定的影响，因此为带宽敏感流安排低负载的转发路径也是较为必要的。从实验结果上看，全部 10 条时延验敏感流能按累积时延最低的路径进行转发，而全部 10 条带宽敏感流总是能沿着可用带宽最大的路径进行转发，10 条丢包敏感流中有 7 条传输过程中丢包率最低，30 条数据流中有 27 条是按照最佳路径转发。若以控制器是否能为 QoS 流安排适合其需求类型的路径为评判标准，则上述实验中 QoS 感知路由的准确率能达到 90%。实验中发现，控制器检测到的最低时延路径往往跳数较少和丢包更低，时延敏感流和丢包敏感流的最优路径常常重合，此外，在前两条数据流使用同一条路径进行转发时，带宽敏感流总是能避开该高负载链路，选择可用带宽更高的路径进行转发，整个实验中未出现三条数据流使用同一条路径的情况。

从总体上看，控制器可以为时延敏感流安排低时延链路，为丢包敏感流安排低丢包路径，为带宽敏感流安排低负载链路，实验结果和预期相符。另外，实验中的流量调度测试都是在网络运行良好并且链路不出现明显拥塞的条件下进行，网络链路带宽能够满足数据流的理论传输速率。若在网络带宽不足的情况下，链路拥塞会使数据流会产生大量的丢包，使数据流传输受到影响，这种情况下还需

要一些带宽保障或流量整形等技术以保障 QoS 流的传输。

表 5.7 QoS 流测试数据表

序号		带宽敏感流	丢包敏感流	带宽敏感流
1	时延	32ms	32ms	49ms
	丢包率	1.1%	1.3%	0.6%
	传输速率	3.95Mb/s	3.43Mb/s	2.87Mb/s
2	时延	7ms	7ms	31ms%
	丢包率	1.4%	1.2%	1.6%
	传输速率	3.94Mb/s	3.48Mb/s	2.99Mb/s
3	时延	15ms	31ms	42ms
	丢包率	0.1%	0.0%	0.6%
	传输速率	4.00Mb/s	3.62Mb/s	3.32Mb/s
4	时延	18ms	18ms	57ms
	丢包率	0.0%	0.1%	0.1%
	传输速率	4.00Mb/s	3.40Mb/s	2.81Mb/s
5	时延	13ms	13ms	25ms
	丢包率	0.9%	0.8%	0.4%
	传输速率	3.95Mb/s	3.45Mb/s	2.92Mb/s
6	时延	47ms	57ms	60ms
	丢包率	1.4%	0.9ms	1.2ms
	传输速率	3.95Mb/s	3.39Mb/s	3.01Mb/s
7	时延	39ms	52ms	60ms
	丢包率	0.7%	1.2%	2.0%
	传输速率	3.97Mb/s	3.39Mb/s	3.01Mb/s
8	时延	43ms	53ms	59ms
	丢包率	0.7%	0.7%	1.6%
	传输速率	3.97Mb/s	3.30Mb/s	2.89Mb/s
9	时延	27ms	27ms	36ms
	丢包率	0.7%	0.7%	0.1%
	传输速率	4.0Mb/s	3.30Mb/s	2.76Mb/s
10	时延	14ms	14ms	54ms
	丢包率	0.5%	0.3%	1.2%
	传输速率	3.99Mb/s	3.40Mb/s	2.86Mb/s

表 5.8 QoS 流测试总体数据表

数据流	平均时延	平均丢包率	平均速率
时延敏感流	25.5ms	0.73%	3.97Mb/s
丢包敏感流	30.4ms	0.81%	3.43Mb/s
带宽敏感流	47.3ms	1.04%	2.95Mb/s
总体	34.4ms	0.86%	3.45Mb/s

5.3.3 普通流调度测试

实验将测试控制器对普通流的调度情况，由于控制器对普通流中的 elephant 流和 mice 流方式有差异，因此实验分为两个部分进行，实验中将占用带宽超过

链路带宽 30%的数据流视为 elephant 流，其他数据流视为 mice 流。

（1）mice 的调度

中小规模数据流使用单路径进行转发，路由时选择最空闲的链路进行转发，实验使用拓扑图和图 4.13 相同，实验以主机 H_1 为源主机，H_2 为目的主机，数据流皆为 1Mbps 的 UDP 数据流，UDP 流使用不同的端口，因此使用不同的流表进行转发，实验每生成一条数据流就控制器就会检测一次路径的速率。实验结果图如图 5.9 所示：

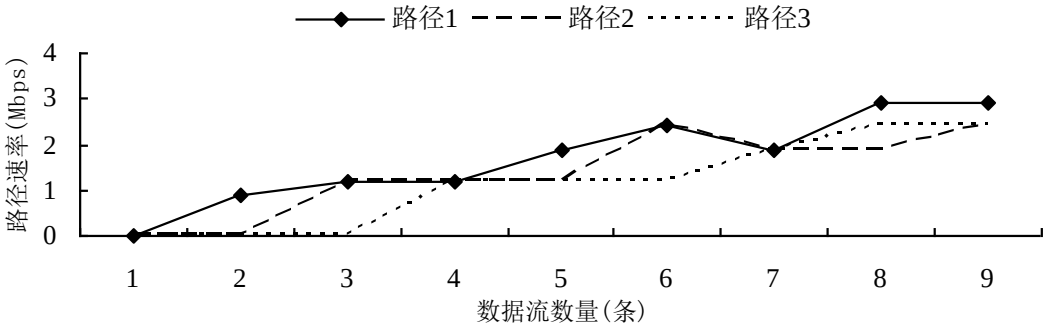

图 5.9 路径速率图

从图中可以看出，源主机和目的主机间 3 条路径初始速率为 0，前 3 条数据按不同的路径进行转发，然后随着数据流的不断增加，控制器总是能选取当前最空闲的路径进行转发，使得各路径的速率平缓增加，各链路利用率能保持在相近的水平。实验结果显示，对于中小规模的数据流，控制器路由时能较好地利用源节点和目的节点间的多条路径进行转发，能充分利用网络可用链路，避免数据流都汇集到同一条路径上造成网络拥塞。

（2）elephant 流的调度

实验生成 UDP 数据流从主机 H_{10} 发送到主机 H_{11}，交换机 S_{10} 中计量表 Meter 设置如表 5.9 所示，实验将生成 10 条普通流，数据流的速率分别设置为 1Mb/s，2Mb/s，…,10Mb/s，各条数据流独立进行测试。实验过程中记录各条路径的速率。

表 5.9 交换机 Meter 数据表

Band Type	Rate	Type specific arguments	说明
dscp_remark	3M	prec_level=4	dscp+1
dscp_remark	6M	prec_level=8	dscp+2
drop	9M	Drop	丢弃数据包

实验结果如图 5.10 所示，从图中可以看出当数据流速率为 1Mb/s，2Mb/s 和 3Mb/s 时，控制器将数据流视为 mice 流，只使用一条路径进行数据转发，在数据流速率超过 3Mb/s 后，数据流为 elephant 流，控制器开始使用第二条路径进行转发，速率超过 6Mb/s 后使用第三条路径进行数据的转发。控制器信息显示，在 H_1 发送速率达到 10Mb/s 后，路径 S_1-S_5 的平均使用带宽为 2.93Mb/s，路径 S_1-S_2-S_5

的平均使用带宽为 2.92Mb/s，路径 S_1-S_3-S_4-S_5 的平均使用带宽为 2.93Mb/s，总体速率约为 8.78Mb/s。可以看出，数据流按照三条路径进行转发，每条路径的速率限制在 3Mb/s 左右，数据流的总体速率控制在 9Mb/s 以内，与计量表设置的数值相符。实验结果显示，控制器可以根据交换机端口速率安排路径，数据流速率超过计量表阈值后会安排新的路径进行转发，每条路径的流速保持在 3Mb/s 以下，使普通流流量得到比较均衡的传输。

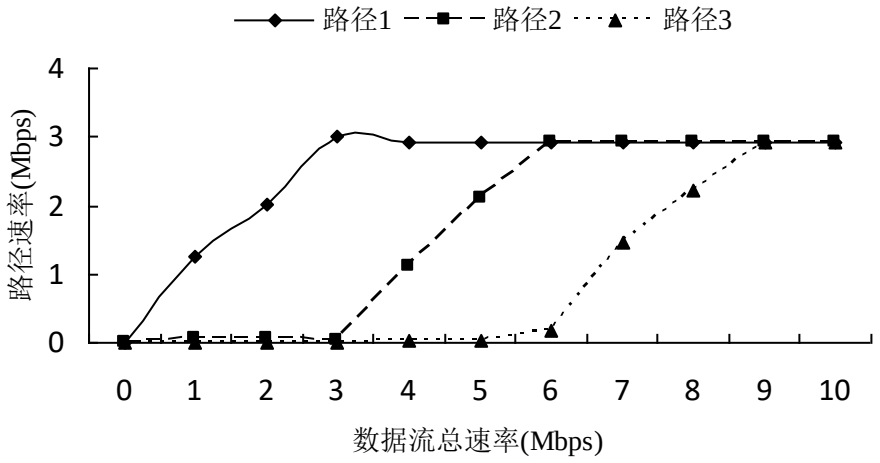

图 5.10　交换机端口发送速率图

下面将对影响实验结果的一些因素进行分析：

（1）数据监测的精度

由于控制器监测网络时采用的是主动采集的方式，控制通道带来的处理延迟是无法避免，控制器无法监测到网络的实时链路状况，因此数据的精确性和有效性会受到一定的影响。但从实验结果上看，控制器采集的信息还是能较为准确地反应网络链路的状况，为数据流安排合适的路径。网络内大多数丢包发生在流量洪峰时期，流量稳定期产生的丢包量较少。测试结果中显示，由于控制器内测量的是交换机端口的瞬时丢包率，当链路负载较大时，链路丢包测量值的准确率较高，此时控制器能较好地绕开高丢包链路对丢包敏感流进行转发。而当链路负载较低时，丢包样本数据值较小，计算出的丢包值的相对误差较大，但此时各链路的绝对丢包数小，此时各丢包值相近链路的选择对丢包敏感流的传输差别并不明显，因而对丢包敏感流的影响不大。

（2）系统硬件处理能力

在系统运行处理能力不足时网络对于数据流处理效果会降低，具体表现为，CPU 运行能力不足时，控制器数据流寻路的时间会增加，软件交换机会转发数据的时延明显增加，数据流会产生一定程度的丢包，传输速率也受到影响。而当 CPU 处理能力严重不足时，甚至会出现交换机一度不能和控制器正常交互信息，

交换机丢失链接的情况。因此实验中限制了实现流量的规模，只进行小规模的流量调度测试，因而实验结果无法反映大型网络或复杂网络运行状况。

（3）网络拓扑：因特网中的网络拓扑结构复杂，往往没有统一的结构，节点之间也不一定会存在多条传输路径，因而文中方案不适合与在源节点和目的节点只存在单一路径的情况。而数据中心中大量采用的树形层次结构，在节点和节点之间存在多条等价路径，文中策略能较好地将流量平均分配到各条链路上，对树形对称结构拓扑的网络的效果会更好。

5.4 本章小结

本章通过 SDN 仿真技术对文中方案进行测试。通过 Mininet 工具搭建 了网络环境，首先验证了 Floodlight 控制器的功能，应用可以通过控制器应用接口配置 QoS 流，在实验中同源同目的数据流，模块可以为时延敏感流安排低延迟的路径进行转发，为丢包敏感流安排低丢包链路的路径进行转发，为带宽敏感流安排可用带宽较高的路径进行转发，对于普通流的实验结果显示，使用计量表可以较好地将网络流量使用多路径进行转发，有效地利用网络带宽资源。本章最后对实验进行了总结，分析了影响实验结果的因素，总结了实验中的不足。

第 6 章 总结与展望

6.1 总结

软件定义网络化简了网络的管理，支持灵活而高效的网络控制，具有突出的研究意义。本文实现了 QoS 感知的流量管理策略，充分利用了 SDN 控制层可编程的特性，实现了 QoS 感知的路由，使控制器能针对性地为不同数据流制转发路径，实行差异化的流量调度。

本文的工作主要总结如下：

（1）研究软件定义网络的背景知识，详细介绍 SDN 架构和主要特征，以及 OpenFlow 协议和相关的 QoS 技术，同时对国内外相关研究现状进行分析。

（2）以 Floodlight 控制器为原型，分析了控制器原型的不足，对控制器的 QoS 功能进行设计开发：设计 QoS 交互接口，实现 QoS 管理模块对数据流信息进行管理，制定监测网络链路状况的方案，在转发模块识别不同的数据流并采取差别化的转发策略，面向 QoS 的路由算法等等。

（3）设计并实现了 QoS 感知的流量管理策略，充分利用 SDN 集中控制和可编程的优势，为时延敏感流，丢包敏感流和带宽敏感流进行 QoS 选路，并使用计量器对普通流进行有效地管控，合理地规划不同数据流的路径。

（4）通过 SDN 仿真软件搭建平台对控制器测试，验证该流量管理策略的效果。实验结果表明，通过控制器 REST API 可以便捷而有效地配置 QoS 数据流，网络可以通过接口快速感应到上层应用业务的变动，进而实施不同的应对策略。同时，控制器能识别出不同 QoS 流报文并为数据流安排适合其 QoS 类型的路径。实验数据显示，时延敏感流的平均时延较低，时延敏感流和丢包敏感流的平均丢包较少，并且带宽敏感流总是能按照可用带宽较大的路径进行传输，QoS 流的传输质量得到了提升。控制器还能通过计量表监测数据流速率，较好地对普通流进行实行多路径的转发，能充分利用网络带宽资源。

6.2 未来工作与展望

流量爆发式增长要求下一代网络可以更加智能地处理流量，作为新型网络的代表，SDN 已经在诸多取得了突出成果，但未来在网络通用性和稳定性以及与其他学科的交叉应用等诸多方面仍有待进一步研究。针对文中的不足，还需要针对以下几方面的内容进行研究。

（1）带宽保障问题。文中实现主要保障 QoS 选路的灵活和便捷性，但没有

采取对 QoS 流的带宽进行保障措施，因而在网络处理能力及其有限的情况下不能很好地保障 QoS 流的传输。若能将 QoS 路由与队列和计量表机制进行结合，灵活对网络带宽等资源进行分配，则可以更好地保障传输过程中的服务质量，提高用户体验。

（2）控制层和数据层之间的控制代价问题。文中控制器监测方式为 OpenFlow 框架下的主动测量模型，控制通道的固有延迟难以避免，而且主动采集来回带来额外的开销，因而网络信息的精确性和有效性会受到影响。其他 OpenFlow 框架外的技术（例如基于端口信息采样的被动式信息采集技术）则可以从一定程度上减少这个问题带来的影响。

（3）控制器的性能和稳定性问题。文中采用的是单控制器的模型，网络数据流处理的能力会受到控制器计算能力的限制。特别是在大型网络中，维护整个网络的拓扑和路由信息是一件十分复杂的工作，控制器信息处理不及时往往会导致网络瓶颈，甚至出现网络瘫痪。因此如果能将多控制器模型引入，不同控制器之间明确分工，相互协同工作，能极大地提高网络处理的效率，而目前多控制器网络协同的问题还有待进一步研究。另外，文中控制器没有实现故障恢复功能，面对服务器宕机，控制器中信息丢失后往往难以恢复，在系统的稳定性和可靠性上还有待进一步加强。

文中还有很多值得完善和优化的地方，需要在日后工作学习中不断改进。相信随着 SDN 技术的不断发展成熟，网络的服务质量定会稳步向前实现质的飞跃，从而进一步推动信息技术相关产业的发展，造福人们的生活。

参考文献

[1] 周桐庆, 蔡志平, 夏竟,等. 基于软件定义网络的流量工程[J]. 软件学报, 2016, 27(2):394-417.

[2] 黄晓雯. 云计算体系架构与关键技术[J]. 中国新通信, 2014(13):29-29.

[3] Sharma S, Staessens D, Colle D, et al. Implementing Quality of Service for the Software Defined Networking Enabled Future Internet[C]// Third European Workshop on Software Defined Networks. IEEE, 2014:49-54.

[4] 赵钊. 基于 OpenFlow 的视频流媒体路由选择算法[D]. 深圳大学, 2017.

[5] Jeong S, Lee D, Choi J, et al. Application-aware Traffic Management for OpenFlow networks[C]// Network Operations and Management Symposium. IEEE, 2016:1-5.

[6] Cheng L C, Wang K, Hsu Y H. Application-aware Routing Scheme for SDN-based cloud datacenters[J]. 2015:820-825.

[7] Tegueu F S, Abdellatif S, Villemur T, et al. Towards application driven networking[C]// IEEE International Symposium on Local and Metropolitan Area Networks. IEEE, 2016:1-6.

[8] Wang P, Lin S C, Luo M. A Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs[C]// IEEE International Conference on Services Computing. IEEE, 2016:760-765.

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

[10] MIT Technology Review [EB/OL].http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/.

[11] 张朝昆, 崔勇, 唐嚣祎,等. 软件定义网络(SDN)研究进展[J]. 软件学报, 2015, 26(1):62-81.

[12] 陈忠. SDN 网络中 QoS 控制技术研究与实现[D].电子科技大学,2017.

[13] 段鹏飞. 基于软件定义的车联网 QoS 路由框架研究[D]. 华东师范大学, 2017.

[14] Kreutz D, Ramos F M V, Esteves Verissimo P, et al. Software-Defined Networking: A Comprehensive Survey[J]. Proceedings of the IEEE, 2014, 103(1):10-13.

[15] Sardis F, Condoluci M, Mahmoodi T, et al. Can QoS be dynamically manipulated using end-device initialization?[C]// IEEE International Conference on Communications Workshops. IEEE, 2016:448-454.

[16] Egilmez H E, Dane S T, Bagci K T, et al. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined

Networks[C]// Signal & Information Processing Association Summit and Conference. IEEE, 2012:1-8.

[17] Egilmez H E, Civanlar S, Tekalp A M. An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks[J]. IEEE Transactions on Multimedia, 2013, 15(3):710-715.

[18] Zhao S, Sydney A, Medhi D. Building Application-Aware Network Environments using SDN for Optimizing Hadoop Applications[C]// Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:583-584.

[19] Bari M F, Chowdhury S R, Ahmed R, et al. PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks[C]// Future Networks and Services. IEEE, 2013:1-7.

[20] Krishna H, Adrichem N L M V, Kuipers F A. Providing bandwidth guarantees with OpenFlow[C]// Communications and Vehicular Technologies. IEEE, 2016:1-6.

[21] 曹绍华, 张鑫. 面向业务的 SDN 网络带宽保障研究[J]. 计算机工程与应用唯一官方网站, 2016, 52(22):127-132.

[22] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: dynamic flow scheduling for data center networks[C]// Usenix Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, Ca, Usa. DBLP, 2010:281-296.

[23] Curtis A R, Mogul J C, Tourrilhes J, et al. Devoflow: Scaling flow management for high-performance networks[J]. Acm Sigcomm Computer Communication Review, 2011, 41(4):254-265.

[24] 李夺. 基于 Floodlight 控制器的 QoS 模块扩展[D].吉林大学,2017.

[25] Foundation O N. Software-Defined Networking: The New Norm for Networks[J]. 2012.

[26] OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04) [EB/OL].https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf.

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

[28] Nguyen X N, Saucez D, Barakat C, et al. Rules Placement Problem in OpenFlow Networks: A Survey[J]. IEEE Communications Surveys & Tutorials, 2016, 18(2):1273-1286.

[29] 胡恂. 软件定义网络路由选择算法研究[D].重庆邮电大学,2016.

[30] Chen F, Wu C, Hong X, et al. Engineering traffic uncertainty in the OpenFlow data plane[C]// IEEE INFOCOM 2016 - the, IEEE International Conference on Computer Communications. IEEE, 2016:1-9. 1

[31] Floodlight Architecture[EB/OL]. http://docs.projectfloodlight.org/display/floodlight

controller/Architeture/.

[32] 程显雯. SDN 中面向北向的控制器关键技术的研究[D]. 北京邮电大学, 2015.

[33] 付健,沈苏彬.一种基于 SDN 的 QoS 应用编程接口设计方案[J].计算机技术与发展,2015,25(11):99-104.

[34] RFC791. Wilson Boulevard. Internet Protocol[S]. September 1981

[35] Dijkstra E W. A note on two problems in connexion with graphs[J]. Numerische Mathematik, 1959, 1(1):269-271.

[36] Yen J Y. FINDING THE K SHORTEST LOOPLESS PATHS IN A NETWORK[J]. Management Science, 1971, 17(11):712-716.

[37] Benson, Theophilus, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild[C]// Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010.

[38] Lantz B, Heller B, Mckeown N. A network in a laptop:rapid prototyping for software-defined networks[C]// ACM Workshop on Hot Topics in Networks. HOTNETS 2010, Monterey, Ca, Usa - October. DBLP, 2010:1-6.

[39] Open vswitch [EB/OL]. http://openvswitch.org/.

[40] Open Flow 1.3 Software Switch [EB/OL]. http://cpqd.github.io/ofsoftswitch13/.

[41] Party R. GEANT/Abilene Network Topology Data and Traffic Traces[J]. 2015.

致 谢

时间过得很快，不知不觉就接近了毕业的日子。在中国科大读研的阶段有过迷茫，有过彷徨，更有收获和成长。在这里遇到了一批热情睿智的同学，一批的兢兢业业教师，你们是我读研阶段的良师益友，鞭策我在学习和科研道路上不断加强自我素养，稳步前进。感谢学校给我提供了优越的学习的环境，感谢学校老师的栽培。

本论文是在黄刘生教授和徐宏力副教授的悉心指导下完成的，忠心地感谢恩师对我的谆谆教导。论文撰写过程中曾经碰到过许多问题，实验室的老师和学长们曾经结过我许多建议，这给了我莫大的帮助。老师和学长们严谨的治学态度，严于律己的生活作风深深感染了我，在此谨向黄刘生教授和徐宏力副教授致以衷心的感谢和崇高的敬意。

最后，我要向百忙之中参与审阅、评议本论文各位老师、向参与本人论文答辩的各位老师表示由衷的感谢！感谢在研究生阶段曾经帮助过我的所有老师、同学和朋友们，有你们的陪伴我的学习生活变得更加精彩。

2018 年 4 月