密　级　公开
分类号　TP393

西安工业大学

XI'AN TECHNOLOGICAL UNIVERSITY

硕士学位论文

题　目：　基于 SDN 网络的视频流媒体传输性能研究

作者：　才新

指导教师：　李静　教授

申请学位学科：　模式识别与智能系统

2019 年 4 月 27 日

基于 SDN 网络的视频流媒体传输性能研究

学　科：模式识别与智能系统

研究生签字：

指导教师签字：

摘　要

伴随着互联网行业与多媒体技术的蓬勃发展，视频流媒体应用在互联网上大放异彩。其严格的QoS需求为人们的生活和工作带来了便利，但同时也给网络传输带来了不小的挑战。传统网络尽力而为的转发方式和分布式的控制形式常常导致网络中流量调度的迟滞，这会造成视频传输的不稳定，严重影响视频传输的服务质量。软件定义网络（Software Defined Network，SDN）是一种与传统网络结构有所不同的网络架构，其转控分离、集中控制、可编程的思想为网络中的流量工程、QoS路由等问题提供了独特的解决思路。本文旨在利用SDN的特性提高视频流媒体的传输性能，主要研究内容有以下几个方面：

首先，本文将网络中的业务分为四种类型，并根据业务对QoS需求的不同，对业务进行优先级的区分并为他们设定不同的优先级值，提出了视频流媒体的QoS控制策略。其次，基于遗传算法的QoS路由为视频流媒体（最高优先级业务）计算传输路径，基于Dijkstra算法以跳数为代价为其他优先级业务计算传输路径。当控制器监测到视频流媒体的传输路径出现拥塞时，采取动态路由措施更好地保障视频流媒体的QoS。

再次，使用HTB队列规则在OpenFlow交换机上实现不同优先级业务的区分调度，优先保障视频流媒体业务的QoS。同时提供带宽充足时的借带宽机制，尽力保障每个业务流的传输需求。

最后，对QoS控制策略中的拓扑管理模块、链路信息测量模块、路由管理模块以及队列调度模块分别进行了设计与实现，并在由Mninet、Ryu控制器、摄像头等软硬件搭建的SDN网络传输环境中，对QoS控制策略进行了测试。通过一系列仿真测试实验，从链路的时延抖动、吞吐率等视频流媒体的传输性能参数方面验证了本文控制策略能够较好地提

高视频流媒体的传输性能。实验结果表明，QoS控制策略能为视频流媒体选择一条符合需求的路径进行传输；能在传输路径出现拥塞时为视频流媒体提供重选路由的机制；能够在数据转发层保障不同优先级业务的QoS，较好地保证了视频流媒体业务端到端的QoS需求。

关键词：视频流媒体；软件定义网络；QoS路由；队列调度

Study the Performance of Video Streaming Media Transmission based on SDN Network

Discipline： Pattern Recognition and Intelligent System

Student Signature：

Supervisor Signature：

Abstract

With the booming development of the Internet industry and multimedia technology, video streaming media applications on the Internet shine brilliantly.there strict QoS performance brings convenience to people's life and work, but at the same time they also bring a great challenge to the network transmission. The best effort forwarding mode and distributed control form of the traditional network often lead to the hysteresis of traffic scheduling in the network, which will cause the instability of video transmission and seriously affect the service quality of video transmission.Software Defined Network (SDN) is a kind of network architecture which is different from the traditional network structure, its characteristics of transfer control separation, centralized control and programmability provide a unique solution to the problems of traffic engineering and QoS routing in the network.The purpose of this paper is to improve the transmission performance of video streaming media by utilizing the characteristics of SDN, the main contributions of our work are as follows:

Firstly, this paper divides the businesses in the network into four types, and according to the different needs of businesses for QoS, the paper classifies the priority of businesses and sets different priority values for them, and proposes the QoS control strategy for video streaming media.Secondly, the QoS routing based on genetic algorithm computes the transmission path for the video streaming media (the highest priority business), and calculates the transmission path for other priority services at the cost of the jump number based on the Dijkstra algorithm.And when the controller monitors congestion in the transmission path of video streaming media, dynamic routing measures are adopted to better protect the QoS of video streaming media.

Thirdly, the HTB queue rules are used to implement the differentiated scheduling of

different priority services on the OpenFlow switchs, which gives priority to guaranteeing the QoS of video streaming media services.At the same time, it provides the borrowing bandwidth mechanism when bandwidth is sufficient, so as to guarantee the transmission requirements of each service flow as much as possible.

Finally, the topology management module, link information measurement module, routing management module and queue scheduling module in QoS control strategy are designed and fulfilled separately, and the QoS control strategy is tested in the SDN network transmission environment built by Mninet, Ryu controller, camera and other software and hardware. Through a series of simulation test experiments, the paper verifies that the control strategy proposed in this paper can improve the transmission performance of video streaming media better from the aspects of the transmission performance parameters of video streaming media such as delay jitter and throughput rate of link. Experimental results show that QoS control strategy can select a path that meets the requirements of video streaming media for transmission, it can provide a re-routing mechanism for video streaming media when the transmission path is congested, it can guarantee the QoS of different priority services at the data forwarding layer, so as to ensure the QoS requirement of the video streaming media service end-to-end.

Key Words: Video streaming media; Software Defined Network; QoS routing; Queue scheduling

目　录

1 绪　论

1.1 研究背景及意义

在多媒体业务快速普及的背景下，视频应用及视频用户的数量激增。中国互联网络信息中心（CNNIC）发布的《第42次中国互联网络发展状况统计报告》显示[1]：到2018年6月底，网络视频用户规模达6.09亿，与去年年末相比增长了5.2%；网络直播用户规模达4.25亿，与去年年末相比增长了7%，视频应用正在日益成为多媒体通信中发展最为迅速的领域。

互联网技术的崛起与快速发展促进了视频业务的方兴未艾，视频点播、视频直播、视频会议及监控视频等一系列流媒体视频应用正不断地改变着人们的工作和生活。在庞大且复杂的网络中如何更好地为用户提供视频服务，成为流媒体视频领域研究的一个方向。一般来讲，若想得到较好的视频观看效果，需要在视频传输过程中保证网络带宽的充足以及较低的时延和丢包率，但是传统网络发展到今天已有了较大的局限性，不可能无限制地提供资源来满足不同视频的传输需求。针对视频流媒体业务在网络中传输遇到的困境，IETF提出了多种QoS模型和机制，但这些模型和机制都建立在分布式的网络架构上，全局网络状态信息的收集难度较大，使得不能及时改变数据传输策略的情况经常出现，无法有效地保证视频流媒体的服务质量（Quality of Service，QoS）[2]。当大量数据请求叠加时，容易造成传输路径的拥堵，进而导致视频数据经过很长时间才到达对端或者拥堵时间过长而直接被丢弃，严重影响视频的QoS。

技术的革新促进了视频流媒体应用在互联网上的百花齐放，但同时也对传统网络提出了更高的要求，面对飞速发展的视频流媒体业务，传统网络越来越“心有余而力不足”。面对流媒体应用的大量涌出、传统网络日渐凸显的弊端以及在传统网络中部署新的方案不但难以彻底解决问题反而加剧网络的复杂的情况，越来越多的网络研究人员倾向于寻求一种新的网络体系结构，以便能解决网络传输效率低的问题。软件定义网络（Software Defined Network，SDN）[3]是一种有别于传统网络架构的新式网络结构，其不再让控制平面和转发平面紧紧捆绑在一起，而是让它们“各司其职”，控制平面可以获得网络的全局拓扑及流量信息，并根据这些信息作出转发路径的决策，数据转发平面不做任何控制的决策，其只根据收到的转发命令进行业务流的转发。这对网络和网络设备来说是非常大的解放，于整个网络而言，网络操作变得更简便和容易；于网络设备而言，没有了控制功能的网络设备的转发速度会有很大的提升，能够提高网络的传输效率。由于SDN架构的开放性和可编程性，在控制器上编写相应的程序，便可改变转发层网络设备的行为，实现自己的需求。对于视频传输而言，利用SDN网络架构可以更好地控制视频流的传输，使网络更好地为视频业务服务。

1.2 国内外研究现状

1.2.1 SDN 研究现状

面对传统网络越来越难以满足各种新型网络应用传输的问题，世界范围内掀起了对新型网络进行研究的热潮，比如欧盟的 FIRE[4]、美国的 GENI[5]等。源于校园的 SDN 被认为在解决传统网络的一些问题上有着其他网络架构无法比拟的优势，越来越多的高校及企业都在关注 SDN 的发展趋势甚至投入大量的资金进行研究。

从 2006 年概念的提出到校园研究再到近几年真正的实现应用，SDN 的发展已有十几年。目前，SDN 各项标准的制定由开放网络基金会（Open Networking Foundation，ONF）完成 [6]，其对 OpenFlow 协议进行了不断地修改和完善，并持续充实 SDN 的相关内容。作为南向接口协议的标准，OpenFlow 从最初的 1.0 版本到现在的 1.5[7]版本，其能实现的功能越来越完备，面向应用层的北向接口虽未标准化也在不断的完善中。作为软件定义网络的核心，SDN 控制器和交换机的发展也很快，华为、IBM、H3C 等都推出了自己厂商的交换机，各种语言编写的 SDN 控制器抢占着控制器的市场，目前使用较多的控制器有 Ryu[8]，FloodLight[9]，OpenFaylight[10]。

目前国内外各大高校的研究重点主要在如流量工程、QoS 路由以及与传统网络互联等的 SDN 应用、SDN 测试实验以及控制平面的设计等方面。除此之外，国内外各大互联网公司也纷纷参与到 SDN 的研究与应用工作中，谷歌使用 SDN 网络架构的思想在其跨欧亚美三大陆的 12 个数据中心网络成功部署了 B4 网络[11]，B4 的最大特点是实现了整个网络的链路利用率达到了 95%以上；微软提出的 software-driven WAN[12]实现了网络链路利用率达到了 99%以上；中兴公司正着手进行基于 NFV/SDN 的面向 5G 网络运维管理转型的解决方案架构设计。

国内三大运营商也加入到 SDN 研究的浪潮中，如中国电信为应对云资源池网络的挑战，在云资源池中部署应用了网络业务敏捷灵活、网络规模支持海量多租户、云网融合智能编排的 SDN 方案，并实现了广泛的部署和应用。SDN 交换机的设计与研发也引起了许多厂商的兴趣，如思科、IBM 等厂商都推出了具备 SDN 功能的交换机及芯片；中国的华为公司在 2013 年发布了其第一款支持 SDN 功能的交换机；随后盛科等公司也推出了自己公司的 SDN 交换机。

1.2.2 SDN 网络中视频流媒体 QoS 控制策略的研究现状

1）国内研究现状

文献[13]提出了 OpenFlow 网络下可分级视频编码（SVC）的流媒体传输方案，该方案利用 SVC 编码将视频分成不同的层级（基础层和增强层），根据网络的状况，传输不同的层级，基础层保证了最基本的观看效果。

文献[14]提出了一种新的基于 OpenFlow 网络的 QoS 算法，文中将网络中的流分为两类，对于需要改善其传输性能的数据流使用文中提出的 QoS 算法计算传输路径，对于其他数据流则直接以默认的路径进行传输。

文献[15]通过构建视频流分类模型，并将 DiffServ 的思想应用到视频流的调度中，以 PGPS 等调度算法为基础，给予视频业务差别的 QoS 保障，满足了网络中视频业务的传输需求。

文献[16]利用 SDN 架构的特点分别从控制层面和转发层面实现了基于 SDN 的业务 QoS 保障，文中的 QoS 保障体现了 SDN 架构的优越性。

2）国外研究现状

2007 年，K.-W.Kwong 等人提出了根据权重计算路由的算法[17]，该算法根据一定规则把网络中的流分为两类，根据每类流所需要的链路权重为其计算路由。

Civanlar 等人利用 OpenFlow 网络设计并实现了视频流高效传输的方案[18]，该方案利用 SVC 编码的特点为视频流计算传输路径，对于视频流之外的数据流都采用最短路径进行传输。

2009 年，Nikhil Handigol 等人设计了一种利用 SDN 网络优势的路由算法[19-20]，数据流传输路径的计算由当前网络负载情况决定。

Egilmez 等人设计了一种基于 SDN 的 OpenQoS[21]控制策略，该策略利用不同数据流的数据包头字段存在的差异，将传输的流量分为两种不同类型，对于视频流，OpenQoS 策略考虑传输路径上的延迟、丢包情况为其计算满足 QoS 需求的传输路径，其他数据流的路由仍是最短路径。

Dobrijevic 等人以 QoE 的概念表示接收端收到的视频质量，将蚁群算法应用到路由选路中，为不同需求的业务计算满足其 QoE 最优的传输路径[22]。将运行该策略得到的结果与最短路径算法的结果进行比较可知，该策略优于最短路径算法。

Owens 等人利用 SDN 网络的特点提出了一种用于保障视频业务 QoS 的策略，该策略能为视频流提供严格的服务质量需求[23]。该策略需要修改 OpenFlow 交换机部分功能才能实现，数据流在转发端口排队时，根据控制策略提供的流量规范来调整每个流的流量。

1.2.3 视频流媒体 QoS 研究存在的问题

整体上来看，国内外在 SDN 网络中对视频流媒体 QoS 的研究还存在一些不足，没有深入挖掘和利用 SDN 的优势，具体来说有以下几个问题。

一是虽然利用了 SDN 的可编程性，但并未深入研究 QoS 路由算法，没有使智能算法在 SDN 中充分发挥其优势以及没有借助 SDN 更好地解决网络中视频流媒体的路由问题。

二是尽管利用了 SDN 转控分离的思想，在控制层和数据层分别进行了业务 QoS 保障的研究，但很少有对二者一起进行研究的，缺乏控制的整体性。

三是尽管利用了 SDN 可集中收集网络信息的特点，但是控制策略作出之后便不在更

改，未能考虑到网络中各种信息资源都是动态变化的情况，不能提供灵活、动态的控制策略。

1.3 论文的主要研究内容及结构安排

1.3.1 论文的主要内容

在互联网飞速发展的时代背景下，流媒体类业务以其独特的优势受到人们的追捧，更以势不可挡之势逐渐占据着网络中的大部分流量。随着流媒体业务的增多，其又对网络的传输有着很高的要求，传统网络的弊端逐渐凸显，不能很好地保证端到端的 QoS。同时网络中对 QoS 有需求的业务有很多，不同业务的需求也不尽相同，解决此类问题必须将业务进行分类并设置不同的优先级，然后根据不同优先级业务的不同需求，对业务进行差异化的传输，以实现不同业务的传输要求。根据分析的结果及本课题的需要，本文在 SDN 网络环境中，利用 SDN 有别于传统网络的优势，对队列调度、QoS 路由等策略进行了探究，设计了基于业务优先级的视频流媒体 QoS 控制策略，并实现了控制策略中的每个模块及最后的仿真验证。旨在优先保障视频流媒体的服务质量，同时又兼顾其他不同业务的 QoS。现将本文的研究内容归纳如下：

一是对网络中的业务进行分类，根据每类业务 QoS 需求的不同对业务的优先级进行设置。

二是从控制层和数据层制定整体的视频流媒体 QoS 保障策略。控制层采用 QoS 路由策略，它通过基于遗传算法的 QoS 路由满足了最高优先级业务的 QoS 需求，而对其他优先级业务使用 Dijkstra 算法基于跳数的路由。数据层的队列调度，它能对不同优先级的业务进行区分调度，保障了视频流媒体在转发端口的优先传输，同时提供带宽充足时的借带宽机制，尽量满足其他业务的 QoS。

三是完成了 QoS 控制策略中的各个功能模块。一方面，在控制器上编写程序实现了基于遗传算法的 QoS 路由，较好地保障了视频流媒体的 QoS。另一方面，在 OpenFlow 交换机上进行了队列的配置，通过控制器下发的指令将数据包送入配置好的队列中，实现了不同优先级业务数据包的差别调度，在转发端口进一步保证了视频流媒体的 QoS。

四是搭建软件定义网络仿真实验平台，对 QoS 控制策略进行测试。

1.3.2 论文的结构安排

本文的主要工作是在 SDN 网络中，利用 SDN 网络的优势，在控制层和转发层上制定传输控制策略，以满足视频流媒体业务端到端的 QoS 需求，同时最大可能地满足其他业务的传输。根据课题的研究内容，论文结构的安排如下：

第一章，绪论。论述课题的研究背景和意义，综合分析和总结了SDN以及视频流媒体QoS控制策略的国内外研究现状，并对课题的主要研究内容及论文结构安排给予了概述。

第二章，SDN中QoS相关技术分析。对QoS技术和SDN网络进行了分析，并将传统网络下流媒体传输遇到的问题给予了分析总结。

第三章，QoS 控制策略需求分析与设计。对应用环境和应用需求进行了分析，针对视频流媒体传输的特点，提出了基于业务优先级的视频流媒体 QoS 控制策略，分别对 QoS 路由和交换机端口队列调度策略进行了研究。

第四章，QoS 控制策略实现。根据第三章提出的控制策略，分别对 QoS 路由和队列调度策略进行了实现。包括链路信息获取、基于遗传算法的 QoS 路由、HTB 队列规则的配置。

第五章，实验仿真与结果分析。搭建仿真实验平台，对QoS控制策略进行测试以及对仿真结果进行分析。

第六章，总结与展望。对论文整体研究工作和存在的不足进行了总结，并对未来的研究提出了展望。

2 SDN 中 QoS 相关技术分析

2.1 QoS 技术

2.1.1 QoS 定义

服务质量（Quality of Service，QoS）是指网络利用各种技术保障网络时延、吞吐量等性能服务要求，为某种业务提供优质服务的一种机制[24]。在通信中，QoS 主要通过各种性能参数体现，常见的 QoS 性能参数有吞吐量、丢包率、时延[25]。

吞吐量：受各种因素的影响，网络中数据包的传输并不是百分之百到达对端，吞吐量即指单位时间内顺利到达对端的数据包数量。

丢包率：指数据包从源端传输给目的端的过程中丢失的数据包个数与发送的数据包总数的比值。网络丢包一方面会使用户的体验下降，另一方面大量丢包造成的数据重传会加大网络的负担。

时延：指数据从发送到接收所用的时间，常用单位 ms。由于网络的复杂性，造成时延的原因有很多，例如，数据传输到对端会经过一段时间，这是传输时延；数据到达交换机时，经过一系列处理才能到达出端口，这是处理时延；数据到达交换机的出端口时，往往不能立即从端口出去，要经过排队才能出端口，这是排队时延。

2.1.2 QoS 服务模型及机制分析

IETF 提出了许多 QoS 服务模型和机制，包括尽力而为服务模型（Best-Effort）、综合服务模型（IntServ）、区分服务模型（DiffServ）、多协议标签交换（MPLS）和 QoS 路由。

Best-Effort 模型以尽力而为的方式传输数据包，谁先到达出端口，谁先被转发出去，当传输网络出现拥塞时，将会把排在末尾的数据包丢弃以缓解拥塞，Best-Effort 模型无法有效地保证传输业务的服务质量。

IntServ 模型保障业务的传输是通过为特定业务预留充足的带宽实现的[26]，但预留带宽需要网络中的设备都能实现预留带宽这一功能，从扩展性方面来看，相对较差。

DiffServ 模型在业务流进入网络边缘设备时就执行分类和标记的操作，通过将不同的业务流映照到设定的 QoS 等级值上实现对业务流的分类和标记[27-28]，网络中的其他设备根据已有的 QoS 标记值和分类完成数据的转发，因此只需要边缘网络设备具备数据标记和分类等功能就能实现业务的分类转发。DiffServ 对 IntServ 存在的扩展性差的问题给予了解决，但其架构中缺少端到端的通信协议，对于保障端到端的 QoS 还存在一定的问题。

MPLS 对数据的转发不是基于目的 IP 地址而是利用了标签交换技术[29]，应用 MPLS 能解决网络中的流量工程问题，即将流量分到多条空闲的路径中传输以实现链路的负载平衡。

QoS 路由能为进入网络中的业务提供满足其 QoS（时延、带宽等）需求的传输路径，同时能有效整合网络中的资源。

2.2 传统网络下的流媒体传输分析

传统网络在传输流媒体数据时主要存在以下几个问题：一是流媒体数据量远超出普通数据，对于已臃肿不堪的网络来说，传输大量的流媒体资源会增加网络的负担，为了解决流媒体在网络中的传输问题，在传统网络中加入更多的协议及功能，致使网络变得更加复杂，这样的恶性循环使得网络会有崩塌的危险；二是与其他类型数据相比，媒体类数据的处理往往比较复杂，例如，服务端和客户端需要进行编解码的工作，为了保证数据高质量的传输，还需要在数据传输的过程中运行一些特殊的协议，这些额外的操作都会对网络设备产生影响；三是现在的网络结构不能保证有效的 QoS，因为尽管许多 QoS 模型和机制能在一定程度上实现了网络层的 QoS，但由于向网络设备中引入了很多复杂的功能，导致更换网络设备的成本较高，较好地实现端到端 QoS 的目标依然任重道远。

当流媒体视频流在网络中传输时，网络不能满足其最低的 QoS 要求，会出现以下情况：若传输过程中的时延差即时延抖动较大会造成接收端视频的抖动幅度较大，造成视觉上的不舒服，影响观看；若传输过程中出现特别严重的拥塞导致部分数据包无法到达接收端，整个视频的播放效果就是有的地方清晰，有的地方不太清晰；若拥塞导致数据包到达对端耗时较长，则会造成视频播放的卡顿。

2.3 SDN 相关技术

如何解决传统网络存在的瓶颈问题，学术界一直都是各种声音此起彼伏，一些学者认为现有网络架构已经非常成熟，设计并应用全新的网络架构并不现实；一些学者则认为现有网络架构的弊病实在太多，要改变现状必须设计新的网络架构，以便能够全面提升网络的性能，为此设计出一些新的网络架构，其中内容中心网络（Content Centric Network，CCN）[30]和 SDN 网络的研究较为热门。CCN 网络关注的是数据的内容，数据的传输同样也是根据内容而不是传统网络的 MAC 地址或 IP 地址，而 SDN 则是将耦合的控制层和数据层解体，使数据层从繁重的工作中解脱出来，只专注于数据的转发。与 CCN 的替换网络层相比，SDN 对网络的改变仅仅是分离了控制层和数据层，相对也更容易实现，本文以 SDN 网络为研究基础。

OpenFlow 概念出自斯坦福大学 Nick McKeown 教授主导的 Clean Slate 项目，Nick McKeown 教授等人为使校园网络研究有一个比较合适的验证平台提出了 OpenFlow 网络，在之后的不断发展中，SDN 的概念应运而生[31]，并在网络领域中引起了较大的反响，越来越多的国家和网络研究者开始重视并深入地研究 SDN。

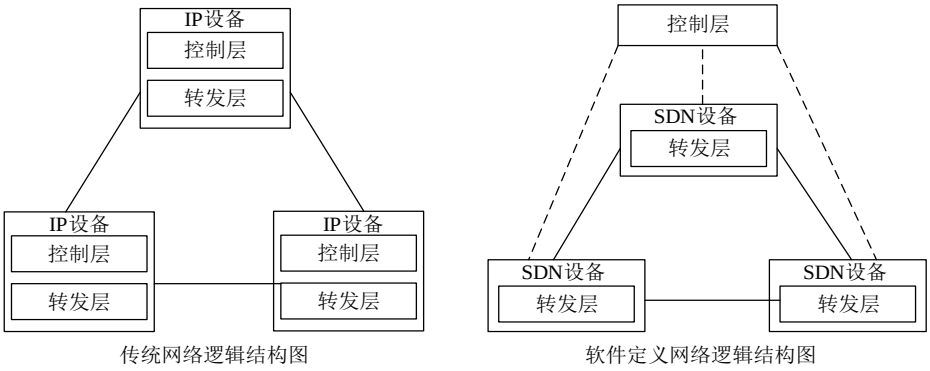


图 2.1 传统分布式结构与 SDN 架构对比图

如图 2.1 所示，传统网络逻辑结构中的控制层和转发层紧紧连接在一起，而软件定义网络的控制与转发是分开的，由控制层控制转发层中所有网络设备的行为。当前通用的 SDN 架构由 ONF 提出，由基础设施层、控制层和应用层组成[32]，如图 2.2 所示。

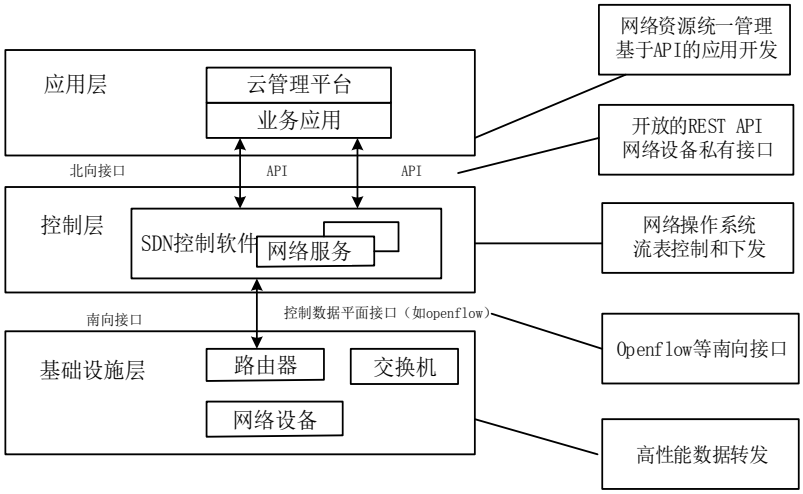


图 2.2 SDN 架构图

控制层是控制平面的抽象，其是 SDN 网络的核心，控制层中的控制器负责收集网络信息并作出控制决策。基础设施层是数据转发平面的抽象，位于该层的转发设备只负责根据控制器的指令传输数据，这与传统网络设备是非常不同的，该层的转发设备除了可以是硬件设备，还可以以软件的形式存在，如 OpenvSwitch。应用层是应用平面的抽象，其通过北向接口，通过控制层的控制器，向基础设施层的网络设备发出控制指令以及进行相关的管理工作。南向接口是控制器和底层网络设备间信息交换的通道，通过该接口，控制器可以获得底层网络的任何信息，并根据收集到的信息对网络设备进行控制和管理。北向接口的作用是实现上层应用与控制器间的通信，对于网络操作人员来说，因为该接口的存在，并不需要了解底层网络的具体信息，只需编写相应的控制程序并通过该接口将命令传递到网络中即可控制网络的行为。

SDN 转控分离的思想使转发设备从繁重的工作中解脱出来，专注于数据的转发工作，相应的控制决策由控制器来完成。同时 SDN 出现伊始就被定义为开放性的架构平台，相比传统网络中修改某个功能的复杂性，软件定义网络的可编程性使得网络的修改操作变得非常方便，大大提升了网络的性能。

2.3.1 SDN 控制器

SDN 控制器位于控制层，由上文介绍可知，其是 SDN 网络的核心。向上，控制器能为上层应用提供底层设备的各类信息；向下，控制器能向底层设备发送一系列指令实现网络的动态管理。随着 SDN 在网络领域逐渐展露头角以及控制器核心作用的凸显，业界已发布了多款控制器，表 2.1 是几个功能和性能较好的控制器[33]。

表 2.1 功能和性能较好的控制器

控制器	编写语言	特点
NOX	C++/Python	第一个 SDN 控制器
POX	Python	使用 Python 编写的 NOX
Beacon	Java	性能稳定，提供基于 WebUI 的管理控制
FloodLight	Java	企业级控制器，对 Beacon 的进一步改进
Ryu	Python	提供多种服务，易于开发
Meastro	Java	跨平台，易部署，支持多线程
NodeFlow	Javascript	极简化的控制器，用于编写可扩展的应用

Ryu 是由日本 NTT 公司贡献的基于组件的开源 SDN 框架，组件以一个或多个线程的形式存在，通过组件提供的接口可方便控制组件状态和事件的产生；提供图形用户界面便于掌握网络拓扑的连接情况，模块清晰，可扩展性好；采用语法简洁的 python 语言开发，易于学习和部署新的功能；能较好地支持 OpenFlow1.0 版本到 1.5 版本[34]。本文的 QoS 路由功能就是在 Ryu 中编写相应模块实现的。

2.3.2 OpenFlow

OpenFlow 协议对转发设备中的流表以及 SDN 控制器和转发设备之间的交互消息进行了规范的定义。

1）OpenFlow 交换机

OpenFlow 交换机中的 OpenFlow 通道是控制器与交换机之间进行通信的关键，其影响通信的可靠和安全，可靠性和安全性分别由使用 TCP 连接和安全传输层协议保障。OpenFlow 交换机中的流表及组表是数据转发的指引者，当数据进入到 OpenFlow 交换机

中，若能匹配上交换机中存在的流表，就按照流表的指示执行相应的动作。图 2.3 是 OpenFlow1.3 定义的 OpenFlow 交换机的结构。

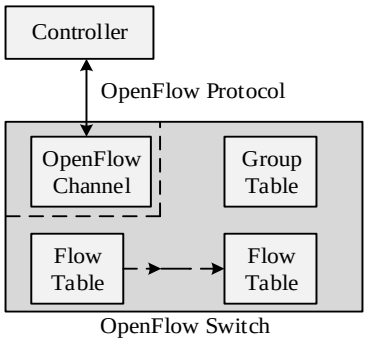


图 2.3 OpenFlow 交换机网络结构

OpenFlow 的流表与传统网络中的 MAC 地址转发表和 IP 路由表类似，都是用于指引数据的传输，但流表项的匹配域中包含了网络一层到四层的各个属性，对网络中匹配属性的整合使得转发设备对网络中数据流的匹配更加精确，对数据流的操作也就更加细粒度。交换机中每个流表都由多个流表项组成，每个流表项都由多个字段构成，例如匹配域、优先级以及超时等字段，OpenFlow1.3 规定的流表项结构如图 2.4 所示。

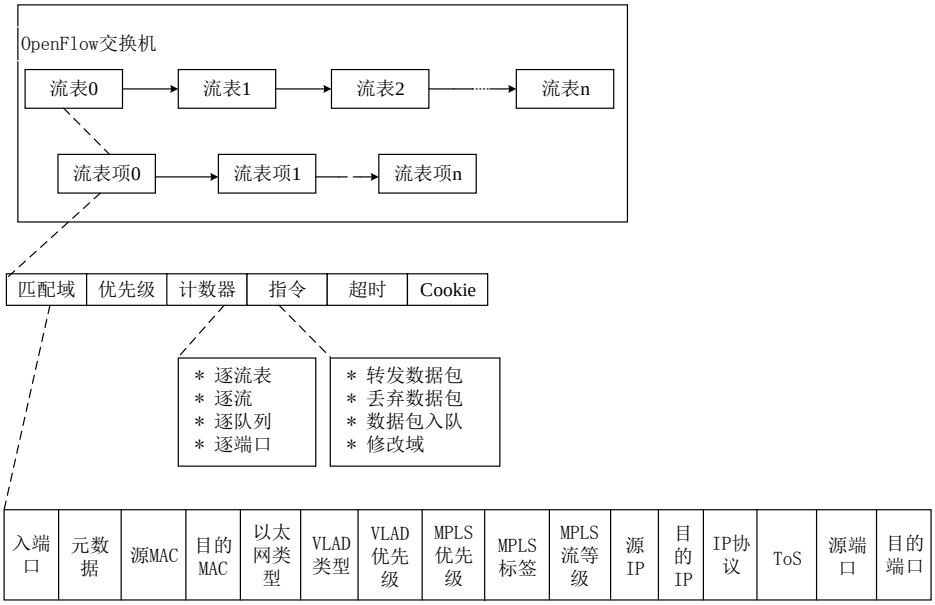


图 2.4 流表项结构图

匹配域用来判断数据包与流表项是否匹配，从图 2.4 可知匹配域由 16 个字段组成，涵盖了 1-4 层的各个匹配项，匹配规则灵活，对于每个元组，可匹配精确值，亦可匹配任意值，更可匹配某一范围内的值。优先级定义了流表被匹配的顺序，OpenFlow 定义中规定该值越高，优先级越高。计数器（counters）的作用是维护一些统计信息，比如某端口

接收与发送的字节数或某个流的统计信息。

超时是用来解决每个流表项在交换机中存活时间的问题，超过设定的时间，流表项就会被自动删除，超时字段中包括 idle timeout 和 hard timeout，分别为设置流表项存活的时间和硬超时的时间。Cookie 用于控制器对数据流进行操作，包括流删除、流改变等操作。

指令是解决数据包“去向”的字段，包括转发（Forward）、丢弃（Drop）、排队（Enqueue）、修改域等指令，流表项中的指令可以包含前述指令中的几个，也可以没有任何指令，但包含多个指令的前提是必须包含转发指令，其是必备项，否则数据包就会被丢弃。

转发是将数据包送到某个端口的动作，转发的方式有很多，包括 IN_PORT、CONTROLLER、ALL 和 NORMAL 等方式，分别表示从入端口转发出去；转发给控制器；转发给除入口端外的其余端口；正常地转发，按照 MAC 地址或 IP 地址等转发数据包。

丢弃是指不转发数据包而是将其丢弃的动作，一般这种情况就是数据包未能匹配上交换机中的任何流表，且交换机中没有 table-missing 流表项，或是有 table-missing 流表项，但其指令是丢弃。

排队是指控制器按照设置好的队列 ID，将一类数据包放到出端口指定队列 ID 中的操作，排队与队列调度策略有关。

修改域（Modify-Field）是对一些字段进行修改的操作，其能修改的字段有很多，但最常用的是对 IP 报文头中的的 ToS（服务类型）字段的前六位即 DSCP 值进行修改，修改成自己设定的值。根据该项操作，可实现数据包分类后的同类数据包的标记，进而实现对同类数据进行相同操作的功能。

SDN 网络中，交换机的行为由控制器下发的流表决定，OpenFlow1.3 采用多级流表的结构，交换机里存在控制器下发的多个流表。OpenFlow 交换机对数据流的处理过程如图 2.5 所示，当交换机接收到外部发来的数据包后，首先对数据包头信息进行解析；之后按着流表的优先级顺序依次进行匹配，若匹配到某个流表中的流表项，就执行该流表项的动作；若数据包在交换机中匹配失败，则根据设定的规则进行操作，或是丢弃该数据包，或是携带数据包信息转发到控制器上，由控制器决策如何处理该数据包。

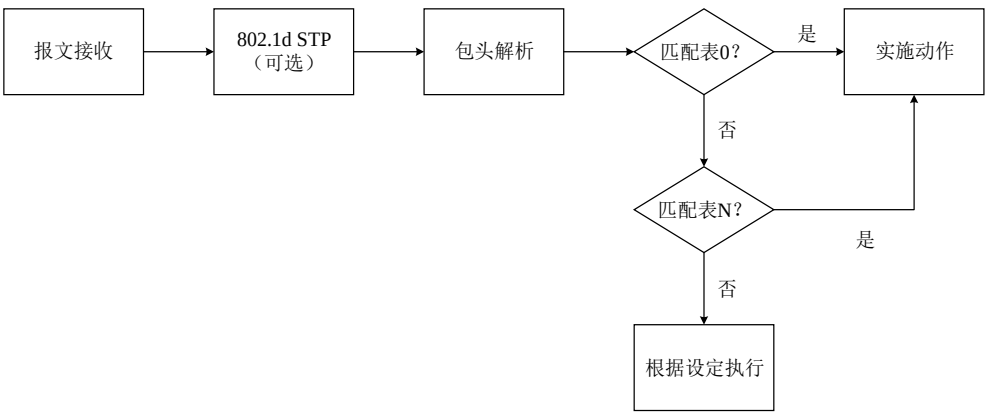


图 2.5 流表匹配流程图

2）OpenFlow 协议

OpenFlow 协议支持三种消息类型用以实现控制器和交换机之间的信息交流，包括控制器-交换机消息（Controller-Switch）、异步消息（Asynchronous）和同步消息（Symmetric）。

Controller-Switch 消息是控制器对交换机状态进行查询的消息，通过发送相应的 Request 消息，对收到的回复消息进行分析，根据分析得到的结果能获得交换机的状态信息。例如，通过 modify-state 消息查询交换机的端口状态（port-mod）和流表项（flow-mod）等信息。

Asynchronous 消息用于交换机向控制器告知网络的变化，不需要控制器必须回复。Asynchronous 消息包括 packet-in 和 ports-status 等消息，进入交换机中的数据包未能匹配上任何流表，就会触发 packet-in 消息携带数据流首个数据包的部分信息或是携带整个数据包的信息发送到控制器，控制器通过分析数据包的信息，得到数据流的整体信息，进而作出相应的控制决策。当网络中加入新的交换机或原有的交换机断开连接时，交换机会向控制器发送 ports-status 消息，控制器得到该消息后对底层网络拓扑连接信息进行重新整合，得到最新的拓扑连接信息。

同步消息用于确认控制器和交换机的连接状态，其发送方可以是控制器也可以是交换机。例如通过 hello 消息确认控制器和交换机是否连接上了，通过 echo 消息确认控制器和交换机连接的状态。

2.3.3 OpenFlow 对 QoS 的支持

QoS 的控制机制是指根据 QoS 流的状态，对其进行实时的控制。前文提到 SDN 与传统网络不同的特点，其可在控制平面上实现复杂的 QoS 控制逻辑，实现全局性的调度。

OpenFlow1.0 有一个可选动作 enqueue（1.1 版本后叫 set queue），其能将数据包放入到指定的交换机队列中。

OpenFlow1.1 中添加了 VLAN 等操作，在 1.1 版本之后，通过对 VLAN 标签执行添加、修改与去除等操作可实现 QoS 控制。

OpenFlow1.2 协议中添加了一个消息，使得控制器可以对交换机中的所有队列进行查询。

OpenFlow1.3 中使用 Meter 引入新的限速机制，Meter 表由多个 Meter 表项组成，结构如图 2.6 所示。其中 Rate 值与限速机制密切相关，其值若小于 Meter 的速率，Meter 就会采取相应的措施，要么将超过 Rate 值的数据丢弃，要么对 DSCP 重新标记来对优先级进行重新的排列。

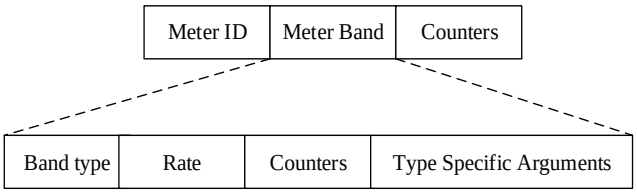


图 2.6 Meter 表项组成图

OpenFlow1.4 中增加了流量监控模块，当其他控制器对流表进行操作时，未操作流表的控制器会得到该消息和操作的结果。

2.3.4 智能算法在 SDN 上运行的可行性及 SDN 特性分析

将智能算法应用到 SDN 网络中用以解决流量问题和流媒体类业务 QoS 问题是切实可行的措施。对于流媒体类业务的传输来说，要综合且独立地考虑其每个 QoS 的需求，这类问题一般通过智能算法来解决，且无论哪种智能算法，解决此类问题的前提都需要获取网络底层资源的使用情况，以便根据全局信息对网络资源进行重新分配。基于上述分析，网络全局信息的获取尤为重要，而 SDN 转控分离、集中控制以及可编程的特性能够较好的实现全局信息的收集，为智能算法的决策提供了很好的参考，使智能算法在 SDN 上运行是可行的。

SDN 集中控制的特点主要体现在底层网络信息的收集上，控制器通过与交换机特定的连接链路获取全网的信息视图，集中控制的形式对于信息的收集来说是非常高效和准确的，根据收集到的全局信息可为业务流计算符合其传输需求的路径。

SDN 中对数据的分析和处理都是基于流的，根据控制器能够获得一条流详细信息的特点，可以实现对一条数据流更加灵活的控制，例如对某条视频流的处理。控制器还能周期地监测网络的拥塞，在网络拥塞时根据获得的网络信息为数据流提供动态路由的机制，具有较好的时效性。

SDN 的开放性和可编程性使每个人都能成为网络应用的开发者，设计特定程序模块以实现不同的功能，例如为了获取链路性能参数信息而设计的链路性能测量模块。

综合以上分析，SDN 架构的优越性使智能算法能够更好地发挥其优势，SDN 的特性使其能较好地解决视频流媒体 QoS 需求的问题。

2.4 本章小结

本章开始先对 QoS 技术进行了分析，主要是 QoS 概念的介绍以及 QoS 服务模型和机制的分析对比。之后对传统网络下视频流媒体传输遇到的困境进行了分析。最后对 SDN 技术进行了分析，主要是对 SDN 网络的运行机制、智能算法在 SDN 上运行的可行性以及 SDN 特性进行了阐述和分析。下一章将以本章内容为基础展开，借助 SDN 网络的优势，提高视频流媒体的传输性能。

3 QoS 控制策略需求分析与设计

3.1 需求分析

3.1.1 应用环境分析

SDN 网络中，所有应用的实现都是依靠控制器上扩展的相应模块，因此需要对 SDN 架构以及控制器的功能做以深入的研究。

本文在控制器上扩展相应模块来实现 QoS 路由的功能，如拓扑管理、链路性能信息测量等模块，但扩展出来的模块不能单独使用，需要与控制器中一些基本模块配合才能实现 QoS 路由功能，如协议适配、事件机制等模块。队列调度功能属于底层应用，摈弃固有的先进先出的队列模式，采用 HTB 队列调度规则，使视频流优先出队列，既能够更好地保证流媒体视频流的 QoS，也能兼顾其他业务优先级流的 QoS，因此需要注意 OpenFlow 流表中关于队列的操作及 OpenFlow 交换机上队列规则的配置。

3.1.2 应用需求分析

实现 QoS 控制策略方案的目的是为了保证视频流媒体的服务质量，同时兼顾其他优先级数据流的传输。本文的控制策略包括控制层面与数据转发层面两方面，控制层的策略是当有数据初进入网络时，对于视频流媒体，根据全局的网络拓扑和所监测的网络链路信息，使用基于遗传算法的 QoS 路由算法进行路由，对于其他优先级的业务流则根据经过的交换机跳数选择传输的路径。当数据流在网络中传输时，控制器对数据流的传输路径进行实时的监控以便能获得传输路径的状态，当监测到传输路径的前方发生拥塞时，及时为视频流采取动态重路由的策略。数据转发层的策略是根据控制器标记的流类型和交换机上配置好的队列调度策略，将业务流放入到端口不同的队列中，优先调度流媒体视频流，使其能得到更好的 QoS 保障。综上所述，控制策略整体的需求包括链路状态信息的获取、视频流传输路径的计算、其他业务流传输路径的计算、动态路由的路径选择、不同类型业务流的划分以及队列调度策略等几个方面，因此需要考虑并解决以下问题：

1)、网络拓扑连接信息的获取：控制策略的有效实施有赖于全网拓扑连接情况的准确获取，由于网络中的数据传输和拓扑连接都是动态的，需要实现对全网拓扑连接状态的动态获取，当有交换机或链路出现故障时，及时对网络拓扑结构进行整合更新，以确保路径选择时有实时的拓扑连接参考。

2)、链路性能信息的收集：本文方案中需要选择符合时延和丢包率指标的路径进行数据的传输，因此必须设计性能信息获取类模块以便准确并及时地获取网络中每条链路的性能指标信息。

3)、视频流服务质量的保证：为了满足视频流的服务质量，需要对数据流进行划分，

区别对待流媒体视频流和其他业务流，为视频流提供满足其个性化要求的最优路径，与此同时，也需要在端口队列调度的过程中对不同类型的业务流进行差别调度，保证视频流优先出端口。

4)、动态路由策略的有效性：动态路由策略能及时为视频流媒体计算出新的传输路径，在较短的时间内恢复数据的高效传输，且在下发流表时注意并解决新旧流表转换带来的时延问题。

3.2 QoS 控制策略框架

3.2.1 数据流分类

在线视频、视频监控等流媒体业务对时延、丢包率等 QoS 性能参数有着较高的要求，为满足此类业务的传输需求，并能充分利用网络资源，必须将业务流按照优先级进行分类，以便能够对分类后的业务流进行有区别的转发控制。

为了能对不同种类业务数据包标记对应的优先级值，本文利用 DiffServ 模型中的 DS 域实现对业务优先级值的存储[35]，即对数据报文头中的 ToS 字段中的 DSCP 值进行修改，不同的值代表优先级不同，值越大，优先级越高。

根据 IETF 定义的应用类型的优先级并结合本文的研究内容，对各种应用的优先级分类以及 DSCP 值的设定如表 3.1 所示，Q1 表示对 QoS 有最高需求的实时视频流媒体业务，Q2-Q4 表示对 QoS 需求依次降低的业务。

表 3.1 应用的优先级分类及 DSCP 值的设定

应用类别	协议举例	QoS 优先级	DSCP 值
在线视频、视频监控	RTP,RTSP	Q1	100000
关键数据业务	NFC,SMB,RPC	Q2	011000
事物处理业务，交互式数据	SQL	Q3	010000
数据同步业务，e-mail	FTP,SMTP	Q4	001000

通常根据数据包头中的源目的 IP 地址、四层协议、源目的端口号等字段对流进行分类，并使用特定的 DSCP 值对分类后的数据流进行重新标记。若想要更细致地分类，可以对数据包进行进一步的解析，但一般并不需要，上述的字段即可区分出网络中大部分的数据流，本文也是使用这个方法对业务流进行分类。

与在传统网络结构中应用 DiffServ 网络模型不同，SDN 转控分离的思想使得数据包的标记动作（标记不同的数值以区分不同类别的业务）并不由交换机来完成，而是利用控制器下发的流表项中的修改域指令，修改域指令能够将 IP 报文头中的 ToS 字段的 DSCP

值进行修改，修改成自己设定的值。在控制器下发的流表项中设置 DSCP 值修改的规则，规定匹配到该流表项的数据流的 DSCP 值作出对应的更改。DSCP 值修改的动作需要在边缘交换机上完成，进入网络中的数据包在边缘交换机上完成 DSCP 值修改的操作，使得其他交换机在数据流转发的过程中能够根据数据包头中的 DSCP 值进行相应的流分类和标记操作。

3.2.2 QoS 控制策略设计

由上节介绍可知，本文依据业务类型将数据流分为四个不同的优先级，视频流媒体业务对时延等 QoS 参数有着非常严格的要求，因此优先级最高，其他业务的优先级则依次降低。为了在多种数据流并存的网络中满足视频流媒体的 QoS，并减少对其他优先级业务流的影响，本文利用 SDN 控制器能获取全局网络拓扑信息并能集中控制 OpenFlow 交换机中的业务数据流转发行为的特性，以及 OpenFlow 协议提供队列服务的特点，提出了基于业务优先级的视频流媒体 QoS 控制策略的框架。该 QoS 控制策略能够满足视频流媒体端到端的服务需求，并能在网络出现拥堵时及时调整传输路径，减少拥堵造成的视频流媒体传输性能变差的问题。

QoS 控制架构如图 3.1 所示，由 QoS 路由与队列调度组成，这两个部分在 SDN 控制器与 OpenFlow 交换机中都有着对应的模块，各个模块的功能及设计在下一节进行详细的阐述。

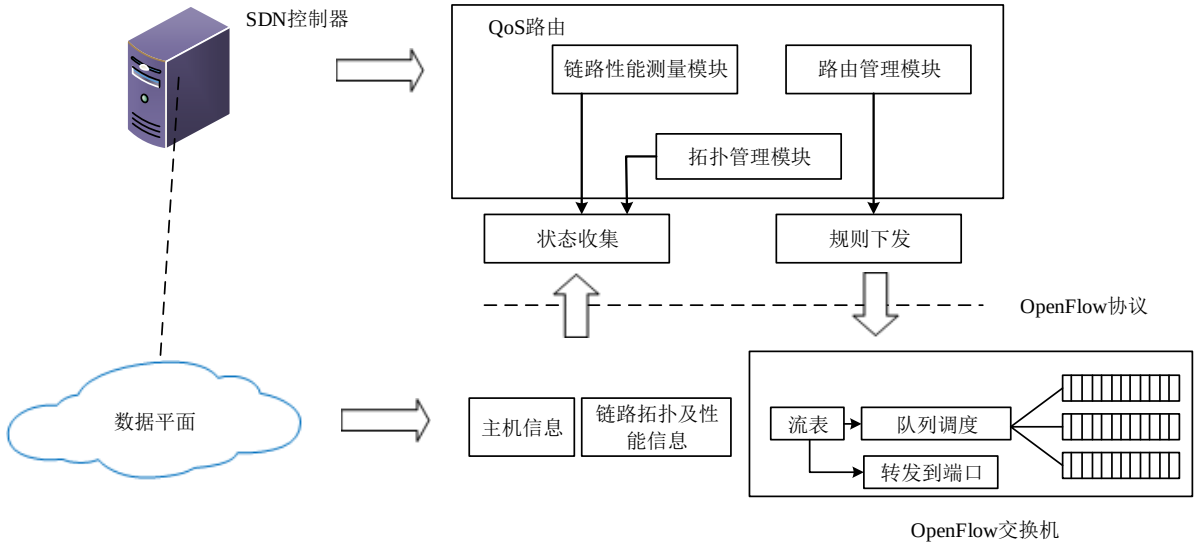


图 3.1 QoS 控制架构图

图 3.2 为 QoS 控制策略框架执行流程，进入交换机中的业务流没能匹配上交换机中的任何流表，就会将数据包信息传递给控制器，由控制器对数据包进行解析，对视频流媒体数据流按照 OpenFlow 控制器所规定的 QoS 约束条件，依据全局的网络拓扑和所监测到的网络链路性能信息，基于遗传算法进行 QoS 路由，对其他优先级业务则使用以路径

跳数为代价的 Dijkstra 算法进行路由。之后下发流表并修改 ToS 字段的 DSCP 值以及对数据包进行入队操作，然后使用 HTB 队列规定对 OpenFlow 交换机出端口队列中的数据包进行调度。

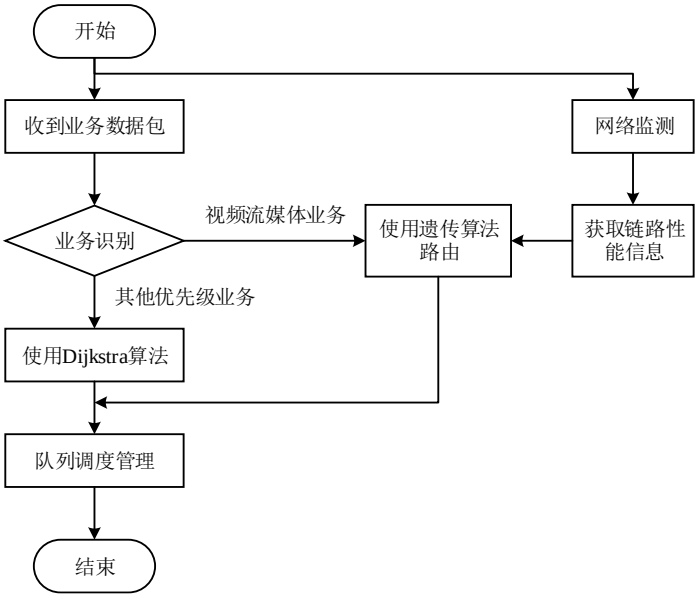


图 3.2 QoS 控制策略框架执行流程

3.3 QoS 控制策略框架模块分析

QoS 控制策略框架分为 QoS 路由和队列调度两个部分，下面分别对这两部分进行阐述。

3.3.1 QoS 路由框架

QoS 路由旨在利用 SDN 的可编程性以及可获得全局网络信息的特点，基于遗传算法为视频流媒体计算满足其 QoS 需求的路径，并能动态地保障其 QoS。QoS 路由框架包括拓扑管理模块、链路性能测量模块以及路由管理模块，其设计充分利用了 SDN 的特点。

1）拓扑管理模块

控制器中的拓扑管理模块主动对网络拓扑进行探测，以获取网络中交换机的物理地址、端口号以及交换机之间的连接等信息，除了交换机之间的链路信息，拓扑管理模块还能获得主机与交换机之间的链路连接情况。由于网络环境以及交换机间的连接并不是固定不变的，模块需要周期性地下发获取网络拓扑连接的消息，以确保提供给决策模块的拓扑信息是最新的。

另一方面，模块会从链路性能测量模块中获取每条链路的时延、丢包率、带宽使用情况等性能信息，根据这些信息以及物理连接信息，拓扑管理模块以带权值的邻接链表表示整个网络，该邻接链表提供的数据是 QoS 路由管理模块计算传输路径的基础。

拓扑管理模块中的拓扑连接信息的获取流程如图 3.3 所示，其中网络设备即网络中连接的交换机或者路由器；信息采集模块对交换机的信息进行统计（交换机 datapath ID，端口号 port 等）；信息分析模块对信息采集模块获取的信息进行相关运算，计算网络的拓扑连接；状态轮询模块则是根据单独提出的子线程周期执行信息采集模块，在网络拓扑由于交换机接入或离开造成拓扑连接发生变化时，对拓扑连接信息进行更新；拓扑图显示模块根据信息分析模块计算得到的拓扑连接信息，将拓扑信息展示在终端。

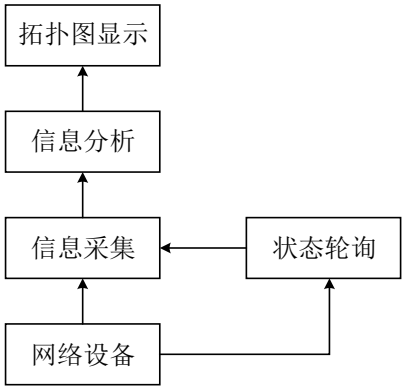


图 3.3 拓扑发现模块图

与传统网络一样，SDN 网络中获取链路的连接信息也是利用了链路层发现协议（Link Layer Discovery Protocol，LLDP），但与传统网络分布式的获取形式不同，SDN 网络中拓扑信息的获取是由控制器统一完成的。LLDP 帧的组成如图 3.4 所示。

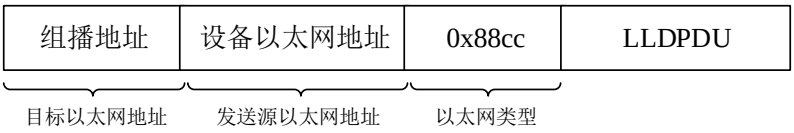


图 3.4 LLDP 以太网帧

LLDP 以太网帧中的 LLDPDU 为有效荷载，其能够存储设备的物理地址等信息。LLDP 能够将设备标识以及物理地址等信息发送给邻居网络设备，邻居设备接收到对端发送的 LLDP 帧后进行解析即可得到存储在 LLDPDU 中的信息并将其保存下来。

控制器执行信息采集的过程如图 3.5 所示，为了精准掌握网络中交换机之间的连接情况，控制器构造含有 LLDP 数据包的 packet-out 消息，将此消息发送给网络中所有的交换机，packet-out 消息中包含如何处理 LLDP 数据包的流表。以 s1 和 s2 组成的链路为例，s1 收到控制器发来的 packet-out 消息后，通过 packet-out 中的流表项规则处理 LLDP 数据包，流表项规则是 s1 将控制器发来的 LLDP 数据包从非接收端口发送出去；s2 的 port1 与 s1 的 port1 相连，因此 LLDP 数据包会从 s1 的 port1 转发出去，发送给 s2 的 port1；收到 LLDP 数据包的 s2 无法处理该数据包，便会触发 packet-in 消息，其将携带 LLDP 数据包的信息到控制器；控制器收到 packet-in 消息后对数据进行解析，得到发送该 LLDP 报

文的是 s1 的 port1，接收该报文的是 s2 的 port1，从而确定交换机 s1 与 s2 之间具体的连接情况，即 s2 的 port1 与 s1 相连，同理可知 s1 的 port1 与 s2 相连。依照这样的方式，控制器获得网络中相连两个交换机间的链路信息，进而得到整个网络的链路连接情况。

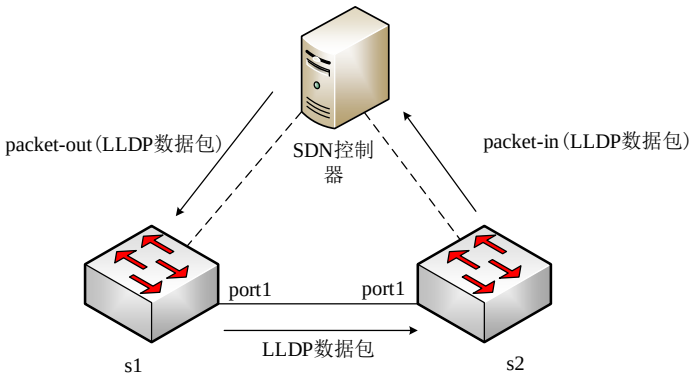


图 3.5 链路连接信息采集图

2）链路性能测量模块

链路性能参数衡量了一条链路传输性能的好坏，QoS 路由的前提是能及时获取链路的这些参数信息，然后根据这些信息计算出最优的传输路径。目前，SDN 中并不能直接得到时延、带宽、丢包率等链路性能数据，因此本文对控制器功能进行扩展，添加了链路性能测量模快，从而高效地对链路的 QoS 性能参数进行测量。下文对链路性能信息的测量原理进行具体的阐述。

a、带宽使用情况及丢包率

OpenFlow 协议定义了交换机中的计数器统计字段，其中对于交换机端口能够查询到的信息如图 3.6 所示。

```
/*  Body of reply to OFPPortStats request. If a counter is unsupported. set the field to all ones. */
Struct ofp_port_stats{
unit32_t port_no;
unit8_t pad[4];                 /*      Align to 64-bits.                                   */
unit64_t rx_packets;            /*      Number of received packets.                         */
unit64_t tx_packets;            /*      Number of transmitted packets.                      */
unit64_t rx_bytes;              /*      Number of received bytes.                           */
unit64_t tx_bytes;              /*      Number of transmitted bytes.                        */
unit64_t rx_dropped;            /*      Number of packets dropped by RX.                    */
unit64_t tx_dropped;            /*      Number of packets dropped by TX.                    */
unit64_t rx_errors;             /*      Number of receive errors. This is a super-set
                                        of more specific receive errors and should be
                                        greater than or equal to the sum of all
                                        rx_*_err values */
unit64_t tx_errors;             /*      Number of transmit  errors. This is a super-se
                                        of more specific receive errors and should be
                                        greater than or equal to the sum of all
                                        tx_*_err values (none currently defined)    */
unit64_t rx_frame_err;          /*      Number of frame alignment errors.                   */
unit64_t rx_over_err;           /*      Number of packets with RX overrun.                  */
unit64_t rx_crc_err;            /*      Number of CRC errors.                               */
unit64_t collisions;            /*      Number of collisions.                               */
unit32_t duration_sec;          /*      Time port has been alive in seconds                 */
unit32_t duration_nsec;         /*      Time  port has been alive in nanoseconds
                                        beyong duration_sec                                 */
};
OFP_ASSERT(sizeof(struct ofp_port_stats) == 112);
```

图 3.6 端口消息结构图

通过对交换机端口进行查询可以得到端口发送与接收的字节数，并能获取到端口发送与接收这些数据所经过的时间，对以上信息进行整合计算便得到每个交换机的每个端口的带宽使用情况，之后通过将一条链路两端端口对应起来并分析计算，即能得到一条链路的带宽使用情况，端口带宽使用情况的计算如以下内容。

若控制器利用 OFP_Port_Stats 消息在 t1 时刻与 t2 时刻取得了 s1 与 s2 对应端口的（s1 的 port1 与 s2 的 port2 相连）信息，对于交换机 s1 的 port1，其 t1 时刻与 t2 时刻发送与接收的字节数如公式(3.1)和(3.2)，端口速率的计算如公式 (3.3)所示， *speed* 单位为 bit/s。

$$p_{t1} = \left(tx\_bytes_{t1} - rx\_bytes_{t1}\right) \tag{3.1}$$

$$p_{t2} = \left(tx\_bytes_{t2} - rx\_bytes_{t2}\right) \tag{3.2}$$

$$speed = \frac{p_{t2} - p_{t1}}{t2 - t1} \times 8 \tag{3.3}$$

则交换机 s1 的 port1 的已用带宽和剩余带宽如公式(3.4)和(3.5)所示，*curr* 为端口固定带宽，单位为 Mbit/s。

$$bw_{use} = speed \tag{3.4}$$

$$free_{bw} = curr - \left(bw_{use} \div 10^6\right) \tag{3.5}$$

链路拥塞率是衡量业务数据流在网络中传输拥塞程度的指标，拥塞严重性随着拥塞率的增大而增大。链路拥塞率的计算如公式(3.6)所示，其中 *speed* 为通过对一条链路两端口进行对应分析得到的链路已用带宽， $\frac{speed}{10^6}$ 为单位换算即 bits/s-Mbit/s。

$$link\_congestion = \left(\frac{speed}{10^6}\right) \div curr \tag{3.6}$$

与端口统计类似，OpenFlow 定义了统计流信息的消息结构体 OFP_Flow_Stats，其能够获取交换机中匹配到某个流表项的流的统计信息，包括 duration_sec、packet_count、byte_count 和 instructions 等字段，分别表示统计的时刻、流表项匹配到的报文数、流表项匹配到的字节数和匹配到该流表项的数据包会被采取的动作。

链路丢包率的测量利用了 OFP_Port_Stats 消息中的 $rx\_packets$ 字段、OFP_Flow_Stats 消息中的 instructions 字段和 packet_count 字段，利用 OFP_Port_Stats 消息得到某个端口接收到的报文总数，利用 OFP_Flow_Stats 消息获得当前交换机某个端口匹配到流表项发出的报文总数。对于 s1 与 s2 间的链路（s1 的 port1 与 s2 的 port2 相连），利用 OFP_Flow_Stats 消息可得到 t1 时刻 s1 所有流表项统计的发送到 port1 的报文总数为 $tx\_packets\_byflow_{t1}$，同理 t2 时刻为 $tx\_packets\_byflow_{t2}$，t1 时刻到 t2 时刻，s1 传输给 s2 的报文数如公式(3.7)所示。

$$tx\_packets = tx\_packets\_byflow_{t2} - tx\_packets\_byflow_{t1} \tag{3.7}$$

根据 OFP_Port_Stats 消息中的$rx_packets$字段得到 t1 时刻 s2 从 port2 端口接收到的报文总数$rx_packets_{t1}$，同理，t2 时刻接收的报文总数$rx_packets_{t2}$，t1 时刻到 t2 时刻，s2 的 port2 实际接收的报文数如公式(3.8)所示。

$$rx\_packets = rx\_packets_{t2} - rx\_packets_{t1} \tag{3.8}$$

丢包率pl的计算如公式(3.9)所示。

$$pl = 1 - \frac{rx\_packets}{tx\_packets} \tag{3.9}$$

b、时延

OpenFlow 协议实现时延的测量是利用了 SDN 网络中可在某些消息中添加时间戳的特点，通过按照对应的计算公式对时间戳进行计算，即可得到链路间的时延信息。本文根据这一特点，利用 packet-out 和 packet-in 等消息间的联系，使消息中携带时间戳，通过时延计算公式得到每条链路的时延信息。

时延测量的具体步骤如下：

（1）控制器经过多个交换机再回到控制器的时延测量是利用了控制器为获得拓扑连接信息而下发的携带 LLDP 数据包的 packet-out 消息。以图 3.7 两个交换机由控制器控制的链路为例，使 packet-out 消息的 data 域携带有控制器发出消息的时间戳T_{lldp_start}，之后与获得拓扑连接的操作一样，当控制器收到 packet-in 消息后会记录此时的时间T_{lldp_end}，$T1 = T_{lldp\_end} - T_{lldp\_start}$ 即是 controller-s1-s2-controller 的大致时延，同理，controller-s2-s1-controller 的时延$T2$计算方式也如此。

（2）控制器到交换机的往返时延测量利用了请求回复消息 EchoRequest，使 Echo Request 消息的 data 域携带有控制器发出消息的时间戳T_{echo_start}，控制器向 s1 发送 Echo Request 消息后，会收到 s1 的回复消息 EchoReply，控制器收到 Echo Reply 消息后会记录此时的时间T_{echo_end}，控制器与 s1 往返时间$T_{rrt_1} = T_{lldp_end} - T_{lldp_start}$，同理，控制器与 s2 之间往返的时间$T_{rrrt_2}$计算方式也是如此。

（3）s1 与 s2 之间的时延计算公式如(3.10)所示。

$$T = \left(T1 + T2 - T_{rrt\_1} - T_{rrrt\_2}\right) \div 2 \tag{3.10}$$

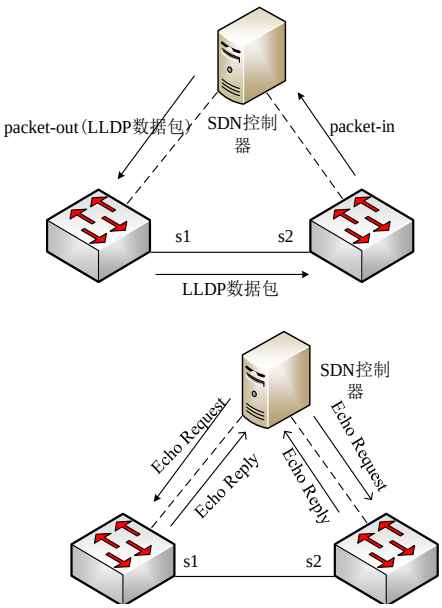


图 3.7 时延测量原理图

3）路由管理模块

路由管理模块是 QoS 路由策略中最重要的模块，其为不同业务流计算传输的路径。路由管理模块有四个具体功能，一是数据流的区分，二是路径的计算，三是流表下发，四是网络拥塞时的动态路由。数据流的区分主要是区分出视频流媒体与其他业务流，进而给予不同的路由策略；路径计算部分采用了双轨并行的方式，根据拓扑管理模块提供的全局网络拓扑结构以及带有链路性能参数的拓扑结构，对于视频流媒体，以其 QoS 需求为核心为其选择路径进行传输，对于其他低优先级业务，使用以跳数为代价的 Dijkstra 算法进行路径的选择和传输，通过两种不同的路由方式，实现了不同优先级业务的差别路由，在控制层较好的保证了视频流媒体端到端的 QoS；流表下发是根据路径计算部分得到的链路路径建立流表信息并下发给 OpenFlow 交换机，OpenFlow 交换机根据流表进行业务数据流的路由转发。

多 QoS 约束的视频流媒体路由，实际上是满足多个指标的相对优解，对于单约束 QoS 路由问题，最短路径算法 Dijkstra 算法即可解决[36]，多约束路由问题即同时满足两个以上相互独立参数的性能要求目前已被证明是 NP-Complete 问题[37]，很多研究将多种约束参数以相关运算的方式组合成一个混合的参数来解决多约束的问题，但这种方式有很大的弊端，其一是运算的方式并不容易确定，其二是通过混合参数计算出来的路径也许不能较好地反应任何一个约束，计算出来的路径往往毫无意义。基于此，更多的研究寻求以更好的方式解决此类问题，运用近似算法或启发式算法求解多约束问题是众多方法中较为优越的方式，两类算法各有各的优势和劣势，近似算法能保证解的质量较好，但计算却要花费较长的时间；启发式算法能在较短的时间内计算出解，却在保证解的质量上有所欠缺，常用的启发式算法有遗传算法[38]、蚁群算法[39]等。本文需要在传输路径不满足传输要求时，能够尽快地找到另一条可以传输的路径，因此本文研究采用启发式算法中的遗传算法

来解决多 QoS 约束问题。

遗传算法是对生物在自然环境中优胜劣汰的模仿，其是一种全局优化概率搜索算法[40-41]。遗传算法使用迭代的方式对一代种群模拟生物的进化过程，通过一系列遗传操作生成下一代群体，之后不断迭代，直到达到算法结束的条件，找到符合需求的最优解。本文将在下文论述如何设计遗传算法进行 QoS 路由的计算。

由于网络状态一直处于动态地变化中，路径计算部分计算出的路径在一定时间后可能不再满足视频流媒体传输的 QoS 需求，动态路由的功能是根据周期检测到的网络拥塞状况，在发现视频流媒体的传输路径出现拥塞时，为视频流媒体计算新的传输路径，然后将新的路径转换成流表下发到对应的交换机上，对于旧的流表项执行删除的动作。本文假定理想情况为触发动态路由机制从原有路径调整到其他路径时，已经发送的数据包不会重新发送，不再重传的数据流依然需要在拥塞的路径中传输，端口转发会一直使用 HTB 队列规则最大可能减缓拥塞，保证数据流的通过。动态路由部分涉及拥塞链路的判定及流表下发时路径切换的问题，下文为具体阐述。

a、拥塞链路判断

本文的方案是应用于混合流量的网络中，即包含不同优先级的业务流，本文的主要目标是为视频流媒体提供最好的 QoS 保障，但是网络中的传输会出现其他业务流挤占视频流资源的情况，即传输路径出现拥挤，这个时候需要对视频流媒体的传输路径进行调整。本文设定的拥塞率为 80%，即路径带宽的使用率为固定带宽的 80%以上，即判定为拥塞，需要使用遗传算法为视频流重新计算一条合适的路径进行传输。

b、路径切换机制

当触发系统使用动态路由机制为一条流重新计算路由之后，需要将新的路径流表下发到交换机中，使后续进入到网络中的流匹配到新的流表，在新的路径中进行传输。这一过程虽然时间很短，却涉及新老流表共存和路径切换过程中的丢包问题，对于数据的传输是非常重要的，因此需要设计路径切换机制来解决上述问题。

路径切换机制首先解决的是路径切换过程中的丢包问题，路径切换主要是指新流表的下发，而新流表的下发先要解决新流表的优先级问题，数据包在交换机中匹配流表的顺序是根据流表的优先级来确定的，优先级值越大，优先级越高。若新路径流表的优先级低于旧路径流表的优先级，则数据包匹配到的依然是旧的流表，传输的路径还是原来的路径，并没有改善数据流的传输情况；若新旧流表优先级一致，进入网络的数据包不知该匹配哪个流表，数据包的传输会存在一定的风险；若新路径流表的优先级高于旧路径流表的优先级，进入交换机中的数据流会匹配新的流表，以新的路径进行传输。因此新流表优先级的设置必是要高于旧流表的优先级。

新流表优先级设定完毕后需要对流表的安装顺序进行设置。若以“源-目的”的正向顺序下发流表，则可能出现数据流到达某一交换机时而流表更新还没有完成的情况，此时数据包匹配不上交换机内任何的流表，停在了此处而不能继续传输下去，影响数据端到端

的传输。

基于“正向顺序”的安装可能影响到数据的传输，采用“反向顺序”安装流表，即以“目的-源”的顺序依次下发流表，在新的流表安装完毕之前，不删除原有路径的流表，这样能够避免路径调整时的数据丢失，确保了数据传输的连续性。

如图 3.8 所示，假设某条流原来的传输路径为 s1-s2，在 T0 时 s1-s2 链路发生拥塞，触发动态路由机制为流重新选择传输路径为 s1-s3-s2，若按照“正向顺序”写入流表，T1 时刻 s1 上的流表已经重新安装成功，新进入到网络中的流依据 s1 中新的流表进行匹配转发，即从 s1 的 port2 转发出去，根据流表项的动作，数据流已经从 s1 到 s3 了，但此时 s3 的流表项可能还没重写完成，就会造成数据流“不知路在何方”的情况出现。根据 OpenFlow 协议的规定，此时对数据包的处理有两种方式，一是将数据包直接丢弃，二是借助 packet-in 消息将数据包信息送到控制器，由控制器决定处理该数据包的方式，前者会导致大量数据包的丢失，后者会致使短时间内大量 packet-in 消息送到控制器，影响控制器的运行。因此无论哪种方式，都会影响到数据的传输。

若采用“反向顺序”下发流表，即按照 s2-s3-s1 的顺序下发新的流表，在最后一个交换机 s1 写入新流表之前，数据流仍在原来的路径进行传输，避免了“不知路在何方”的情况出现。

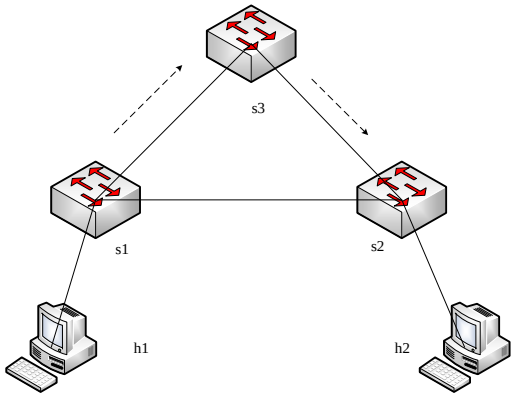


图 3.8 流表安装图

采用主动删除旧流表的方式解决新老流表共存的问题，选择此方式而没有使用流表项超时失效的方式是因为旧流表已经没有任何用处，若旧流表自然失效需要的时间较长，旧流表将会长时间占据着流表的资源，这是一种浪费。

3.3.2 队列调度策略

OpenvSwitch 虚拟交换机是以 FIFO（First In First Out，先进先出）的转发规则将到达出端口的数据包转发出去的。FIFO 是一种“尽力而为”的转发规则，谁先到达，谁先享受更多的资源。当某个优先级低的业务流的带宽需求较大且先到达转发端口，会导致后到达出端口但优先级高的业务能用的带宽较少，有可能造成高优先级业务在短时间内无法得

到转发的情况出现。因此，为保障流媒体视频流良好的传输，QoS 控制策略在 OpenvSwitch 虚拟交换机的出端口设计采用新的队列调度算法，保证流媒体视频流在每个转发节点上的 QoS 要求。根据 OpenFlow 协议提供的队列配置功能，即在流表中增加将数据流送入到端口队列的动作，数据流进入到交换机匹配到流表后，就会根据流表的指引进入出端口的某个队列中，之后根据队列规则的配置进行数据的转发。

1）队列调度分析

队列调度的规则有很多，如 FIFO、PRIO、HTB 等。其中，FIFO 只能提供尽力而为的服务，不能对不同业务进行区分处理，无法提供 QoS 区分服务；PRIO 虽然可以对报文进行分类，使得高优先级业务报文得到优先调度，但容易使低优先级队列中的业务被“饿死”；HTB 队列规则既能够实现不同优先级业务的区分调度，也能实现流量整形和速率限制的功能。

本文队列调度的需求是为视频流媒体提供足够的带宽；带宽充足时，不同优先级的业务流间可以相互借带宽；端口提供不同业务流的优先级转发。根据需求分析以及综合对比多个队列调度规则的优缺点后，利用分层令牌桶（Hierarchical Token Bucket，HTB）实现队列调度的功能。

树状结构的 HTB 队列规定能实现对业务数据进行不同层次分类的功能[42]。其利用 DWRR 算法能按照业务优先级进行差额轮询调度的特点，将优先级不同的业务数据调度到相应的队列中；利用令牌桶算法实现对流量的限速和整形的功能。两种算法的简要介绍如以下内容。

a、差额加权轮询算法

差额加权轮询算法是在加权轮询算法的基础上进行的优化调整，而加权轮询算法则是对轮询算法的完善[43]。下文对三种算法的原理分别给予简单的介绍。

轮循算法（Round Robin，RR）是相对简单的调度算法，其调度的原理是按照次序依次调度每个队列，当队列里没有数据包时则执行跳过的动作。尽管顺序的调度能在一定程度上显示算法的公平性，但顺次调度无法保证不同优先级业务的需求，无法提供区分业务优先级的调度服务，导致不能在多种业务流并存的状态下满足较高优先级业务的 QoS。

加权轮循算法（Weighted Round Robin，WRR）对 RR 算法的改进在于使用加权轮询报文的方式实现队列调度的功能 [44]。其实现的原理是为每个队列设置一个权值和计数器，调度伊始，队列中计数器的值都会被初始化为权值，之后每调度一个报文，该队列的计数器值会自动减 1，若队列中报文个数或者计数器值为 0 时，将计数器的值重置为权值，然后轮询到下一个有效队列中继续执行调度操作。WRR 算法规定没有报文的队列不进行调度，这样能将占有的带宽分给其他队列使用，WRR 算法能实现在多个队列间灵活分配带宽的功能，为不同优先级业务提供不同的服务，图 3.9 为 WRR 算法原理图。

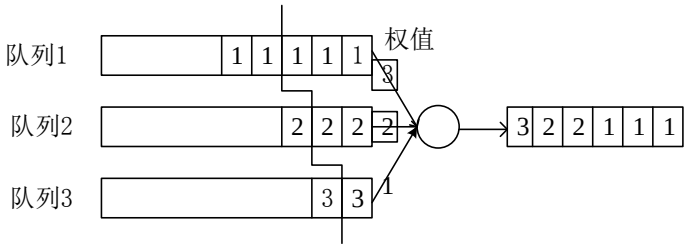


图 3.9 WRR 算法原理图

基于数据包调度的 WRR 算法在一定程度上保证了不同业务的服务需求，但数据包较多的队列会得到更多的调度机会，对于其他队列来说是不公平的。为了解决 WRR 算法存在的这一问题，算法研究者提出了差额加权轮询（Deficit Weighted Round Robin，DWRR）算法。DWRR 算法将 WRR 算法进行了优化，将队列权值的报文个数改为报文的字节数，每个队列都有一个权值 Q 和差额 D，根据 Q 和 D 的相关计算关系完成调度功能，优化了的 WRR 算法可以较为公平地为不同优先级业务提供 QoS 服务。

b、令牌桶算法

令牌桶算法的功能是对网络中的流量进行整形和限制流量的速率，可以对发送到网络中的数据数量进行限制并可允许突发数据的产生，图 3.10 为令牌桶算法的流程图。

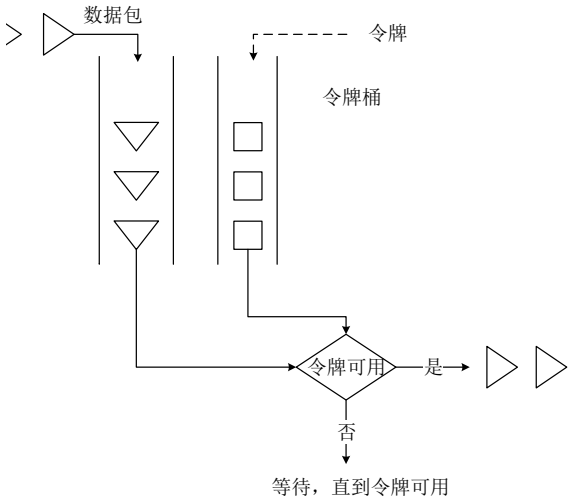


图 3.10 令牌桶算法流程图

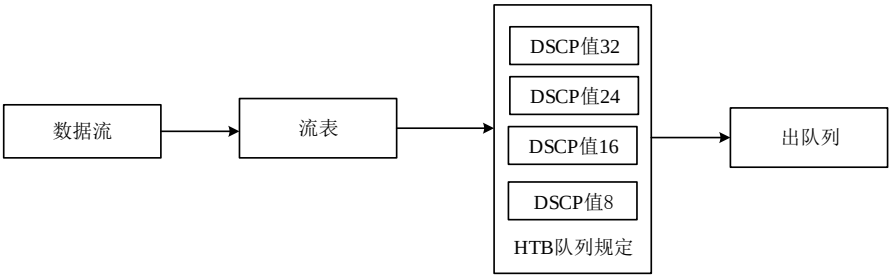
令牌桶内以固定的速率产生令牌，当数据包通过令牌桶时，若桶中的令牌数量充足，则数据包可以被发送出去，同时桶中的令牌数被减去相应值；若桶中的令牌数不足，则数据包只能等到令牌桶中的令牌充足时，才能被发送出去。

令牌桶对突发数据的传输和传输速率的限制是通过令牌产生的速度和数据包的传输速率进行比较来实现的。如果令牌产生的速率一直大于数据包的传输速率，就会使得令牌不断增多直至达到桶的容量，然后就会开始溢出，之后若有突发的数据流经过令牌桶，桶中积攒的令牌即可处理这些突发的流量。如果令牌产生的速率一直小于数据包的传送速

率，可能会出现令牌桶内的令牌消耗殆尽的情况，这将会限制数据包的发送速率，使其与令牌桶的产生速率相同，进而达到限制流量的目的。

2）队列调度策略设计

图 3.11 展示了队列调度策略的设计，数据流进入交换机后，经过流表的匹配后执行流表项中的转发到端口、数据包标记以及入队列的动作，不同优先级业务报文被标记并进入到不同的队列中。通过 OpenvSwitch 上设置的 HTB 队列规定的调度后，视频流不仅可以获得更多的输出带宽，也可以被优先调度出端口，转发到下一个交换机上；带宽充足时的借带宽机制尽力的保障了其他业务流的传输。



3.11 队列调度策略图

3.4 基于遗传算法的多 QoS 约束路由算法分析与设计

根据本文的应用场景，使用遗传算法解决多种约束的 QoS 路由问题，下文将对遗传算法进行设计来满足实际应用的需要。

3.4.1 路由度量数学模型

QoS 控制指标参数分为可加性指标、乘性指标和凹性指标[45]，分别表示一条路径的 QoS 为该路径每条链路 QoS 要求值的和、乘和最值。例如跳数和时延是可加性指标，丢包率是乘性指标，带宽为凹性指标中的最小值。

网络中选择路径进行传输的问题实际上是图论中的寻路问题，一般把网络拓扑结构映射为有向图$G(V,E)$，其中，V 是网络中所有点的集合，在本文的研究中指交换机；E 是网络中所有边的集合，在本文的研究中指交换机间的链路。每条边$(a,b)\in E$。

设有一条路径 $p,s\in V$ 为路径的起点，$d\in V-S$ 为路径的终点，根据以上定义的 QoS 控制指标参数的分类情况，可以得到路径 p 的传输性能参数表示为：

$$delay(p(s,d))=\sum_{(a,b)\in p(s,d)} delay(a,b) \tag{3.11}$$

$$loss(p(s,d))=1-\prod_{(a,b)\in p(s,d)}(1-loss(a,b)) \tag{3.12}$$

$$bandwidth(p(s,d))=\min_{(a,b)\in p(s,d)}\{bandwidth(a,b)\} \tag{3.13}$$

公式(3.11)为该路径端到端的时延，公式(3.12)为该路径总的丢包率，公式(3.13)为该路径中的剩余带宽。

对于视频流媒体的传输，需要找到这样一条路 $p(s,d)$，满足以下的约束条件：

$$delay(p(s,d)) < D \tag{3.14}$$

$$loss(p(s,d)) < L \tag{3.15}$$

$$bandwidth(p(s,d)) < B \tag{3.16}$$

其中 D、L、B 分别表示传输对时延、丢包率、带宽的要求，D 的值为 10ms，L 的值为20×10^{-3}， B 的值为 0.8Mbps。

3.4.2 遗传算法基本概念

遗传算法是模拟生物自然选择的过程，其能较好地解决许多求解最优解的问题，算法中有很多基本概念，下面对这些基本概念进行简要的介绍。

种群是算法操作的对象，是解决问题的子解集；染色体即种群中的个体，每个染色体都有成为解的可能；基因是染色体序列上的位点，QoS 路由中指每个交换机；染色体编码是将具体问题映射到计算语言的操作，编码方式有很多种；适应度函数是评价染色体优劣的函数，进化算子根据染色体的适应度值完成进化操作；遗传算子是对染色体进行操作的进化算子，其推动遗传算法继续执行下去。

3.4.3 染色体编码设计

染色体编码将实际问题映射到计算机语言，利用计算机高效的计算方式解决实际中遇到的问题，其是遗传算法发挥效用的前提。染色体编码的方式影响着算法的计算效率，且不同的应用环境中，选择的编码方式也不同，常用的编码方式有符号编码、二进制编码和基于树的表示。

由于网络拓扑中源节点到目的节点间的路径有多条，且每条路径经过的网络设备数量并不一定相同，即网络中路径的长度不一定相同，其是可变的，因此本文 QoS 路由算法的染色体编码选择变长符号编码的方式。由于每个交换机都有属于自己唯一的 datapath.id 标识，将一条路径上的每个交换机的 datapath.id 按照途经的顺序排列即完成了一条染色体的编码，染色体的头基因是路径的源端，尾基因是路径的目的端，将基因进行顺次排列就是一条路径从源端到目的端经过的网络设备的顺序。以图 3.12 网络组成结构为例，使用箭头标出的路径用变长符号编码的方式表示为[1，2，4，7](一条路径是一个染色体)。

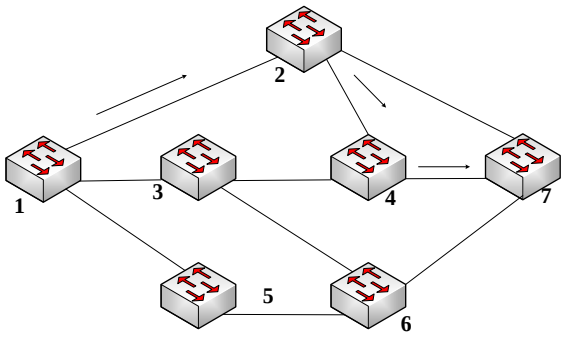


图 3.12 一种示例网络拓扑图

3.4.4 种群初始化设计

应用遗传算法解决实际问题需要初始化产生一些染色体，这些染色体是算法操作的对象。种群初始化一般需要考虑两点，分别是初始化采用的方式和种群的规模[46-47]。

初始化方式指采取何种方式生成初始种群，包括随机式和启发式，随机初始化以随机的形式生成初始种群，启发式初始化利用已有的启发式方法生成初始种群。后者需要牺牲掉种群的多样性，为增大 QoS 路由算法的搜索空间，采用随机初始化的方式对种群进行初始化。

种群规模指一代种群中个体的数量，种群规模小使算法的计算量小，进而缩短计算的时间，但同时也可能掉进局部最优的“圈套”，使计算出来的解并非全局的最优解，影响问题的解决；种群数量过多扩大了算法的搜索范围，可以找到全局最好的解，但过大的种群规模也意味着在进行进化等操作时要耗费大量的计算时间，进而降低了算法的时效性。综上分析，种群规模应该设置成这样一个值，既不会使计算出来的解是局部最优值，也能提高算法的效率，针对本文的拓扑设计情况，将拓扑中所有路径的 30%作为初始种群。

3.4.5 适应度函数设计

适应度用来判别一条染色体的好坏，其值根据适应度函数计算。遗传算法中的各种进化操作都需要以染色体的适应度值作为衡量标准，因此适应度函数的设计成为遗传算法中的关键一步，其对算法收敛的速度以及能否找到合适的解有较大的影响。

由于带宽为凹性参数，不适于作为适应度函数，因此适应度函数的设计只考虑时延和丢包率。染色体 i 的适应度计算函数如公式(3.17)所示，其中 2 为该适应度函数中包含的 QoS 性能个数。

$$f(i)=\left(f_{delay}(i)+f_{loss}(i)\right)\div 2 \tag{3.17}$$

$f_{delay}(i)$ 是染色体 i 所表示的路径的时延适应度计算函数，如公式(3.18)，其中 $sum(delay)$ 是染色体 i 所代表的路径的时延总和，n 是一代种群中个体的数量。

$$f_{delay}(i)=sum_i(delay)\div\sum_{j=1}^{n}sum_i(delay) \tag{3.18}$$

$f_{loss}(i)$是染色体i所表示的路径的丢包率适应度计算函数，如公式(3.20)，其中由于丢包率并不是加性参数，需要通过公式将其转换成加性参数，如公式(3.19)。

$$loss'(a,b)=\log(1+loss(a,b)) \tag{3.19}$$

$$f_{loss}(i)=sum_i(loss')\div\sum_{j=1}^{n}sum_i(loss') \tag{3.20}$$

时延与丢包率的适应度计算函数都除以了该代所有染色体相应的 QoS 参数和，这是因为从丢包率与时延的测量结果可知，丢包率值与时延值相差很多，若不做这样的处理，就不能均衡时延和丢包率，选择的路径可能更多的考虑了时延，而忽视了丢包率，做了相应的处理后取值在同一范围内，被考虑的可能就相对均等。

根据上述定义的公式可知，对于一条染色体的适应度来说，其值越小，表明该条染色体代表的路径更优秀。

3.4.6 遗传算子设计

1）选择算子设计

选择算子与自然选择中的适者生存类似，适应能力强的个体留下来，较弱的个体被淘汰，选择算子根据个体的适应度判断一个个体是否拥有较强的“生命力”。选择算子执行优秀个体保留并遗传到下一代或利用交叉算子生成子代个体的操作。

为了减小当前群体中的最优个体在下一代丢失的概率，提高遗传算法的收敛速度以及收敛到最优解时的稳定性，本文的遗传算子采用“精英选择(elitist selection or elitism)”策略，该策略将每代种群中的优秀个体直接遗传到下一代，并不经过交叉等算子的操作。算法的步骤如下：

（1）确定每次保留的精英染色体数量和随机保留的普通染色体数量。

（2）对于一代群体，按照适应度值对染色体进行由小到大的顺序排列，根据设定值从中选择出适应度较好的的精英染色体，直接复制到下一代。

（3）根据设定值再从剩余的染色体中随机选择普通染色体，复制到下一代。

（4) 对剩余的染色体执行交叉操作，并传入下一代。

2）交叉算子设计

交叉算子是对染色体进行基因交换的操作（本文指的是交换机节点），交叉操作将会得到新的基因组合即新的个体，扩展了算法的搜索空间。交叉算子的设计一般包括交叉率的设置和交叉方式的设计，但本文的设计是在经过选择算子后，对剩下的所有染色中除头尾基因外有相同基因的染色体执行交叉操作，因此只设计交叉方式而不设置交叉率。

交叉算子中的交叉方式需要根据具体的应用场景进行选择，并根据情况进行相应的

改进，交叉方式包括单点交叉、多点交叉和融合交叉等方式，各有各的优缺点。比较常用的是单点交叉，其交叉思想是以随机选择的方式在染色体中选择一个基因点作为交叉点，然后根据交叉点将两个染色体的基因对调，形成两个与原有染色体不同的染色体序列，如图 3.13 所示。

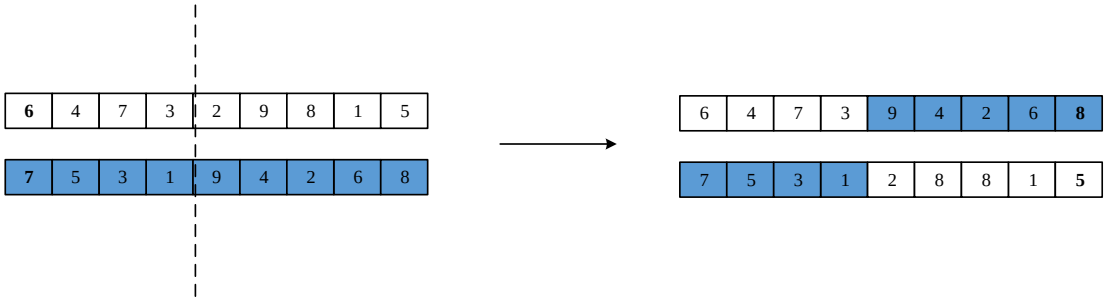


图 3.13 单点交叉示意图

QoS 路由算法中的交叉算子的设计借鉴了单点交叉的思想，根据交叉点将两条路径中的节点进行对调，形成新的路径。但该交叉方式与一般的单点交叉又有些差别，其需要父母双亲的首尾基因相同，即源和目的节点要相同，且除首尾基因相同外，还要有其他基因相同才能进行交叉操作。这些相同的基因片段都可作为交叉点，交叉算子以随机选择的方式在这些潜在的交叉点中选择一个作为交叉点，以该交叉点为中心轴，将两条染色体的前后基因分别进行互换，便得到两条新的染色体序列，如图 3.14 所示。

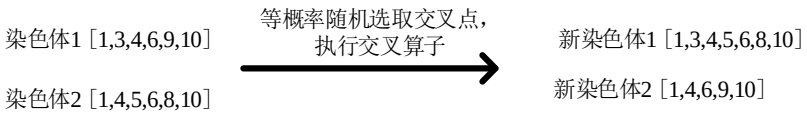


图 3.14 QoS 路由单点交叉示意图

3）变异算子设计

变异算子能扩大基因的多样性，增大算法的搜索范围，在一定程度上防止局部最优解的出现，变异概率和变异方式是实现变异算子的基础。

变异概率指染色体发生变异的几率，这个值并没有统一的规定，但在不同的应用场景中，不同变异率带来的结果不尽相同。变异率的值设计的高一些，能增加种群的多样性，避免计算出来的解是某个范围内的最优解而不是全局的最佳解。但变异率设置的过高，会导致变异的个体增多，上一代种群中的一些优秀基因可能会丢失，进而导致失去最优解。综上分析可知，变异率应该设置成一个合适的值，既不会导致计算出来的解是局部最优解，也能较好地保留上一代种群的遗传信息。通过改变变异率的值进行实验对比可知，在本文设计的拓扑路径中，变异率在 1%能提供一个很好的结果，故将变异概率设为 1%。

常见的变异算子有翻转、随机重置和交换突变。本文的变异算子是随机重置，随机重

置就是随机地选择一条染色体中的某个基因，将其值改为一个合理的随机值。

4）终止检查

终止检查决定运行的算法在何时终止，遗传算法通常设置迭代次数作为算法的终止条件，但本文的 QoS 控制需要对算法收敛的速度给予提升，所以设定了另一个终止条件，即得到满足需求的染色体（路径）的数量达到设置的N值时，也可以将遗传算法终止，并将适应度最好的一条路径作为解。N 的值可根据具体的场景进行灵活的设置，以满足不同场景的需求，包括最初 QoS 请求需要的更好服务（较大 N 值）和动态路由需要的迅速收敛（较小 N 值）。

利用遗传算法求解 QoS 路由的流程如图 3.15 所示，其中 M 为初始种群数，L 为下一代种群数，N，K 为判断算法是否停止的终止条件。

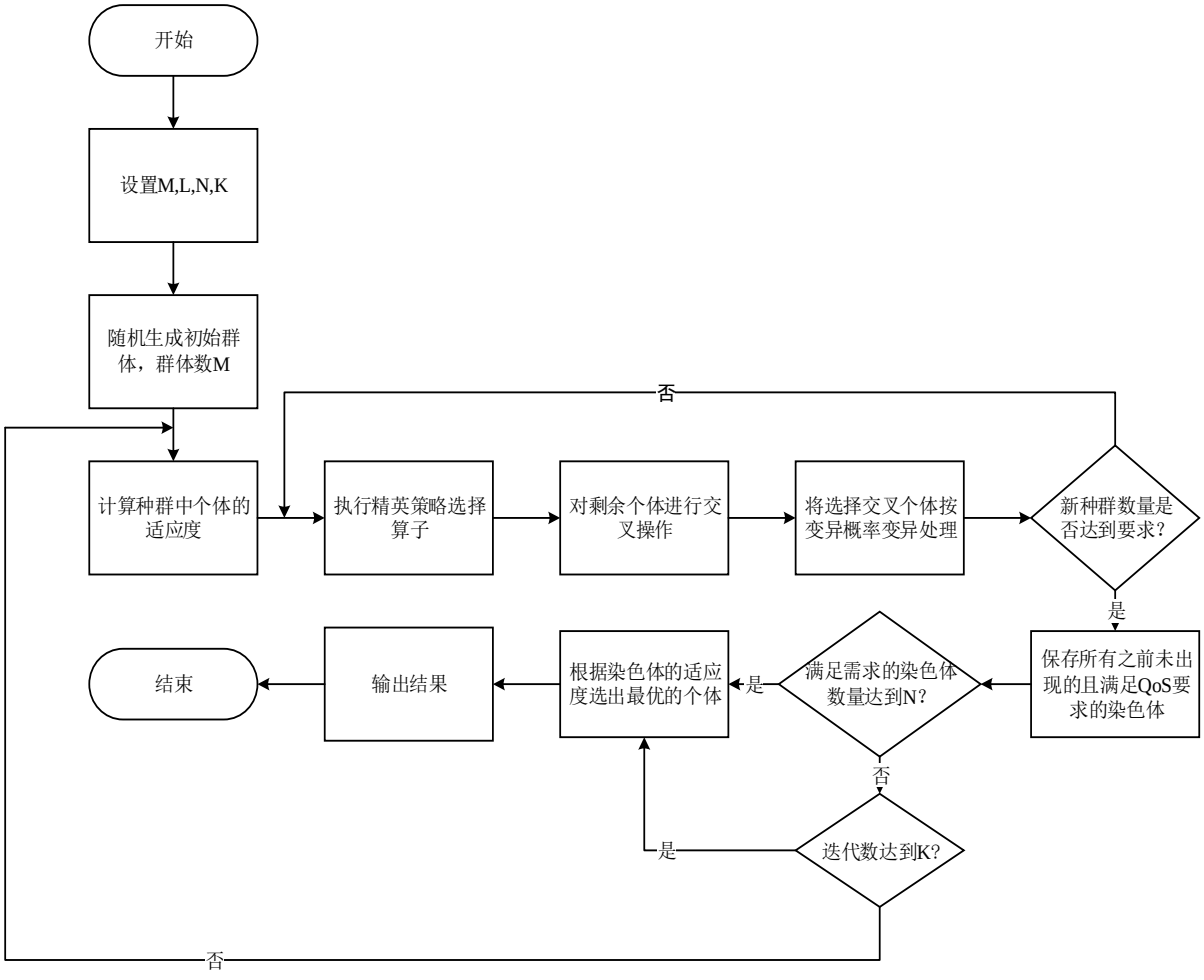


图 3.15 QoS 路由流程图

3.5 本章小结

本章首先分析了 SDN 网络中 QoS 控制策略的需求，并对整体 QoS 控制策略框架的设计进行了阐述，然后对框架中各个模块的具体设计做了详细的介绍，最后对基于遗传算

法的多 QoS 约束路由算法的设计进行了详细的阐述。

4 QoS 控制策略实现

4.1 QoS 控制框架实现概述

基于 SDN 网络的视频流媒体 QoS 控制策略框架的实现包括两方面：QoS 路由和队列调度，本章将阐述这两部分功能的具体实现。

如图 4.1 所示，在 Ryu 控制器中增添了 Topology Manager 模块、Measure 模块、Routing Management 模块，Ryu 中的这三个模块分别对应于 QoS 控制策略框架中的拓扑管理模块、链路性能测量模块、路由管理模块。

Measure 模块获取链路 QoS 性能参数的具体数值。Topology Manager 模块获取网络拓扑视图，并整合链路性能信息和拓扑链路的连接信息，形成全局的网络资源图。Routing Management 实现了区分不同数据流的功能；并进行相应的路径计算；将数据流的标记、入队以及计算出的路径以流表的形式下发到交换机中，供数据流匹配、传输；在网络传输路径出现拥塞时，能够及时进行动态路由，保证视频流媒体的 QoS。

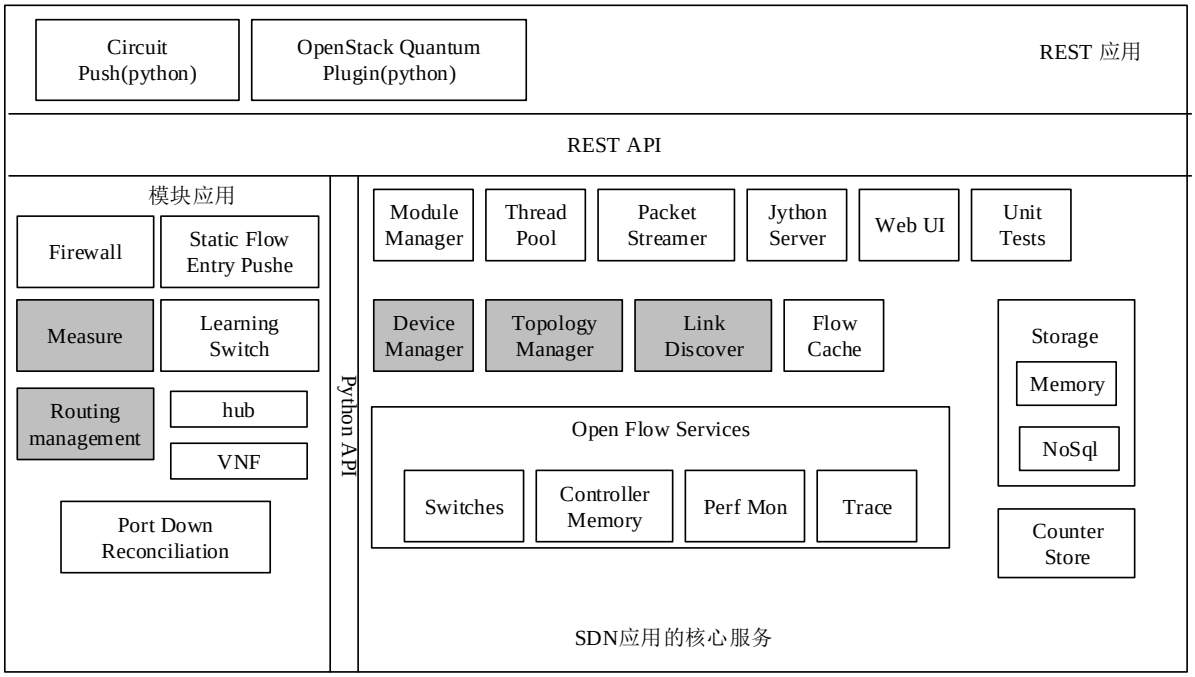


图 4.1 QoS 控制在 Ryu 中的模块组成

本文利用控制器和 OpenvSwtich 实现了 QoS 控制策略中的队列调度模块，主要负责在数据转发层面上为业务提供 QoS 保障。下面将阐述 QoS 控制策略框架中各个模块的具体实现过程。

4.2 QoS 路由的实现

4.2.1 链路性能测量模块的实现

由第三章可知，在 SDN 网络中，链路性能参数的测量利用了 OpenFlow 交换机内部统计信息的计数器，根据记录的每个 Port、Flow 等信息即可以完成链路性能信息的测量，下面对链路性能测量模块的实现进行详细的介绍。

1）链路性能测量模块框架

链路性能测量模块获取网络中每条链路性能参数的数据值，图 4.2 是链路性能测量模块的框架图，下面对模块中各个类、接口及类和接口之间的关系进行简单的介绍。

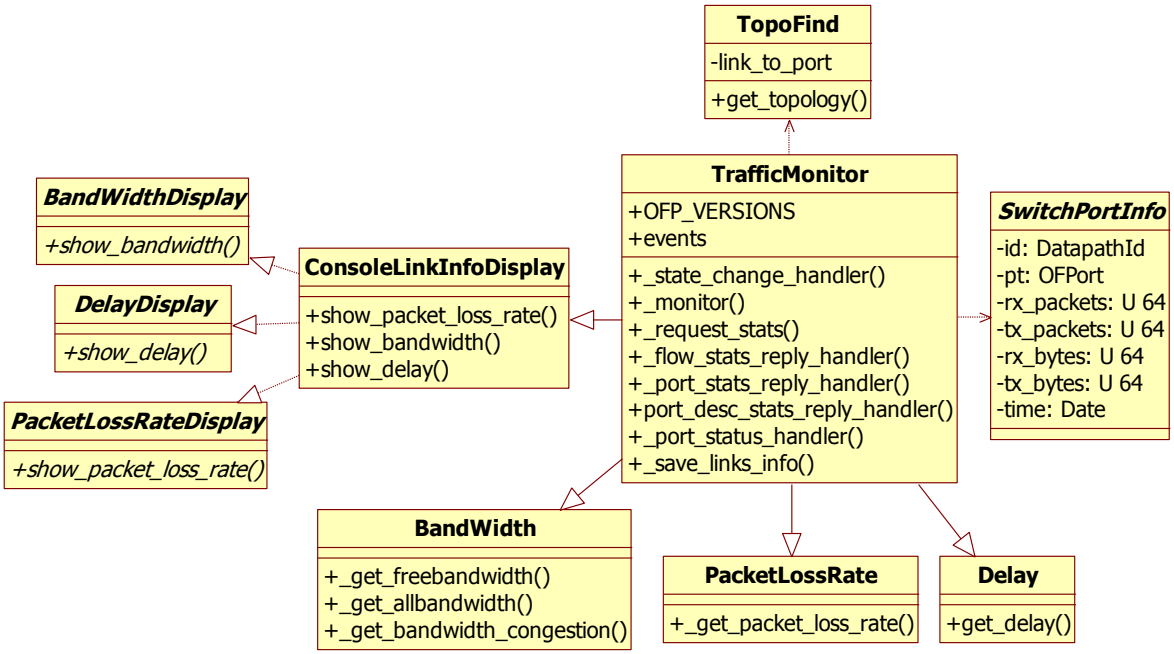


图 4.2 链路性能测量模块类和接口图

TrafficMonitor 类为链路性能测量模块的主体，类中的_monitor()方法能够以固定时间间隔向底层的所有交换机发送 Port_Stats_Request 和 Flow_Stats_Request 请求消息。类中的_flow_stats_reply_handler()方法和_port_desc_stats_ reply_handler()方法的作用是在交换机收到请求报文后，将需要的流信息和端口固定带宽信息上报给控制器。

BandWidth 类定义了_get_allbandwidth()、_get_freebandwidth()、_get_bandwidth_congestion ()三个方法，其中_get_allbandwidth()是获取拓扑结构中每条链路固有带宽的方法，其余两个方法用来计算拓扑结构中每条链路的剩余带宽和拥塞率。

PackLossRate 类定义了_get_packet_loss_rate()方法用以计算网络拓扑链路的丢包率。

Delay 类中的_get_delay()方法能够计算得到网络拓扑中所有链路的时延值。

BandWidthDisplay、PacketLossRateDisplay、DelayDisplay 三个接口分别定义了show_bandwidth()、show_packet_loss_rate()、show_delay()三个方法用来展示网络中每条链路的带宽、丢包率以及时延的信息。ConsoleLinkInfoDisplay 类实现了以上三个接口中的方法，用以在控制台展示相关的信息。

TopoFind 类为整个链路性能测量模块提供依赖，提供整个网络的拓扑连接情况。

SwitchPortInfo 类保存控制器通过端口、流等统计消息获得的网络中的基本数据，为链路带宽、链路拥塞率、链路丢包率计算提供数据依赖。

2）带宽、拥塞率及丢包率测量实现

带宽、拥塞率以及丢包率的测量为 QoS 路由提供了数据的支撑，下面分别阐述它们的实现。

Traffic 类中的_monitor()方法是一个周期执行的线程，该线程以 2s 为一个周期向所有交换机发送获取端口信息、流信息以及固定带宽值的请求报文，交换机根据收到的报文内容，将端口的流量统计数据、流的统计数据以及端口固定带宽信息发送给 Ryu 控制器，Ryu 对收到的反馈报文进行解析计算，即可得到所需的数据值。根据端口流量统计数据得到的交换机端口发送与接收的字节数以及两次统计的时间间隔能够计算出端口的速率值，见公式(3.3)；根据 OFP_Port_Stats 消息中的rx_packets字段、OFP_Flow_Stats 消息中的instructions字段和packets_count字段能够得到端口接收与发送的 packets，见公式(3.7)和(3.8)。

OpenvSwitch 端口的带宽为 20Gbps，需要利用 Mininet 对交换机端口的实际带宽进行限定，以符合真实的情况。当底层网络连接上 Ryu 控制器后，Ryu 中的链路性能测量模块便会发送 Port_Desc_Stats_Request 统计报文获取这个带宽数值，根据反馈的报文将带宽信息进行存储。

得到端口固定带宽和端口速率后，链路性能测量模块利用公式(3.5)和(3.6)计算端口的剩余带宽和端口的拥塞率，之后根据一条链路端口间的对应关系得到链路的剩余带宽和拥塞率。在得到送往当前交换机的某个端口的数据包总数以及对端端口收到的报文总数后，链路性能测量模块会利用公式(3.9)计算链路的丢包率。图 4.3 为获取交换机链路剩余带宽、拥塞率以及丢包率的流程图。图 4.4、图 4.5、图 4.6 分别为链路剩余带宽、拥塞率以及丢包率的终端展示，可以更直观地看到网络中的这些信息。

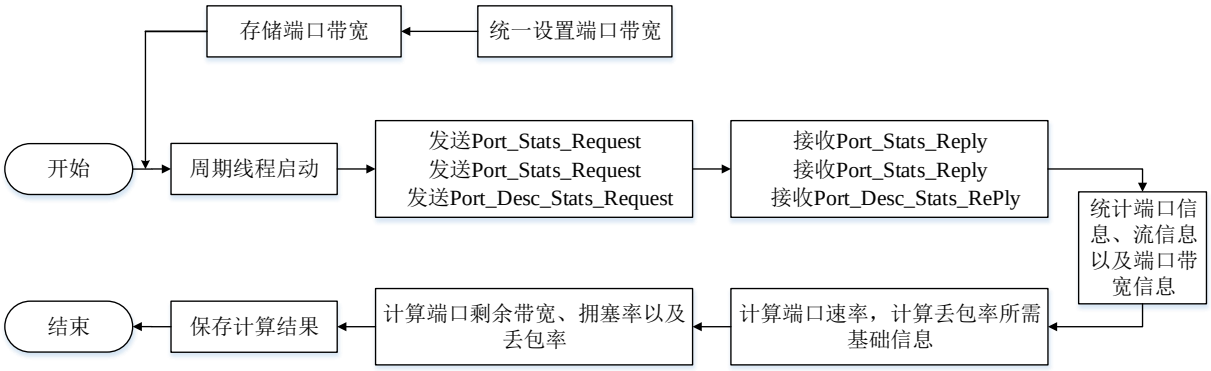


图 4.3 获取链路性能信息的流程图

----------bandwidth ----------

src_dpid	src_port_no	src_bandwidth	src_capacity	dst_dpid	dst_port_no	dst_bandwidth	dst_capacity	free_bandwidth
0000000000000001	3	0.000879853357774	2.3	0000000000000005	1	0.000882205513784	2.3	2.2991177945
0000000000000005	4	0.000561403508772	1.4	0000000000000003	5	0.000561309722686	1.4	1.3994385965
0000000000000002	1	1.00086631016	1.1	0000000000000001	1	0.99995334111	1.1	0.0991336898
0000000000000006	2	0.000561966884094	2	0000000000000008	2	0.00056206088993	3.2	1.9994380331
0000000000000005	1	0.000882205513784	2.3	0000000000000001	3	0.000879853357774	2.3	2.2991177945
0000000000000007	2	0.901556485356	3	0000000000000008	3	0.901104048177	1.6	0.6988959518
0000000000000001	2	0.000879853357774	2.5	0000000000000003	3	0.00088205813565	2.7	2.4991201466
0000000000000003	3	0.00088205813565	2.7	0000000000000001	2	0.000879853357774	2.5	2.4991201466
0000000000000008	1	0.000481766477083	1.3	0000000000000004	2	0.000481122619445	3.2	1.2995182335
0000000000000002	2	0.000561497326203	1.3	0000000000000003	1	0.000561309722686	2.4	1.2994385027
0000000000000005	3	0.000481203007519	1.6	0000000000000007	3	0.000482008368201	1.6	1.5995179916
0000000000000004	1	0.000882205513784	2.3	0000000000000003	2	0.00088205813565	2.5	2.2991177945
0000000000000001	1	0.99995334111	1.1	0000000000000002	1	1.00086631016	1.1	0.0991336898
0000000000000006	4	0.000481685900652	3	0000000000000004	3	0.000481122619445	2	1.9995188774
0000000000000003	2	0.00088205813565	2.5	0000000000000004	1	0.000882205513784	2.3	2.2991177945
0000000000000007	1	1.00044317992	1.5	0000000000000001	4	0.99995334111	1.5	0.4995568201
0000000000000001	4	0.99995334111	1.5	0000000000000007	1	1.00044317992	1.5	0.4995568201
0000000000000008	3	0.901104048177	1.6	0000000000000007	2	0.901556485356	3	0.6988959518
0000000000000003	4	0.000561309722686	1.8	0000000000000006	3	0.000561966884094	2.1	1.7994386903
0000000000000006	1	0.000882943143813	1.8	0000000000000005	2	0.000882205513784	2.1	1.7991170569
0000000000000007	4	0.000482008368201	1.5	0000000000000006	5	0.000481685900652	3	1.4995179916
0000000000000005	2	0.000882205513784	2.1	0000000000000006	1	0.000882943143813	1.8	1.7991170569

图 4.4 剩余带宽展示图

----------bandwidth ----------

src_dpid	src_port_no	src_bandwidth	src_capacity	dst_dpid	dst_port_no	dst_bandwidth	dst_capacity	bandwidth_con
0000000000000007	3	0.000470819028936	1.6	0000000000000005	3	0.000471281296024	1.6	0.000295
0000000000000001	3	0.000943488943489	2.3	0000000000000005	1	0.000942562592047	2.3	0.000410
0000000000000003	2	0.000943952802236	2.5	0000000000000004	1	0.000942100098135	2.3	0.000410
0000000000000002	1	0.772637168142	1.1	0000000000000001	1	0.772257493858	1.1	0.702397
0000000000000006	2	0.000471976401 18	2	0000000000000008	2	0.000471744471744	3.2	0.000236
0000000000000005	1	0.000942562592047	2.3	0000000000000001	3	0.000943488943489	2.3	0.000410
0000000000000007	2	0.771521099117	3	0000000000000008	3	0.770020638821	1.6	0.481263
0000000000000001	2	0.000707616707617	2.5	0000000000000003	3	0.000707964601 77	2.7	0.000283
0000000000000003	3	0.000707964601 77	2.7	0000000000000001	2	0.000707616707617	2.5	0.000283
0000000000000002	2	0.000471976401 18	1.3	0000000000000003	1	0.000235988200 59	2.4	0.000363
0000000000000004	1	0.000942100098135	2.3	0000000000000003	2	0.000943952802 36	2.5	0.000410
0000000000000001	1	0.772257493858	1.1	0000000000000002	1	0.772637168142	1.1	0.702397
0000000000000005	4	0.000471281296024	1.4	0000000000000003	5	0.000471976401 18	1.4	0.000337
0000000000000007	1	0.771521099117	1.5	0000000000000001	4	0.772257493858	1.5	0.514838
0000000000000001	4	0.772257493858	1.5	0000000000000007	1	0.771521099117	1.5	0.514838
0000000000000004	2	0.000706575073602	3.2	0000000000000008	1	0.000707616707617	1.3	0.000544
0000000000000006	5	0.000707964601 77	3	0000000000000007	4	0.000706575073602	1.5	0.000471
0000000000000008	3	0.770020638821	1.6	0000000000000007	2	0.771521099117	3	0.481263
0000000000000003	4	0.000707964601 77	1.8	0000000000000006	3	0.000707964601 77	2.1	0.000393
0000000000000007	4	0.000706575073602	1.5	0000000000000006	5	0.000707964601 77	3	0.000471
0000000000000004	3	0.000706575073602	2	0000000000000006	4	0.000707964601 77	3	0.000353
0000000000000005	2	0.000471281296024	2.1	0000000000000006	1	0.000471976401 18	1.8	0.000262

图 4.5 拥塞率展示图

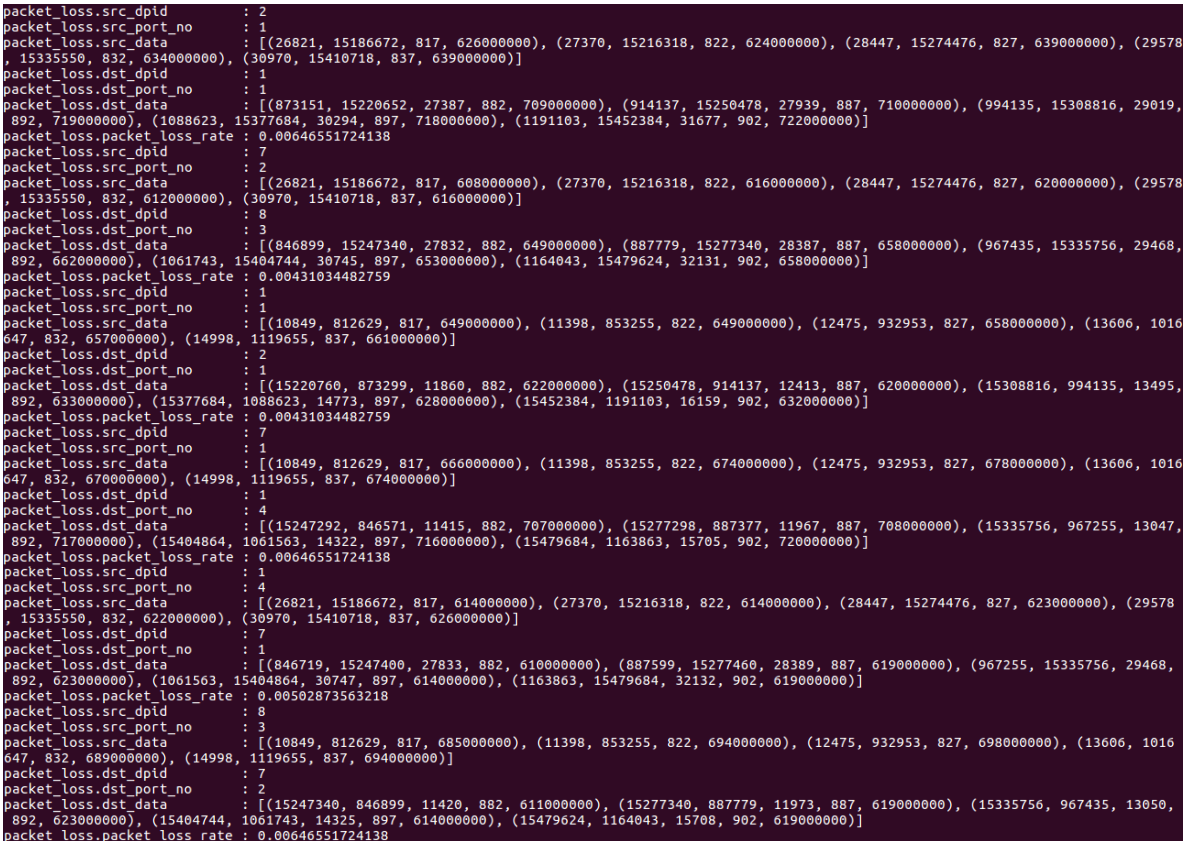


图4.6 丢包率展示图

3）时延测量实现

通过发送 LLDP 数据包和 Echo 数据包并携带时间戳得到相关的时间信息，根据时延计算公式计算出链路的时延。图 4.7 是获取链路时延的相关类的关系图，图 4.8 是链路时延信息的展示图，下面阐述每个类的功能。

OFEchoHandler 类通过发送 Echo 消息实现交换机和控制器之间的信息交流，类中的 send_echo_message()方法发送 EchoRequest 消息到网络中所有的交换机上，并将信息发出的时间T_{echo_start}放入请求报文中，交换机收到 Ryu 控制器发来的 EchoRequest 报文后，向 Ryu 回复带有T_{echo_start}信息的 EchoReply 报文，Ryu 使用 dispose_echo_reply()方法对 Echo Reply 报文进行处理分析，得到 EchoRequest 消息发出的时间T_{echo_start}，Ryu 收到 EchoReply 报文的时间T_{echo_end}与T_{echo_start}的差值即是控制器到交换机再回到控制器的大致时间。

LinkDiscovery 类实现对 OpenFlow 交换机间链路连接情况的监控，类中的 send_LLDPmessage()方法向网络中所有交换机发送 LLDP 数据包，并将消息发出的时间戳T_{lldp_start}插入到消息报文中，每个交换机收到控制器发来的 LLDP 数据包后，根据流表的规则将 LLDP 数据包发送给自己的邻居交换机，邻居收到 LLDP 数据包后无法处理，把该数据包交给 Ryu；Ryu 使用 handle_LLDPmessage()方法对该 LLDP 数据包进行解析，得到插入的时间戳T_{lldp_start}的信息，Ryu 收到 LLDP 数据包的时间记为T_{lldp_end}；T_{lldp_end}与T_{lldp_start}的差值即为 LLDP 报文从控制器发出，经过交换机后，再回到控制器的时间。

LinkDelay 类中的 monitor_server_latency()方法周期地向网络中发送 LLDP 报文与 EchoRequest 报文，get_latency()方法根据 LinkDiscovery 类和 OFEchoHandler 类获得的时间数据通过公式(3.10)计算并存储时延信息。

TopoFind 为整个时延测量模块提供拓扑结构的依赖，构建网络拓扑视图。

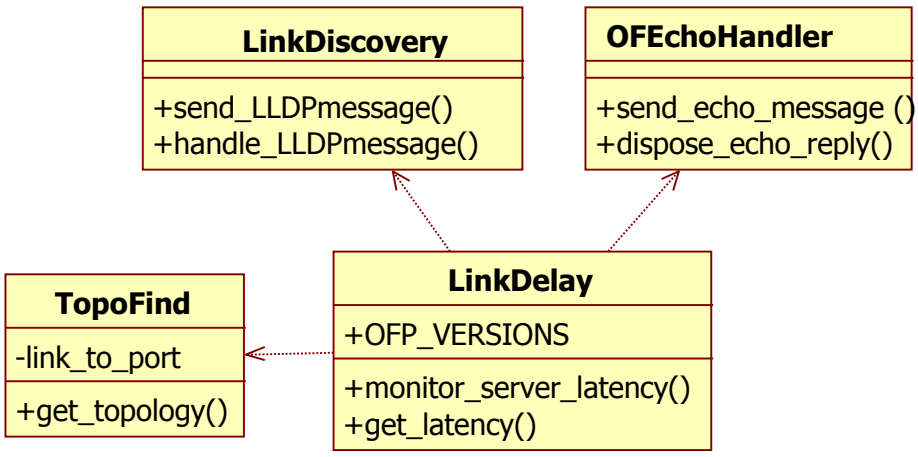


图 4.7 时延测量实现图

```
src   dst      delay
----------------------------
1<-->1 : 0
1<-->2 : 0.000402688980103
1<-->3 : 0.000939011573792
1<-->5 : 0.0027060508728
1<-->7 : 0.00105106830597
2<-->1 : 0.000402688980103
2<-->2 : 0
2<-->3 : 0.000771880149841
3<-->1 : 0.000939011573792
3<-->2 : 0.000771880149841
3<-->3 : 0
3<-->4 : 0.000718951225281
3<-->5 : 0.000301241874695
3<-->6 : 0.000947952270508
4<-->8 : 0.00253617763519
4<-->3 : 0.000718951225281
4<-->4 : 0
4<-->6 : 0.00109362602234
5<-->1 : 0.0027060508728
5<-->3 : 0.000301241874695
5<-->5 : 0
5<-->6 : 0.00105941295624
5<-->7 : 0.000855922698975
6<-->3 : 0.000947952270508
6<-->4 : 0.00109362602234
6<-->5 : 0.00105941295624
6<-->6 : 0
6<-->7 : 0.00135147571564
6<-->8 : 0.000896453857422
7<-->8 : 0.605813622475
7<-->1 : 0.00105106830597
7<-->5 : 0.000855922698975
7<-->6 : 0.00135147571564
7<-->7 : 0
8<-->8 : 0
8<-->4 : 0.00253617763519
8<-->6 : 0.000896453857422
8<-->7 : 0.605813622475
```

图 4.8 时延信息展示图

4.2.2 拓扑管理模块的实现

拓扑管理模块负责提供全局的拓扑结构，并为 QoS 路由算法提供带有最新链路性能参数的拓扑数据。该模块利用 LLDP 协议获取的链路连接关系信息和链路性能测量模块提供的链路 QoS 参数信息实现全局信息的获取。图 4.9 展示了拓扑管理模块相关类和接

口的关系，下面阐述每个类的功能。

TopoManager 类为拓扑管理的主体，类中的 discover()方法以线程的形式周期地发现网络资源。

TopoFind 类负责网络的监测并发现网络中的拓扑，它内部维护了全局网络的拓扑映射信息。

Graph 类是物理拓扑的抽象图，提供方法将拓扑信息转换为邻接矩阵和邻接链表以供路径计算模块使用。类中的 get_adjacency_matrix()方法将网络拓扑与链路性能信息结合，附带有网络实时链路性能参数的拓扑可以为 QoS 路由算法提供最新的资源，Dijkstra()方法将拓扑结构转换成邻接矩阵供 Dijkstra 算法使用。

Edge 类负责存储链各个性能的数据值，src、dst、bandwith、loss Rate、delay 和 congestion rate 分别表示一条链路的源交换机、目的交换机、剩余可用带宽、丢包率、时延和拥塞率。

ConsoleTopoDisplay 是 TopoDisplay 接口的一种实现，负责将获取到的拓扑信息展示于终端。

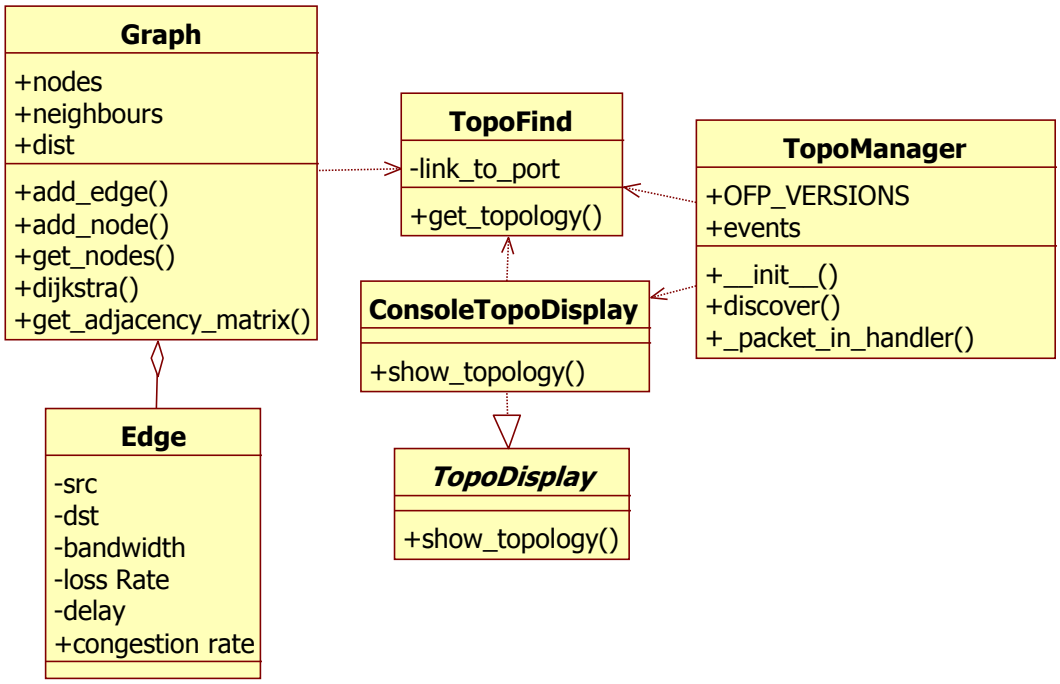


图 4.9 扑管理模块类与接口间的关系

4.2.3 路由管理模块

1）路由管理模块框架

本文在 Ryu 控制器上增添了 Routing Management 模块，对应于 QoS 路由控制框架的路由管理模块，它实现了数据流的区分以及基于 Dijkstra 算法和遗传算法的路由功能，用以计算相关应用数据流的路由，并将控制行为以流表的形式下发到交换机上，使交换机依

照流表执行命令。路由管理模块实现的框架图如图 4.10 所示，下文对模块框架中的类和接口进行介绍。

PacketInMessageHandler 类是对 PacketIn 报文的处理，包括丢弃、泛洪以及转发处理，转发处理（do_forwarding ()方法）即是对有路由请求的数据流进行路径的计算及转发。

RoutePlanner 接口定义了 calculate()方法用以路由的计算，QoSRoutePlanner 类和 DijkstraRoutePlanner 类实现了 RoutePlanner 接口，分别是视频流媒体基于遗传算法的 QoS 路由，其他优先级数据流基于 Dijkstra 算法的路由。

Graph 类为 QoSRoutePlanner 类和 DijkstraRoutePlanner 类提供带有链路性能数据的拓扑信息以及拓扑信息。

RouteHandler 类主要实现数据流的区分，将路由展示在终端以及下发流表到交换机上。

DynamicQoSRoute 类为网络拥塞时的动态路由模块，类中的 add_path()与 del_path()方法的作用分别为添加新路径和删除旧路径。

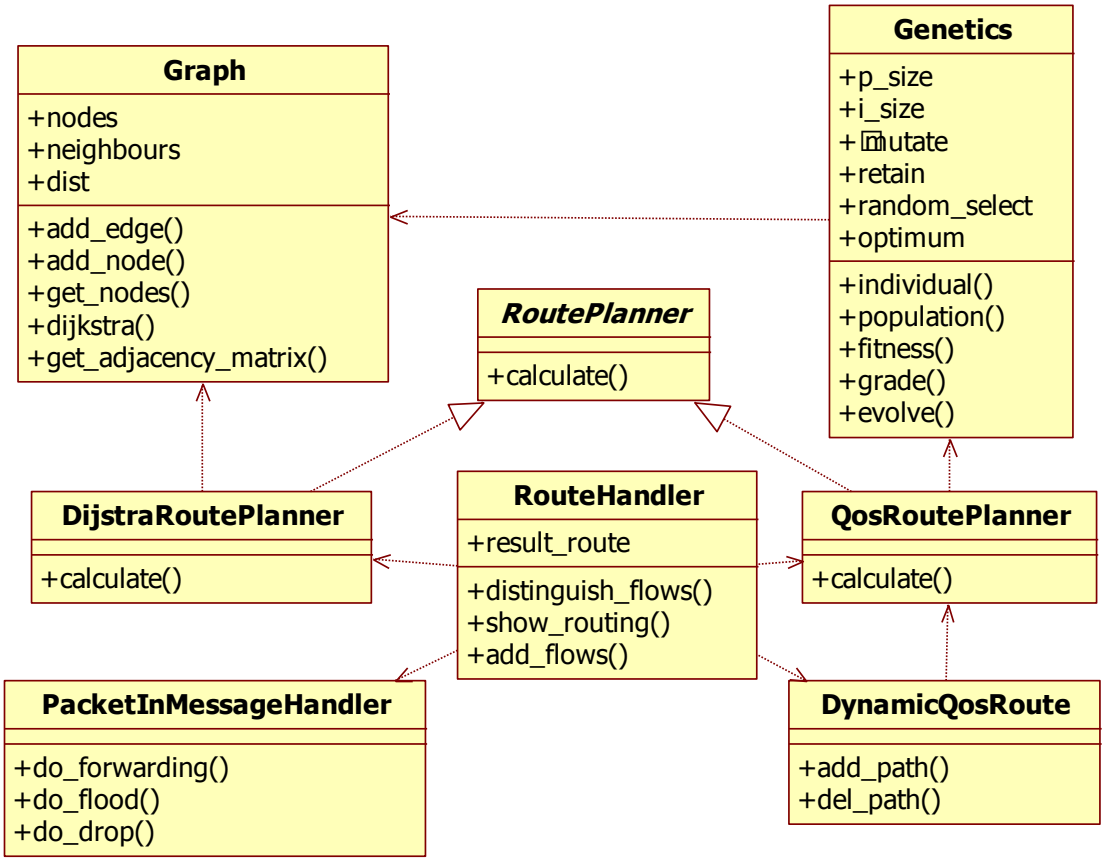


图 4.10 路由管理模块类与接口图

2）路由管理模块的实现

在 SDN 网络中，数据流的传输依靠流表中流表项的动作，若匹配到某个流表中的流表项，就按照该流表项规定的动作进行传输；若未能匹配到任何流表，一般情况下，将会触发 packet-in 消息将数据包的信息交给 Ryu，Ryu 的路由管理模块中的 PacketInMessageHandler 类会对该 packet-in 消息进行不同的处理。为了对视频流媒体进行 QoS 路由，本文扩展了 do Forwarding ()方法，使 Ryu 控制器在区分出视频流媒体数据流后能基于遗传算法进行 QoS 路由，其他优先级的业务数据流基于 Dijkstra 算法进行最小跳数路由。

图 4.11 为扩展后的 do Forwarding()方法的执行过程，在 PacketInMessageHandler 类收到 packet-in 消息并解析为路由请求后，由 do Forwarding ()方法对 packet-in 消息进行进一步的处理，do Forwarding ()方法先要提取消息报文中的报文头信息，随后对报文头中的数据进行解析得到源目的交换机的信息和应用层协议，若解析得到的是非最高优先级的业务流，Ryu 使用 Dijkstra 算法（跳数为代价）计算传输路径，然后将路径流表下发到对应的交换机上，以指引业务数据流的传输。若解析得到的是最高优先级的视频流媒体，则需要使用遗传算法获得源目的交换机间的传输路径。若传输路径出现拥塞，则会触发 DynamicQoSRoute 类调用 QoSRoutePlanner 重新规划路径，若新的路径与 RouteHandler 中的 result_route 不同，则直接增加新的路径，将旧的路径删掉，按照新路径进行流表的下发，若新的路径与 RouteHandler 中的 result_route 相同，则表示网络中已经出现了全面拥塞，这时候只能依靠队列调度策略在端口对视频流媒体进行优先调度，最大可能减少拥塞带来的问题。流表中的动作除了有转发到端口，还有数据包标记以及入队的动作，二者都是队列调度策略的基础。

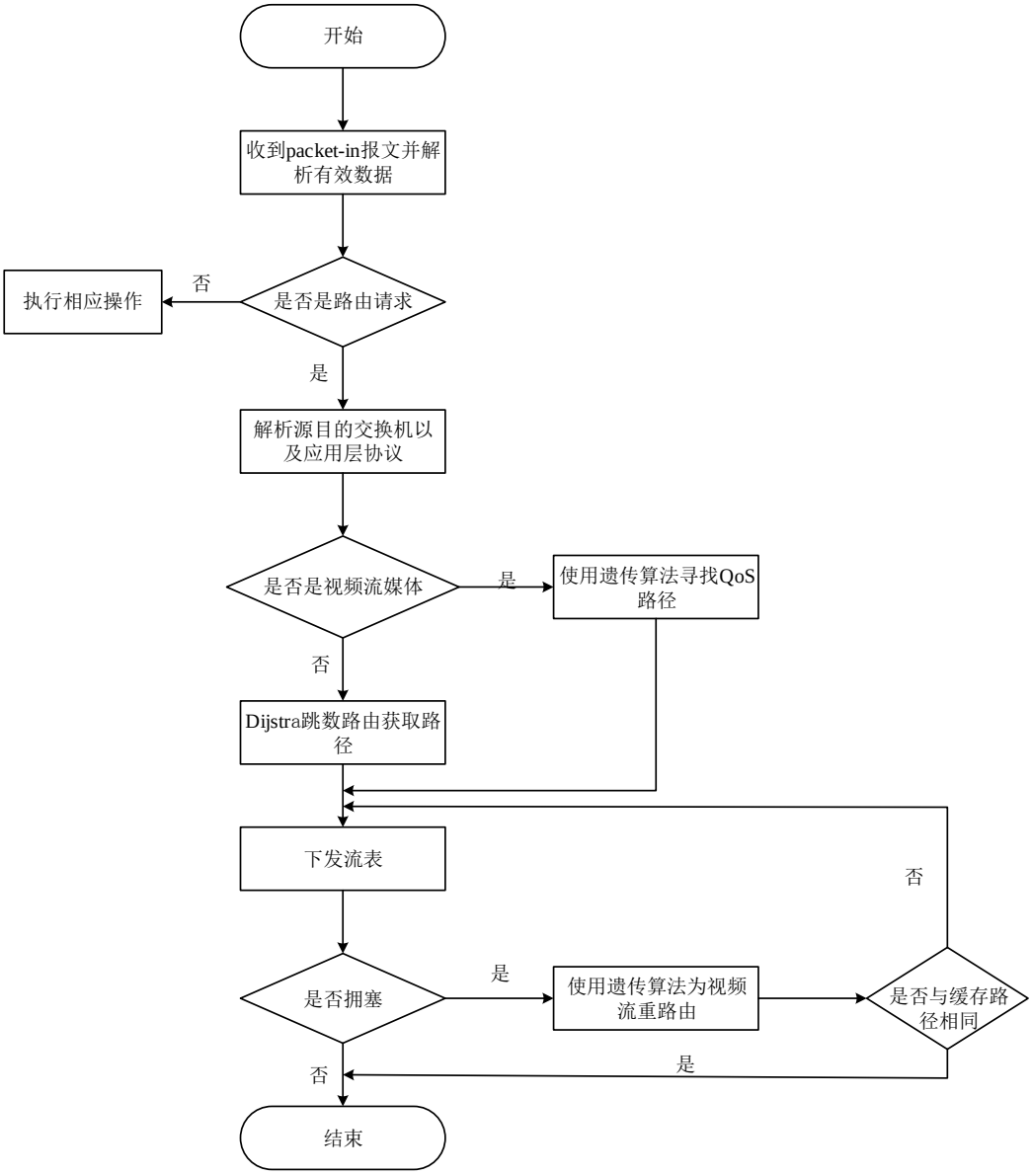


图 4.11 do Forwarding()方法的执行过程

4.2.4 遗传算法的实现

1）遗传算法模块框架

本节阐述基于遗传算法的 QoS 路由的实现，算法的实现基于第三章对算法的设计，图 4.12 是遗传算法实现的模块框架图，其中 Genetics 类实现了基于遗传算法的 QoS 路由，individual()是随机路径生成算法，能够随机生成不同的路径即种群中的个体；population()方法实现初始种群的生成，该方法生成一定数量的种群，是遗传算法实现的基础；fitness()是适应度函数的计算；evolve()是遗传算法的逻辑主体，内部实现了各种遗传算子。

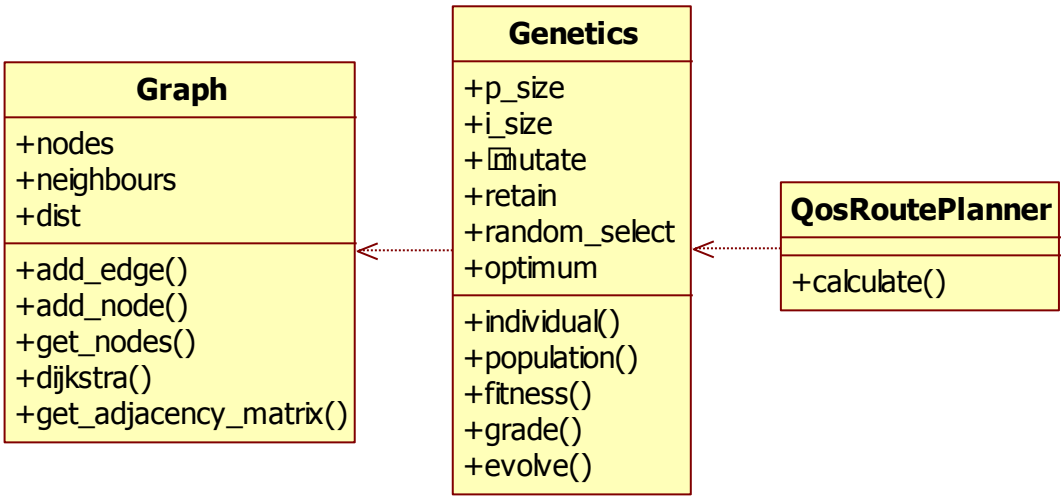


图 4.12 遗传算法模块框架图

2）种群初始化策略的实现

Genetics 类是种群初始化的核心类，类中的 individual()方法随机生成从起点到终点的路径作为初始种群，算法的流程如图 4.13 所示。在算法执行选择未被标记的邻接节点操作之前，需要对所有还未标记过的邻居节点进行随机排列，以便下一跳的选择是完全随机的，保证路径生成的随机性。当一个节点没有未被标记的邻接点时，首先需判断是否为源点，若不是源点，则继续寻找，直到找到有邻接节点未被完全标记的点时停止算法的搜索，并将其邻接节点添加至路径链表中，重复上述操作，直到得到连接源点和终点的一条路径。

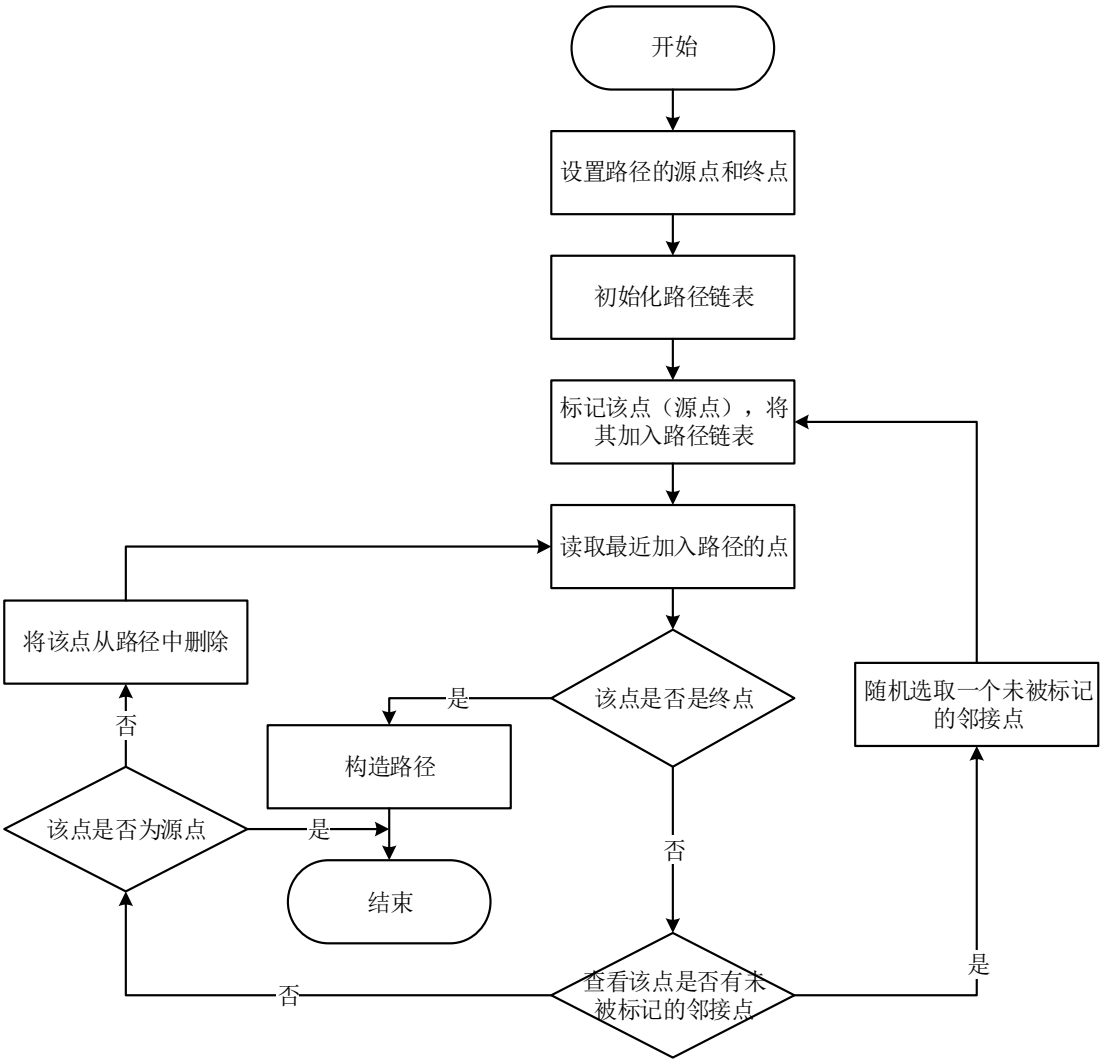


图 4.13 individual()方法流程图

Genetics 类的 population()方法通过 individul()生成随机路径，每次生成的路径会保存到路径集合中，如果新生成的路径在此集合中，新的路径将被丢弃，不用于构造种群；如果新生成的路径不在此集合中，就将新路径加入到路径集合中，直到达到种群的数量，这样的操作减少了个体的重复，能够有效地提高种群基因的多样性及搜索的广度。种群初始化实现的流程如图 4.14 所示。

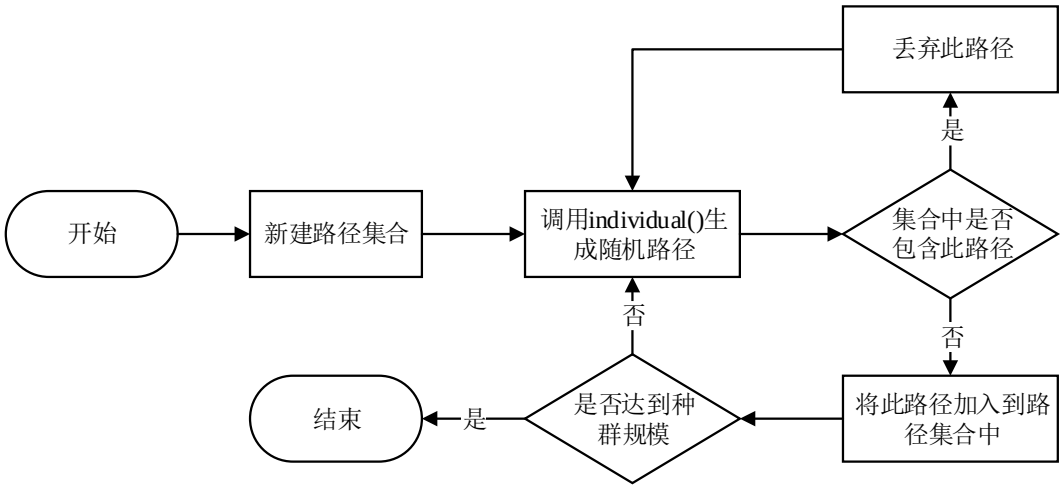


图 4.14 种群初始化流程图

3）适应度的计算

适应度的计算由 Genetics 类的 fitness()方法实现，适应度计算的流程如图 4.15 所示。在染色体适应度计算前需要设计好适应度函数，适应度函数在第三章已经进行了设计，除适应度函数外还需获取含有路径链路信息的链表以及每条链路的 QoS 性能参数（由拓扑管理模块提供）。最后根据适应度函数对每条染色体的适应度进行计算。在设计适应度函数时，对时延和丢包率的计算都除以了该代所有染色体相应性能值的和，所以在获得路径链表及 QoS 性能参数后，应计算出所有染色体相应性能值的和，以便每个性能的适应度的计算。

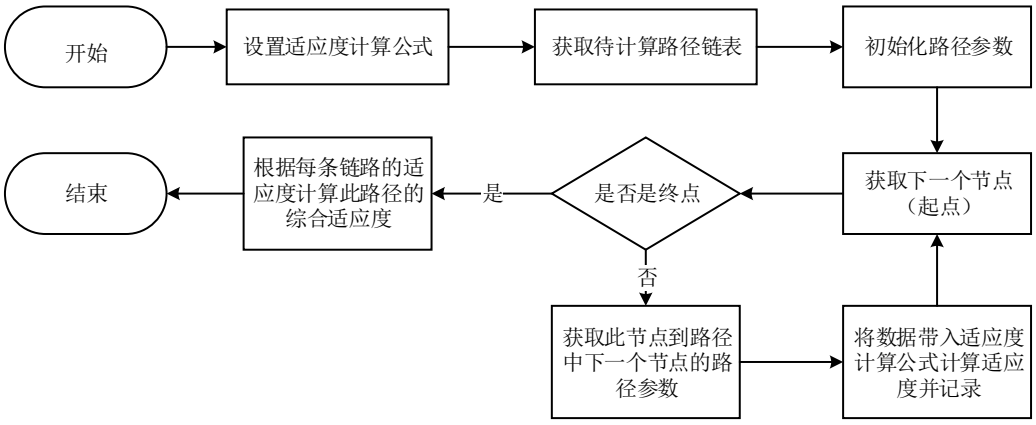


图 4.15 适应度值计算流程图

4）遗传算子的实现

本节对遗传算法中的选择、交叉等遗传算子的实现进行详细的阐述，在 Genetics 类中完成遗传算子的实现。

a、选择算子

精英选择策略实现的流程如图 4.16 所示，精英选择策略以当前的种群作为输入，对

输入种群中的个体进行适应度值的计算，得到个体的适应度值后根据适应度值进行排序，之后根据预先设定的保留率，选择群体中较好的个体，直接复制到下一代，保留率值设置为 10%；在剩下的个体中遍历，每遍历一个个体，使用随机生成器生成一个随机值，与选择率做比较，若选择率大于随机值，则将这个个体作为下一代种群中的个体，选择率设置为 1%，随机生成器生成的随机值在 0-1 之间。

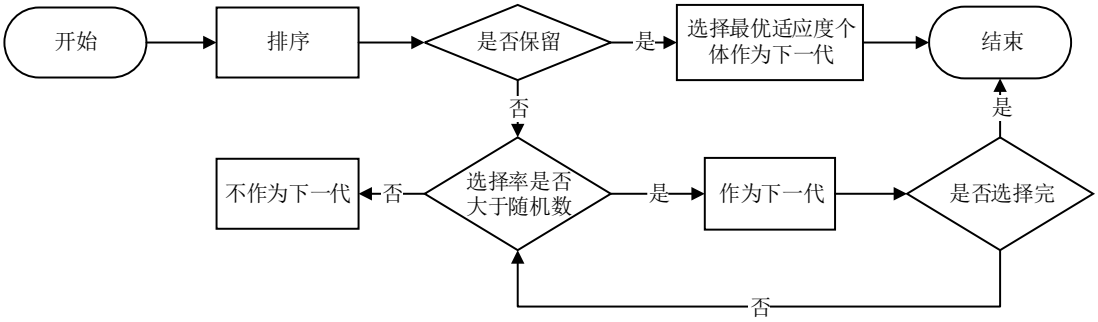


图 4.16 精英选择法流程图

b、交叉算子

交叉算子的实现流程如图 4.17 所示，对经过精英选择策略后剩下的所有个体进行交叉，交叉的规则为在剩余的染色体中随机选择两个染色体，若两个染色体除首尾基因外有超过一个的相同基因，则以随机选择交叉基因位点的方式执行交叉操作，由于交叉后形成的路径可能会存在环路的情况，因此需要对交叉后形成的新染色体进行基因是否重复的判断，以便排除有环路的染色体。

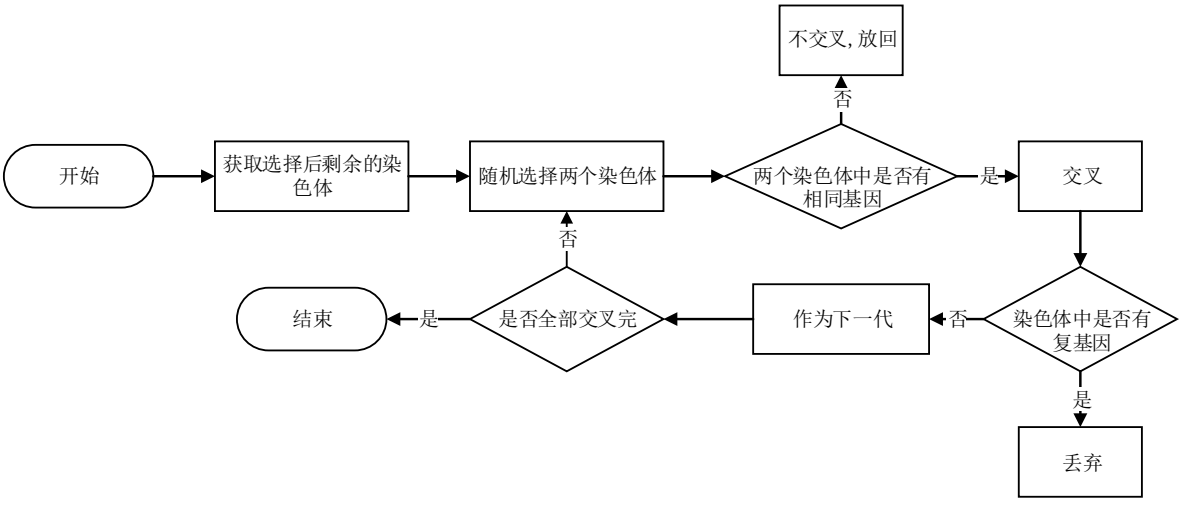


图 4.17 交叉算子流程图

c、变异算子

变异算子为随机重置的方式，其实现如图 4.18 所示。遍历所有经过选择与交叉后的下一代个体，每遍历一个个体，使用随机生成器生成一个随机数，与变异率做比较，若变

异率大于随机值，在该染色体中随机选择一个基因位点，将其随机重置为新的值。

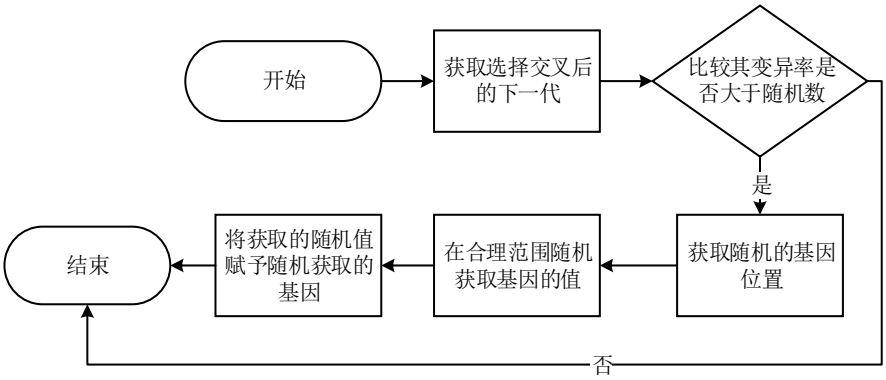


图 4.18 变异算子实现流程图

d、算法终止运行的实现

算法停止运行的实现如图 4.19 所示，考虑到算法应用的实际情况，设计了两种方式用以终止算法的运行，一是迭代数达到设定的值，算法将不再继续执行，返回最好的一条路径；二是需要依赖算法初始化时设置的 N 值，即存储的满足需求的染色体（路径）的数量到达N时，也可以终止遗传算法，并返回其中适应度最好的一条路径。这样做是因为本文的 QoS 控制对时间要求非常敏感，算法收敛速度的提升可以提高路由效率，提高 QoS 控制的性能。

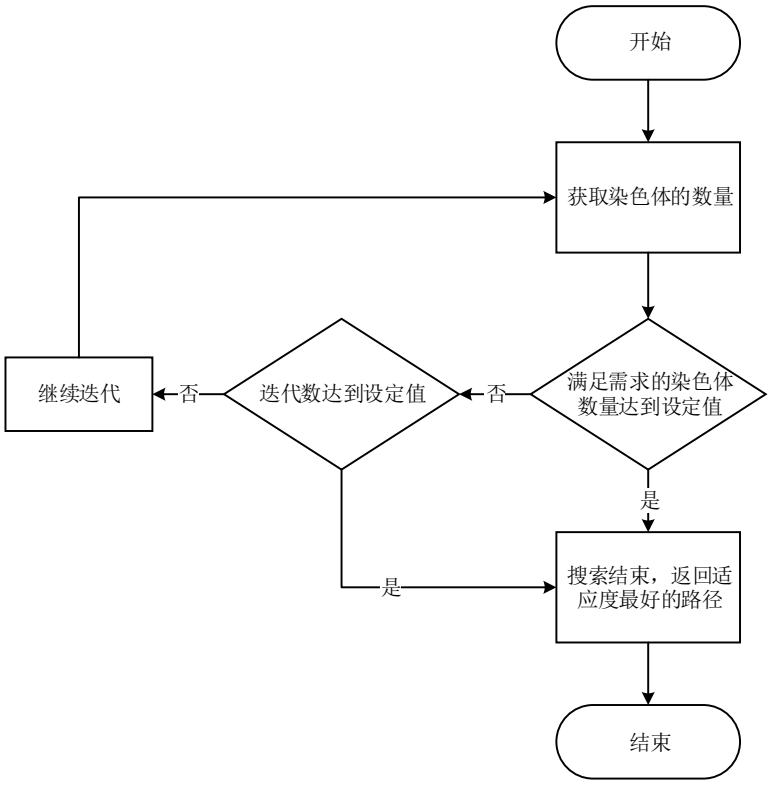


图 4.19 算法终止运行流程图

4.3 队列调度策略的实现

在线视频与监控视频等视频流媒体业务对 QoS 的需求最为严格，其他业务对 QoS 的需求和优先级有关，例如优先级最低的业务，其 QoS 需求也是最低的。依据表 3.1，本文将在线视频与监控视频、关键数据业务、事物处理与交互式数据、数据同步与 e-mail 的优先级分别映射为 32、24、16、8。根据第三章提出的队列调度策略，本节对队列调度功能予以实现。

队列调度策略中的流分类、标记、入队操作由控制器下发流表来完成，队列调度规则的配置在 OpenvSwitch 上完成，具体流程如图 4.20 所示。

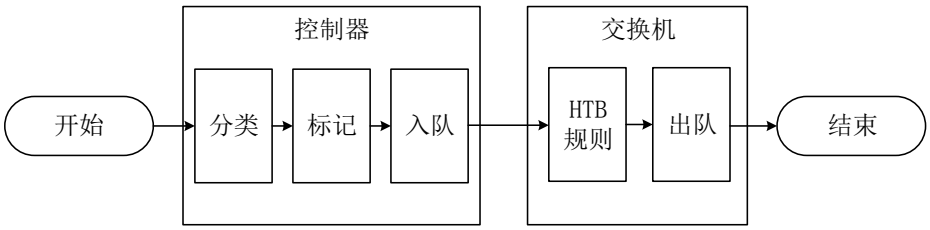


图 4.20 队列调度策略实现流程图

当数据到达边缘交换机时，控制器根据四层协议将数据流进行分类，根据分类的结果把数据流 IP 头中的 DSCP 值标记为对应的优先级值，如表 3.1。对 DSCP 值的修改是利用了 OpenFlow 指令集中的修改域，修改域可选的部分字段如图 4.21 所示。

Argument	Value	Description
in_port	Integer 32bit	Switch input port
in_phy_port	Integer 32bit	Switch physical input port
metadata	Integer 64bit	Metadata passed between tables
eth_dst	MAC address	Ethernet destination address
eth_src	MAC address	Ethernet source address
eth_type	Integer 16bit	Ethernet frame type
vlan_vid	Integer 16bit	VLAN id
vlan_pcp	Integer 8bit	VLAN priority
ip_dscp	Integer 8bit	IP DSCP (6 bits in ToS field)
ip_ecn	Integer 8bit	IP ECN (2 bits in ToS field)
ip_proto	Integer 8bit	IP protocol
ipv4_src	IPv4 address	IPv4 source address
ipv4_dst	IPv4 address	IPv4 destination address
tcp_src	Integer 16bit	TCP source port
tcp_dst	Integer 16bit	TCP destination port
udp_src	Integer 16bit	UDP source port
udp_dst	Integer 16bit	UDP destination port

图 4.21 修改域可选字段图

其中 ip_dscp 为 IP 报文头中 ToS 的前六位，通过修改 ip_dscp 的值，实现对数据包的标记，具体操作如图 4.22 所示，parser.OFPActionSetField 是执行 DSCP 值修改的动作，parser.OFPActionOutput 是执行数据包发送到端口的动作，此动作是所有动作的前提，若没有该动作，数据包将会被丢弃。入队操作是指控制器根据 OpenvSwicth 交换机中队列的配置，通过流表的形式将不同优先级的数据包放入到不同的队列中，具体命令为 parser.OFPActionSetQueue(queue_id)，queue_id 为交换机端口上已经配置好的队列号。

```
actions=[parser.OFPActionOutput(out_port)]
actions.insert(0,parser.OFPActionSetField(parser.OFPMatchField
.make(ofproto.OXM_OF_IP_DSCP,DSCP值)))
```

图 4.22 修改 DSCP 值的程序图

HTB 队列调度规则的实现是通过对 OpenvSwitch 软交换机上的 linux-htb 队列模块进行配置来完成的,具体配置如图 4.23 所示。

```
sudo ovs-vsctl set port s3-eth3 qos=@caiqos
-- --id=@caiqos create qos type=linux-htb
      other-config:max-rate=1000000000 queue=0=@q0,1=@q1,2=@q2,3=@q3
-- --id=@q0 create queue dscp=32
      other-config:max-rate=400000000,burst=40000000,priority=0,quantum=40000000
-- --id=@q1 create queue dscp=24
      other-config:max-rate=300000000,burst=30000000,priority=1,quantum=30000000
-- --id=@q2 create queue dscp=16
      other-config:max-rate=200000000,burst=20000000,priority=2,quantum=20000000
-- --id=@q3 create queue dscp=8
      other-config:max-rate=100000000,burst=10000000,priority=3,quantum=10000000
```

图 4.23 HTB 队列规则配置图

其中，type 用于配置 OpenvSwitch 端口队列的类型；dscp 用于配置进入队列的业务流的类别，具体见表 3.1；max-rate 指令牌桶的产生速率，表示当前队列每秒传输数据的最大字节数；burst 表示桶的大小，指能处理的突峰发送速率；priority 用于配置 DWRR 算法的优先级，用来保证重要的数据被优先发送，其值越小，该队列中的数据越优先被发送出去；quantum 用于配置 DWRR 算法调度的字节数。图 4.24、图 4.25 与图 4.26 分别是端口列表查询、qos 列表查询和 queue 列表查询的结果图。从图 4.24 中可以看到配置了端口 qos 策略和未配置端口 qos 策略的 qos 值的不同；从图 4.25 和 4.26 中可以看出 qos 和 queue 的配置结果。

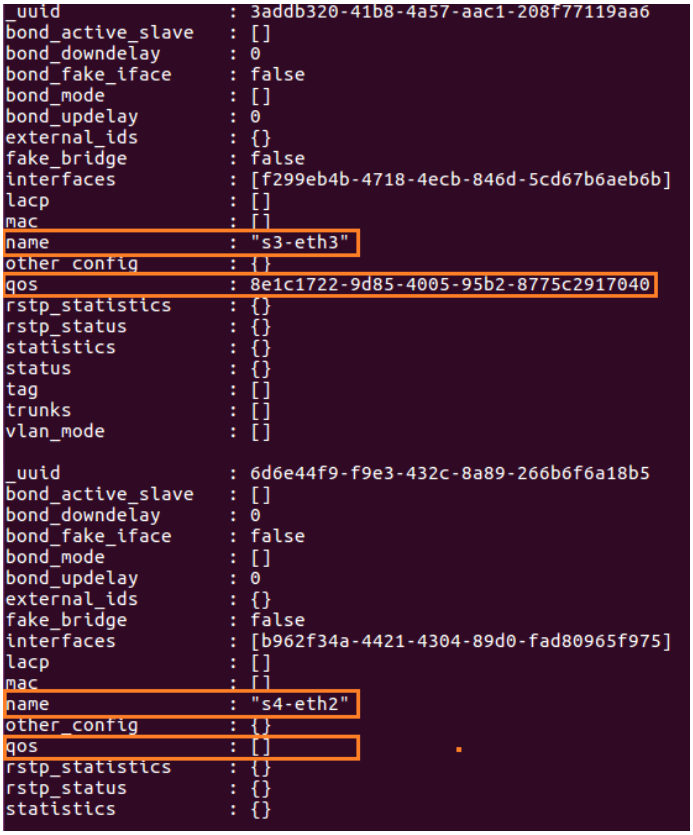


图 4.24 端口列表查询图

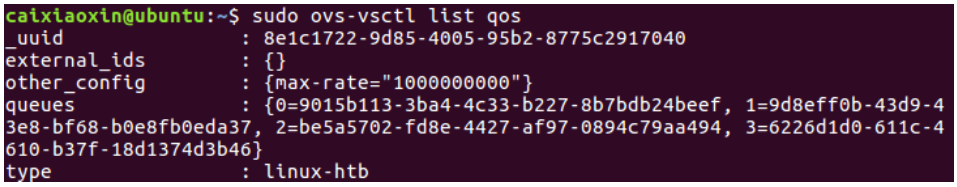


图 4.25 端口 qos 查询图

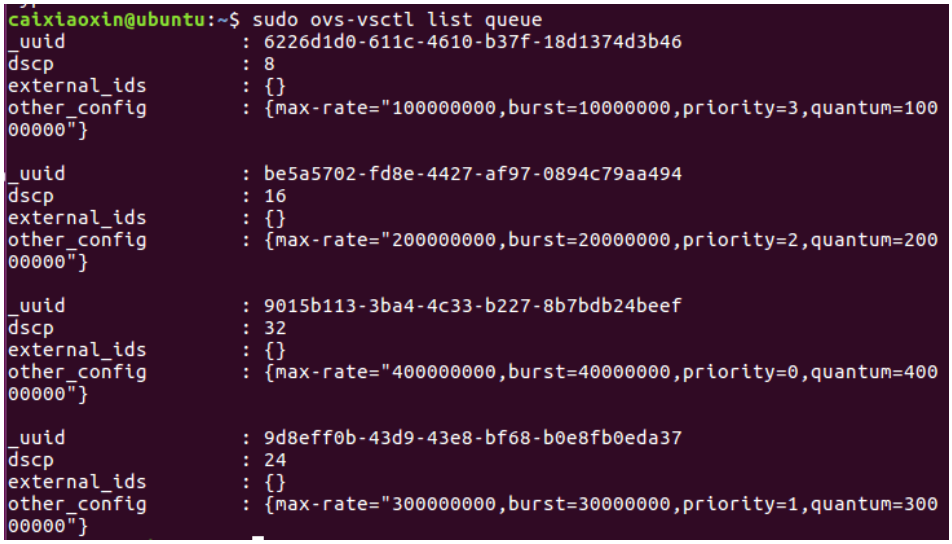


图 4.26 端口 queue 查询图

4.4 本章小结

本章叙述了视频流媒体 QoS 控制策略框架中的各个模块的实现，对 QoS 路由中的拓扑管理、链路性能测量等模块的具体实现过程作出了全面的阐述，并在 OpenvSwitch 的端口上完成了 HTB 队列规则的配置，以实现转发端口对 QoS 的保障。

5 实验仿真与结果分析

5.1 仿真实验环境介绍

5.1.1 网络仿真软件 Mininet

与 NS2/3、OPNET 等网络仿真器不同，Mininet[48-49]在单一的系统上就可模拟一个完整的网络功能，即便在一台普通的电脑上，也可以轻易的创建出与真实网络环境一致的网络组成结构，模拟的网络拓扑最多可有 4096 台主机的网络结构。Mininet 支持 OpenFlow 协议的各种版本以及 OpenvSwitch，可以很方便地进行 SDN 应用的测试和验证。Mininet 支持系统级别的回归测试，其不仅提供可由命令行直接创建的常见网络拓扑结构，还支持用户自己创建所需的任何拓扑结构。除此之外，在 Mininet 上编写相应程序实现的功能可以直接迁移到真实的硬件环境中，这对于开发与 SDN 有关的应用是非常方便的，这是因为搭建 SDN 硬件网络环境的难度较大且需要消耗大量的时间，Mininet 的这一特性在一定程度上缩小了 SDN 应用开发测试以及验证的周期，灵活性较大。

Mininet 是一个非常强大的网络仿真工具，其借助应用程序编程接口(Application Programming Interface)可以很方便地创建复杂的拓扑网络。表 5.1 描述了 Mininet 中创建网络拓扑常用的函数及变量，通过这些函数可以简单、快速地创建所需的网络。Mininet 也支持 Linux 命令行接口(CLI)的调用，在 CLI 上输入 nodes、net 命令可打印出网络中所有的节点、节点的连接信息，利用 link 可以模拟链路的通断情况。

表 5.1 Mininet 创建网络常用的函数

函数	功能
addHost()	为网络添加主机
addSwitch()	为网络添加交换机
addLink()	为网络添加链路
start()	开启网络运行
pingall()	测试网络的连通性
stop()	停止网络运行

5.1.2 其他实验环境及相关软件介绍

操作系统：Windows7 专业版 64 位，内存（RAM）：8.00GB

虚拟机和虚拟机系统：VMware Workstation10.0.1build-1379776，Ubuntu14.04

VLC media player：2.2.2

数据包抓包软件 Wireshark

OpenvSwitch：2.5.0

Mininet：2.3.0

Ryu：4.25

摄像头：Z-star Gsou USB 2.0

5.2 网络环境搭建

视频传输仿真平台主要由 Ryu 控制器、流媒体服务器、摄像头和 Mininet 软件模拟的网络拓扑组成，如图 5.1 所示。

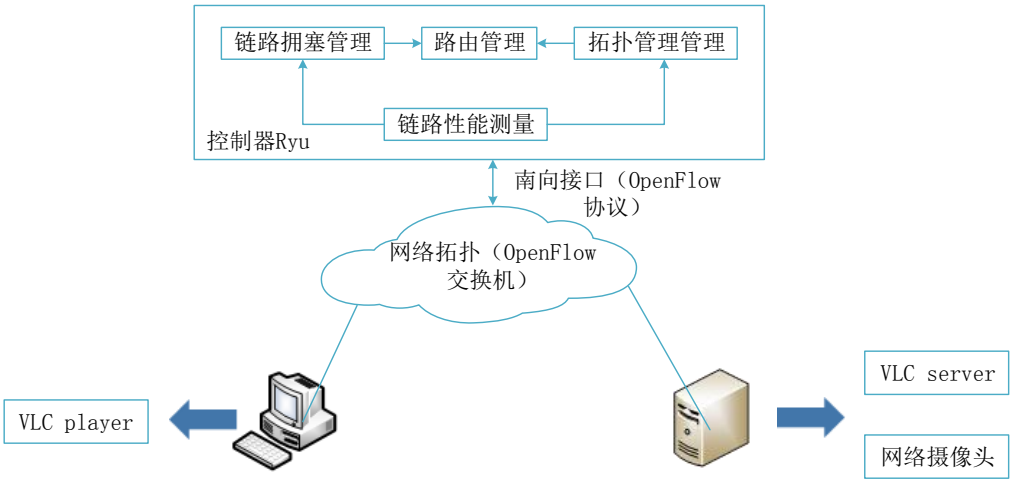


图5.1 视频传输平台结构图

5.2.1 拓扑环境搭建

本文在 Mininet 中通过 python 脚本完成网络拓扑的构建，整个网络由 s1、s2、s3、s4、s5、s6、s7 和 s8 八个虚拟交换机组成，它们都由 Ryu 控制器控制和支配；有 5 个服务器和 2 个客户端，具体连接方式如图 5.2 所示。其中 server1 为流媒体服务器，server2 为其他优先级业务流的服务器，server3、server4、server5 是背景流的服务器，client2 为背景流的接收端。由于 OpenvSwitch 默认的链路带宽为 20Gbps，不符合真实情况，也不利于仿真验证，因此在创建拓扑进行仿真验证时，需要利用 Mininet 对链路带宽进行设置。

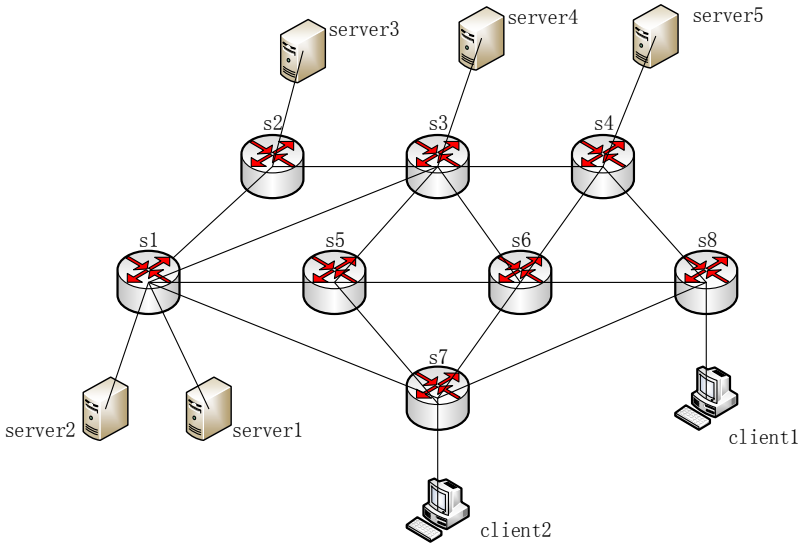


图5.2 仿真实验拓扑图

使用 VMware 创建两台 Ubuntu 虚拟机，分别安装 Ryu 和 Mininet。根据图 5.2 的拓扑连接结构图，在 Mininet 上完成 Python 脚本 cai-topo.py 的编写，如图 5.3 所示，然后在运行脚本的时候连接 Ryu 控制器，当出现图 5.4 所示的情况，且运行拓扑管理模块出现图 5.5 所示的结果，则表示拓扑创建成功且控制器能正确获得底层网络拓扑的连接情况。

```
def __init__(self):
    "Create custom topo."
    # Initialize topology
    Topo.__init__(self)
    # Add  switches
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    s3 = self.addSwitch('s3')
    s4 = self.addSwitch('s4')
    s5 = self.addSwitch('s5')
    s6 = self.addSwitch('s6')
    s7 = self.addSwitch('s7')
    s8 = self.addSwitch('s8')
    # Add hosts
    server1 = self.addHost('server1', ip='10.0.0.1', mac='00:00:00:00:00:01')
    server2 = self.addHost('server2', ip='10.0.0.3', mac='00:00:00:00:00:03')
    server3 = self.addHost('server3', ip='10.0.0.4', mac='00:00:00:00:00:04')
    server4 = self.addHost('server4', ip='10.0.0.5', mac='00:00:00:00:00:05')
    server5 = self.addHost('server5', ip='10.0.0.6', mac='00:00:00:00:00:06')
    client1 = self.addHost('client1', ip='10.0.0.2', mac='00:00:00:00:00:02')
    client2 = self.addHost('client2', ip='10.0.0.7', mac='00:00:00:00:00:07')
    # Add links
    self.addLink(s1,  s2, bw=1.5)
    self.addLink(s2,  s3, bw=1)
    self.addLink(s3,  s4, bw=1.2)
    self.addLink(s4,  s8, bw=2.5)
    self.addLink(s1,  s3, bw=3.2)
    self.addLink(s1,  s5, bw=2.8)
    self.addLink(s5,  s6, bw=1.4)
    self.addLink(s6,  s8, bw=2.5)
    self.addLink(s3,  s6, bw=2.8)
    self.addLink(s1,  s7, bw=1.2)
    self.addLink(s7,  s8, bw=0.9)
    self.addLink(s5,  s7, bw=2)
    self.addLink(s4,  s6, bw=2.5)
    self.addLink(s3,  s5, bw=2)
    self.addLink(s6,  s7, bw=1.4)
    self.addLink(s1,  server1, bw=1.2)
    self.addLink(s1,  server2, bw=1.2)
    self.addLink(s2,  server3, bw=1)
    self.addLink(s3,  server4, bw=2.1)
    self.addLink(s4,  server5, bw=1.3)
    self.addLink(s8,  client1, bw=1)
    self.addLink(s7,  client2, bw=1.4)
```

图 5.3 自定义拓扑程序图

```
caixiaoxin@ubuntu:~/mininet/custom$ sudo mn --custom cai-topo.py --topo mytopo --controller=remote,ip=192.168.1.1 --link tc
[sudo] caixiaoxin 的密码:
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.1.1:6653
*** Adding hosts:
client1 client2 server1 server2 server3 server4 server5
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8
*** Adding links:
(1.50Mbit) (1.50Mbit) (s1, s2) (3.20Mbit) (3.20Mbit) (s1, s3) (2.80Mbit) (2.80Mbit) (s1, s5) (1.20Mbit) (1.20Mbit) (s1, s7) (1.20Mbit) (1.20Mbi
t) (s1, server1) (1.20Mbit) (1.20Mbit) (s1, server2) (1.00Mbit) (1.00Mbit) (s2, s3) (1.00Mbit) (1.00Mbit) (s2, server3) (1.20Mbit) (1.20Mbit) (
s3, s4) (2.00Mbit) (2.00Mbit) (s3, s5) (2.80Mbit) (2.80Mbit) (s3, s6) (2.10Mbit) (2.10Mbit) (s3, server4) (2.50Mbit) (2.50Mbit) (s4, s6) (2.50M
bit) (2.50Mbit) (s4, s8) (1.30Mbit) (1.30Mbit) (s4, server5) (1.40Mbit) (1.40Mbit) (s5, s6) (2.00Mbit) (2.00Mbit) (s5, s7) (1.40Mbit) (1.40Mbit
) (s6, s7) (2.50Mbit) (2.50Mbit) (s6, s8) (1.40Mbit) (1.40Mbit) (s7, client2) (0.90Mbit) (0.90Mbit) (s7, s8) (1.00Mbit) (1.00Mbit) (s8, client1
)
*** Configuring hosts
client1 client2 server1 server2 server3 server4 server5
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8 ...(1.50Mbit) (3.20Mbit) (2.80Mbit) (1.20Mbit) (1.20Mbit) (1.20Mbit) (1.50Mbit) (1.00Mbit) (1.00Mbit) (1.00Mbit) (1.20M
bit) (3.20Mbit) (2.80Mbit) (2.00Mbit) (2.10Mbit) (1.20Mbit) (2.50Mbit) (2.50Mbit) (1.30Mbit) (2.80Mbit) (1.40Mbit) (2.00Mbit) (2.00Mbit) (1.40M
bit) (2.50Mbit) (2.80Mbit) (2.50Mbit) (1.40Mbit) (1.20Mbit) (0.90Mbit) (2.00Mbit) (1.40Mbit) (1.40Mbit) (2.50Mbit) (2.50Mbit) (0.90Mbit) (1.00M
bit)
*** Starting CLI:
mininet> links
s1-eth1<->s2-eth1 (OK OK)
s1-eth2<->s3-eth3 (OK OK)
s1-eth3<->s5-eth1 (OK OK)
s1-eth4<->s7-eth1 (OK OK)
s1-eth5<->server1-eth0 (OK OK)
s1-eth6<->server2-eth0 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s2-eth3<->server3-eth0 (OK OK)
s3-eth2<->s4-eth1 (OK OK)
s3-eth5<->s5-eth4 (OK OK)
s3-eth4<->s6-eth3 (OK OK)
s3-eth6<->server4-eth0 (OK OK)
s4-eth3<->s6-eth4 (OK OK)
```

图5.4 运行结果及链路信息图

```
----------------------Link Port----------------------
    switch          1          2          3          4          5          6          7          8
         1         NO     (1, 1)     (2, 3)         NO     (3, 1)         NO     (4, 1)         NO
         2     (1, 1)         NO     (2, 1)         NO         NO         NO         NO         NO
         3     (3, 2)     (1, 2)         NO     (2, 1)     (5, 4)     (4, 3)         NO         NO
         4         NO         NO     (1, 2)         NO         NO     (3, 4)         NO     (2, 1)
         5     (1, 3)         NO     (4, 5)         NO         NO     (2, 1)     (3, 3)         NO
         6         NO         NO     (3, 4)     (4, 3)     (1, 2)         NO     (5, 4)     (2, 2)
         7     (1, 4)         NO         NO         NO     (3, 3)     (4, 5)         NO     (2, 3)
         8         NO         NO         NO     (1, 2)         NO     (2, 2)     (3, 2)         NO
```

图5.5 控制器终端拓扑连接展示图

5.2.2 视频服务器搭建

本文使用 VLC media player 作为视频服务器，使用外置摄像头 Z-star Gsou USB2.0 Camera 动态获取 640*480 分辨率的实时视频，通过 VLC media player 将摄像头捕获的实时视频发送到 Mininet 虚拟的网络中。具体步骤如下：

使用 Gsou USB 2.0 Camera 连接电脑，并将其挂载至 Ubuntu 虚拟机，在 Ubuntu 中打开 VLC media player，将其捕获的视频信息使用 RTSP 协议发送至/camera 路径的 8854 端口。

在客户端中使用 VLC media player 的客户端模式连接至 rtsp://10.0.0.1:8854/camera，使用 Wireshark 监控客户端和服务器之间的视频传输，结果如图 5.6 所示，说明视频服务器搭建成功且客户端能够与视频服务器建立正确的连接并正常通信。

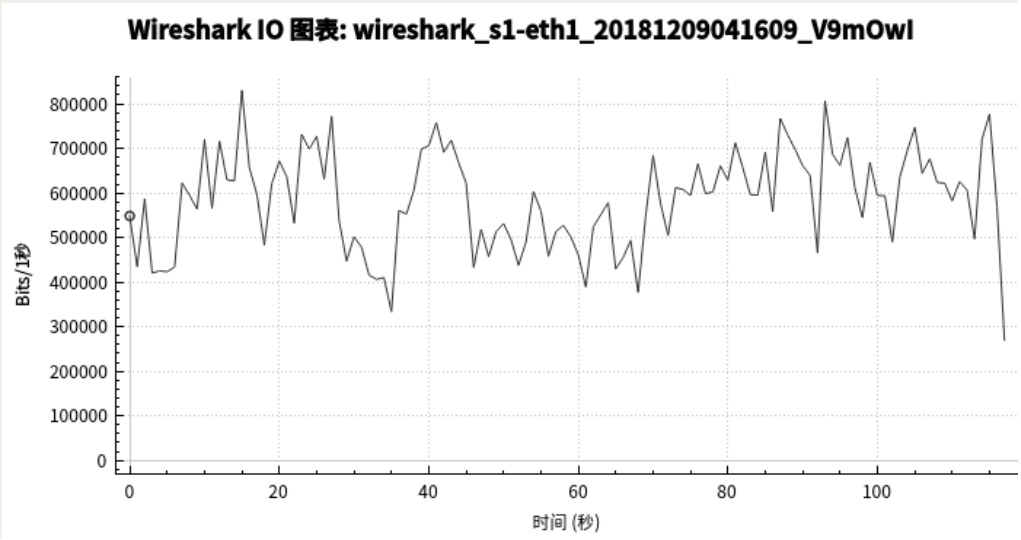


图5.6 Wireshark监控的传输情况图

5.3 实验仿真与结果分析

基于本文对视频流媒体 QoS 控制策略的研究，设计了四组仿真实验分别对 QoS 路由计算的有效性、动态路由的有效性、队列调度的 QoS 控制性能、QoS 控制策略性能进行仿真验证，用以说明 QoS 控制策略对视频流媒体传输性能的提高。

为了模拟真实网络中的流量情况，在每个仿真实验开始之前，分别使用 iperf[50]由 server3、server4、server5 向 client2 发送 0.5Mbps 的背景流，在以后的实验中不再说明。

5.3.1 QoS 路由计算有效性分析

链路的时延反映了当前链路的传输效率，丢包率则反映了链路数据传输的可靠性，它们是衡量流媒体传输性能的重要指标。对于视频流媒体业务来说，其对网络的传输环境具有很高的要求，为其选择较低丢包率和较小时延的路径进行传输，才能保证其高效的传输以及对端较好的观看效果。仿真实验的流程如下：

1)、启动控制器和 Mininet 仿真工具，通过脚本创建图 5.2 的实验拓扑，并连接控制器。

2)、分别在 server2 和 client1 启动 iperf 工具，通过其模拟向 SDN 网络中不断发送 1Mbps 的其他类型业务流。

3)、分别在 server1 和 client1 上启动 VLC 的服务端和客户端，通过 SDN 网络发送和接收视频流媒体数据流。

4)、通过控制器监控网络中路径的时延与丢包率变化情况，并使用 matplotlib 库分别绘制遗传算法和 Dijkstra 算法所选路径上的时延与丢包率的变化图，如图 5.7 和 5.8 所示。

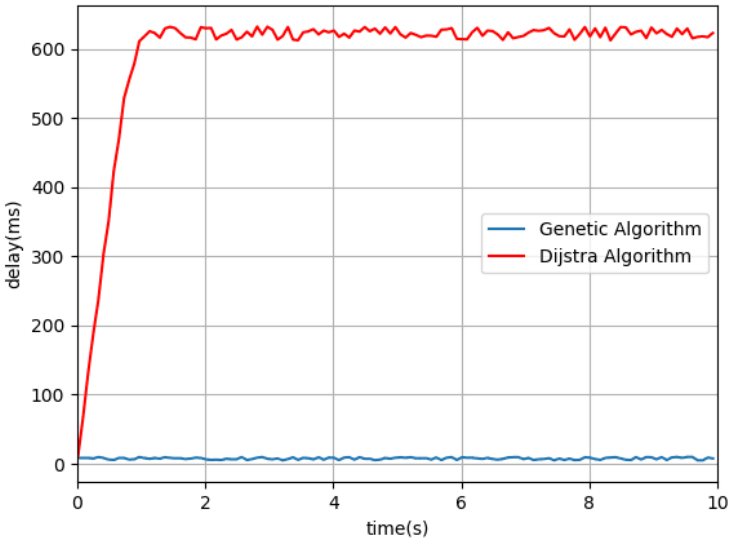


图5.7 时延变化情况图

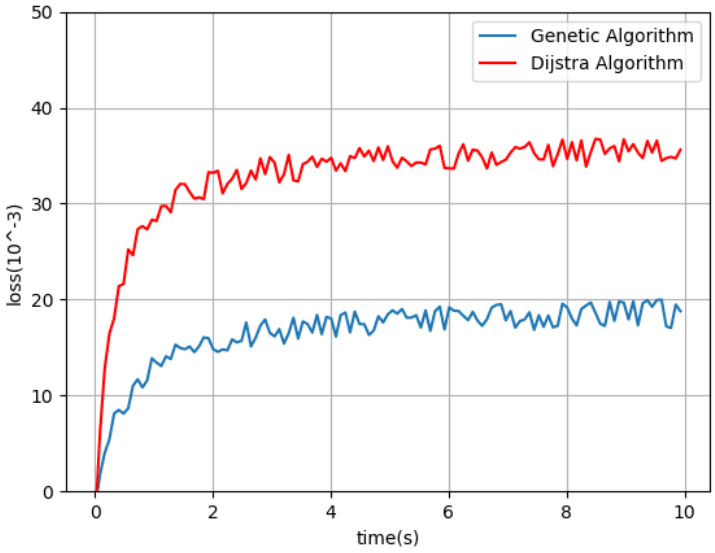


图5.8 丢包率变化情况

从图中可以看出，当视频流和其他类型的流进入 SDN 网络中之后，路径的时延和丢包率逐渐变大，一段时间之后，虽然二者都逐渐趋于稳定，但使用遗传算法所选路径的时延和丢包率都小于使用 Dijkstra 算法所选的路径，并且使用遗传算法所选路径的时延和丢包率的值均在设定的 QoS 指标范围内，即丢包率在20×10^{-3}以内，时延在 10ms 以内。综上所述，基于遗传算法的 QoS 路由计算能够提高视频流的传输性能。

5.3.2 动态路由有效性分析

遗传算法的 QoS 路由计算仅仅能保证视频流在初次选路时有较好的 QoS 保障，由于网络中的环境是多变的，一旦视频流在传输过程中遇到洪峰，视频流的 QoS 将会受到极大的影响，此时必须使用动态路由策略为其优化路径，才能保证其 QoS。通过传输过程中

是否使用了动态路由策略，对比视频流的时延抖动、吞吐量以及 PSNR 的变化情况，验证动态路由策略的有效性。为了便于仿真实验的验证，需要将 s1-s7-s8 链路的带宽提高，使基于遗传算法选出的路径与最短跳数的路径相同，将 s1-s7 间链路带宽设置为 2.8Mbps，s7-s8 间链路带宽设置为 3Mbps。

时延抖动用于衡量网络时延的稳定性，吞吐量用于表明单位时间内成功传输的数据大小。网络拥塞会导致时延抖动过大以及吞吐量过小，若持续时间过长，将会影响视频端到端的传输。仿真实验的流程如下：

1）、使用 client1 分别向视频服务器 server1 的 8854 和 8855 端口请求动态视频，其中 8854 端口的视频使用动态路由策略，8855 端口的视频流不使用动态路由策略。使用 iperf 分别记录两种视频流传输过程中的时延抖动和吞吐量。

2）、在流媒体视频传输的第 10s，通过 iperf 工具模拟向网络中发送 3Mbps 的其他类型业务流。

3）、使用 iperf 监控网络中时延抖动和吞吐量的变化情况，图 5.9 与图 5.10 分别描述了有无动态路由策略时，视频流的吞吐量和时延抖动随时间的变化趋势。

4）、在客户端获取前 40s 传输的视频，使用 ffmpeg 进行解码，并计算出两个视频的峰值信噪比(PSNR)，比较的结果如图 5.11。

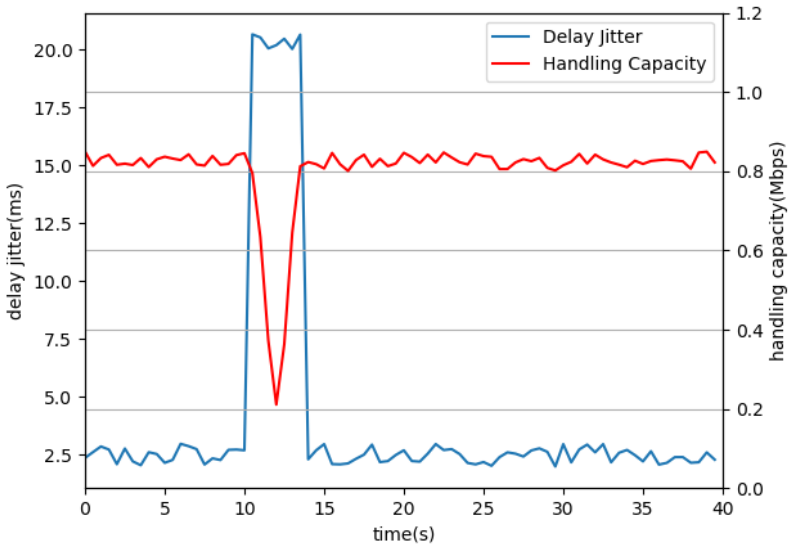


图 5.9 视频流在网络拥塞时有动态路由策略的时延抖动与吞吐量变化图

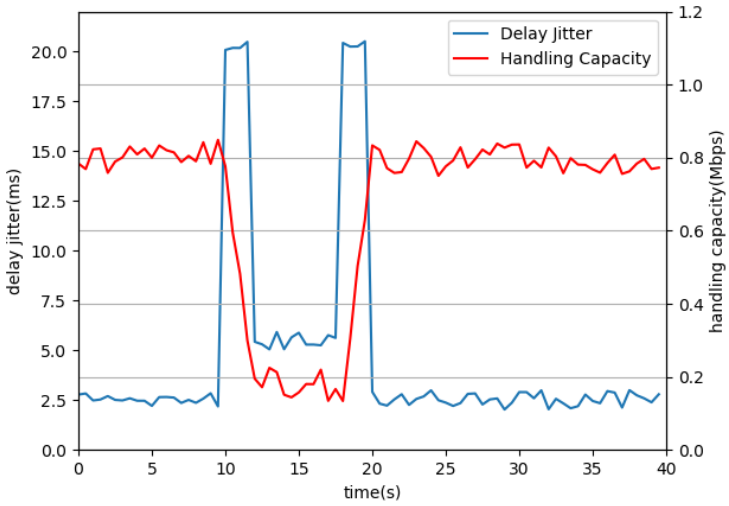


图5.10 视频流在网络拥塞时无动态路由策略的时延抖动与吞吐量变化图

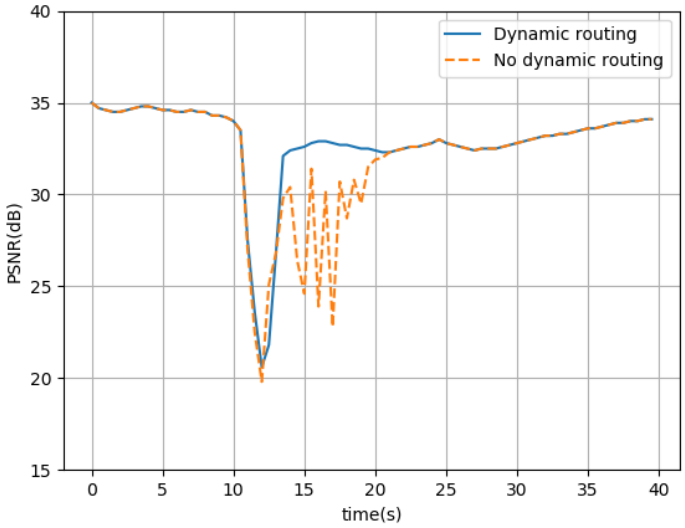


图 5.11 PSNR 对比图

从图 5.9 和 5.10 中可以看到，在第 10s 时，因注入其他类型业务流致使网络中的流量突增，进而使得传输链路出现拥塞，视频流媒体的吞吐量迅速下降，时延抖动增加，若使用动态路由策略，控制器监测到链路拥塞后即会触发动态路由机制，使用遗传算法计算新的满足 QoS 需求的路径，因此在 13 秒后视频流媒体吞吐量逐渐上升，其时延抖动在 4s 后回落至稳定状态。而未使用动态路由策略的传输，只是在最初为视频流媒体选出一条符合 QoS 需求的路径，并未在网络状态发生变化时动态地改变传输路径，各个优先级的数据平等的争夺路径中链路的带宽，因此在网络拥塞后，网络中所有数据流“互不相让”，视频流媒体的吞吐量在 18 秒后才逐渐向稳定状态恢复，而时延抖动也是经历了两次跳跃才恢复至稳定。

峰值信噪比（Peak Signal Noise Ratio，PSNR）能判别出接收端接收的视频质量的优劣。从图 5.11 中两个视频流的 PSNR 对比可知，当视频流出现下降时，即传输路径出现

拥塞时，动态路由机制能够使视频流的传输状态在较短的时间内恢复，而没有使用动态路由机制的视频流则恢复的较慢。两个视频流的 PSNR 对比同样验证了动态路由机制的有效性。

5.3.3 队列调度的 QoS 控制性能测试

传输速率是 QoS 的重要指标，业务接收速率与发送速率是否相同反映了网络是否提供给业务流最好的传输保障。本次性能测试获取在不同发送速率下，分类业务的接收速率，以检测所提出的队列调度能否优先保障视频流媒体业务的 QoS。

为了测试调度策略的控制性能，首先排除监控视频速率的不稳定性，为此四种优先级不同的数据流都由 iperf 来模拟；其次需要排除 QoS 路由的干扰，为此使用 Dijkstra 算法基于跳数为数据流计算路径。当 server1 向 client1 发送报文时，业务报文将通过路径 server1->s1->s7->s8->client1 进行传输。由于背景流传输路径不固定，因此需要对所有交换机的出端口进行队列策略的配置，同时设置交换机间的链路带宽为 16Mbps。图 5.12 是配置的一个端口队列规则示意图，其他交换机端口队列规则配置与其一样。

```
sudo ovs-vsctl set port s7-eth2 qos=@caiqos
-- --id=@caiqos create qos type=linux-htb
       other-config:max-rate=16000000 queue=0=@q0, 1=@q1, 2=@q2, 3=@q3 4=@q4
-- --id=@q0 create queue dscp=32
       other-config:max-rate=7900000, burst=790000, priority=0, quantum=790000
-- --id=@q1 create queue dscp=24
       other-config:max-rate=4500000, burst=450000, priority=1, quantum=450000
-- --id=@q2 create queue dscp=16
       other-config:max-rate=2200000, burst=220000, priority=2, quantum=220000
-- --id=@q3 create queue dscp=8
       other-config:max-rate=900000, burst=90000, priority=3, quantum=90000
-- --id=@q4 create queue dscp=0
       other-config:max-rate=500000, burst=50000, priority=4, quantum=50000
```

图5.12 队列规则配置图

server1 利用 iperf 工具向 client1 发送 6 组 UDP 数据，6 组报文的发送速率都是从 1000kbps 增加到 7000kbps，每次的递增值为 1000kbps，每组内不同优先级业务数据的发送速率相同，每组报文传输的时间相同，最后计算 6 组报文的平均接收速率。图 5.13 和图 5.14 分别展示了调度策略配置与否下的不同优先级报文接收速率的情况。

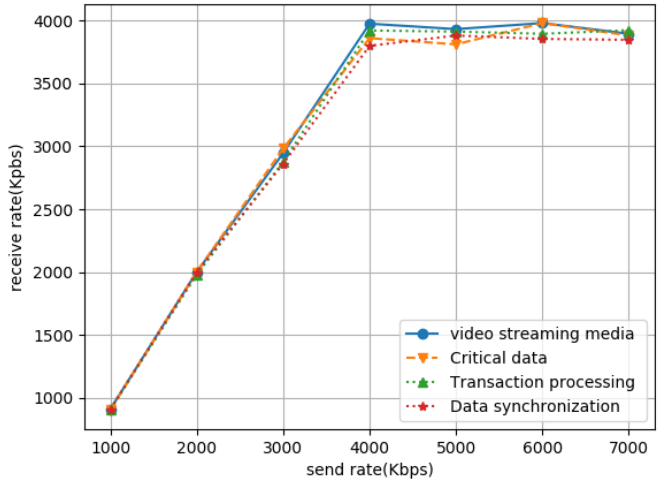


图5.13 未配置调度策略的接收速率图

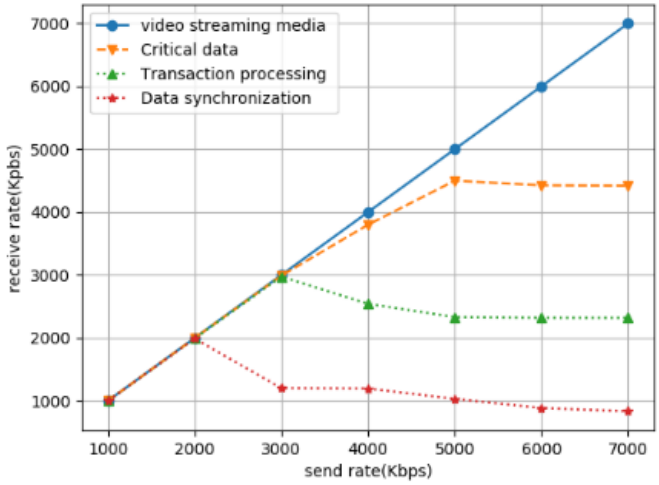


图5.14 配置了调度策略的接收速率图

从图 5.13 中可以看到，在没有调度策略时，初始时，各种业务数据的接收速率和发送速率基本相同；在发送速率为 4000Kbps 左右时，接收速率的值基本都在 4000Kbps 及以下；之后尽管发送速率的值在不断增加，四种业务报文的接收速率却都在 3800Kbps 上下浮动。由此可看出，没有调度策略控制时，不同优先级业务之间是平等的竞争关系，网络并未提供不同优先级业务的服务质量保证。

从图 5.14 中可以看到，配置了调度策略后，在发送速率增大的过程中，除最高优先级的视频流媒体外，其他优先级业务的接收速率均有所下降。优先级最低的数据同步类业务的接收速率在发送速率超过 2000Kbps 后就已经开始下降，它得到的服务最差；接着是优先级第三的事务处理类业务，其接收速率在发送速率超过 3000Kbps 时开始下降，直到达到稳定的接受速率；优先级次高的关键数据类业务报文在发送速率大于 5000Kbps 时开始下降。这说明队列调度策略有效地保障了视频流媒体的 QoS，并能实现不同优先级业务的 QoS 保障。

从图 5.11 队列配置图中可以看到，优先级最低的数据同步类业务报文设置的最大传输速率为 900Kbps，但是在发送速率为 2000Kbps 时，其接收速率也为 2000Kbps，是因为此时网络中的带宽还有很大的盈余，HTB 启用了借带宽的策略用于保证其传输，但是随着报文发送速率逐渐增大，同步类业务报文的接收速率逐渐减小，是因为网络中的剩余带宽逐渐减少，为保证高优先级报文的传输，逐渐增加了对数据同步类报文的限制，由于其优先级最低，所以在传输路径拥塞时接收速率最先被限制。

综上分析可知，队列调度策略可有效保障视频流媒体类业务的传输，并在网络带宽充足的情况下提供借带宽的的机制，尽力保障其他业务流的传输。

5.3.4 QoS 控制策略性能分析

上述三个仿真实验从三个方面验证了 QoS 控制策略对视频流媒体传输性能的提高，本仿真实验将其综合起来，以验证 QoS 控制策略框架的整体性能。

1）、设置队列调度规则，图 5.15 是配置的一个端口队列规则示意图，其他交换机端口队列规则配置操作与其一样。端口的配置规则为视频流媒体、其他三个优先级业务流及背景流所占带宽分别是根带宽的 50%、20%、15%、10%以及 5%。

```
sudo ovs-vsctl set port s4-eth3 qos=@caiqos
-- --id=@caiqos create qos type=linux-htb
       other-config:max-rate=2800000 queue=0=@q0, 1=@q1, 2=@q2, 3=@q3  4=@q4
-- --id=@q0 create queue dscp=32
       other-config:max-rate=1400000, burst=140000, priority=0, quantum=140000
-- --id=@q1 create queue dscp=24
       other-config:max-rate=560000, burst=56000, priority=1, quantum=56000
-- --id=@q2 create queue dscp=16
       other-config:max-rate=420000, burst=42000, priority=2, quantum=42000
-- --id=@q3 create queue dscp=8
       other-config:max-rate=280000, burst=28000, priority=3, quantum=28000
-- --id=@q4 create queue dscp=0
       other-config:max-rate=140000, burst=14000, priority=4, quantum=14000
```

图 5.15 队列规则配置图

2）、分别在 server2 和 client1 启动 iperf 工具，通过其模拟向 SDN 网络中不断发送其他类型的业务流，其他类型数据流的相关设置如表 5.2 所示。

表 5.2 数据流相关设置

数据流类型	发送速率	发送端口	TOS 值
关键数据类业务	0.4Mbps	5000	96
事务处理类业务	0.4Mbps	5001	64
数据同步类业务	0.4Mbps	5002	32

3）、分别在 server1 和 client1 上启动 VLC 的服务端和客户端，通过 SDN 网络发送和接收视频流媒体数据流。

4）、使用 iperf 工具监控网络中业务流吞吐量的变化情况，图 5.16 为网络中业务流的吞吐量随时间的变化情况。

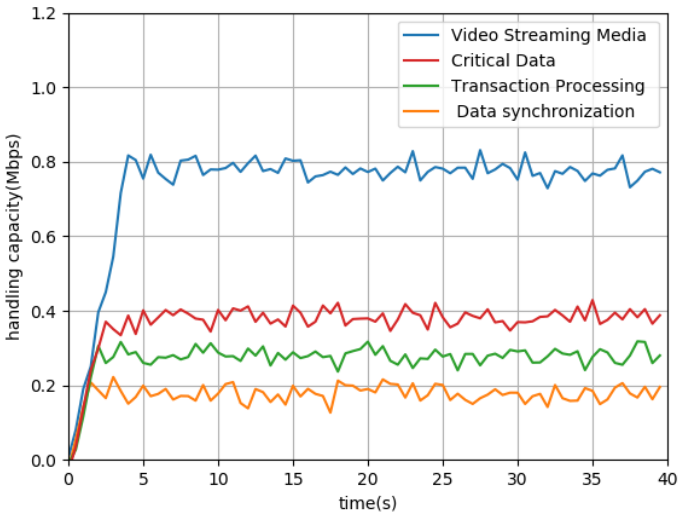


图 5.16 吞吐量变化情况

从图 5.16 中可以看出，Q2、Q3 以及 Q4 三种业务流的发送速率相同，即 0.4Mbps，传输的路径也相同，即跳数最短的路径，吞吐量却依次降低，这是因为在交换机的各个端口对不同优先级的业务流的速率以及出端口的顺序进行了设置，如图 5.15 所示。这说明队列调度策略可以根据业务报文的类型来保障不同的 QoS 需求。通过视频流的吞吐量与其他三种业务流的吞吐量对比可看出，在网络条件相同的情况下，视频流的吞吐量高于其他三种业务流，这说明使用基于遗传算法的 QoS 路由可以为视频流选择性能更好的路径进行传输。

5.4 本章小结

本章完成了本文 QoS 控制策略的仿真实验，并分析了仿真结果。先对传输环境、网络拓扑结构以及相关设置给予了说明，然后对 QoS 路由、动态路由、队列调度策略以及整体控制策略的有效性进行了分析。通过相应 QoS 指标对比表明，本文的 QoS 控制策略可以较好地保障视频流媒体的传输性能，并可根据业务优先级的不同，在端口保障不同优先级业务报文的 QoS。

6 总结与展望

6.1 总结

随着网络技术的升级创新，互联网上的业务流将会越来越多，流媒体业务的应用也将会逐渐占据着网络流量的半壁江山，这类业务对 QoS 的需求非常严格，然而传统网络架构却存在着诸多不能满足流媒体业务传输的弊端，导致流媒体业务的传输性能较差。而 SDN 网络架构转控分离、可编程的特点，简化了网络管理的复杂性，并能灵活地调用网络资源，与传统网络相比，SDN 为提高流媒体视频的传输提供了可行的方案。因此本文基于 SDN 技术，通过将智能算法应用到 SDN 网络以及将新的队列调度算法应用到底层转发设备上，提高视频流媒体的传输性能。通过以上的介绍，本文主要研究工作内容如下：

第一，控制层的 QoS 路由控制策略。QoS 路由策略利用 SDN 网络的可编程性以及能够周期地获得网络链路信息的特点，将遗传算法应用于 QoS 路由选路，并能执行动态路由的策略，在网络拥塞时，为视频流媒体计算新的传输路径，同时优化了流表更新的顺序，解决了流表一致性的问题，避免动态路由造成的流传输中断以及丢包等问题。

第二，转发层的队列调度策略。队列调度策略根据 HTB 调度规则在转发端口处给视频流媒体分配较大带宽并给予其优先转发的“权利”，在底层保证视频流媒体业务的 QoS；且带宽充足时，不同优先级的业务间可以相互借带宽，进而保障不同优先级业务的服务质量。

第三，利用 Ryu、摄像头等软硬件模拟搭建了 SDN 中视频流媒体的传输环境，在环境中实现了本文的控制策略，从算法选路的优越性、动态路由的有效性、底层调度策略的实现以及 QoS 控制策略的有效性做出了测试。仿真实验表明，本文提出的控制策略能够较好地提高视频流媒体的传输性能，降低拥塞造成的视频不流畅等问题，达到了优化视频流媒体传输性能的目的。

6.2 展望

本文利用 SDN 网络架构的可编程性及可周期获取链路信息的特点，提出了视频流媒体的 QoS 控制策略，比较有效地提高了视频流媒体的传输性能。但尚有不足之处需要改进和进一步研究。主要包括：

第一，队列调度策略的配置是静态的，之后需要在交换机转发端口实现动态配置队列调度规则的功能。

第二，本文使用的网络拓扑结构相较于真实的网络来说比较简单，真实的网络拓扑结构通常是规模较大且较为复杂的，像数据中心的网络结构就比较庞杂，因此本文所涉及的视频流传输控制策略还需要在其他复杂网络拓扑中实现仿真验证。

第三，控制策略只在软件形式的 SDN 网络中进行了仿真验证，由于真实环境中的某些因素在模拟环境中不一定能体现出来，有必要在真实网络环境中进行进一步的验证和完善。

参考文献

［1］第 42 次中国互联网络发展状况统计报告 [R/OL]. http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201808/t20180820_70488.htm

［2］冯径，马小骏，顾冠群．适应 QoS 路由机制的网络模型研究[J]. 计算机学报,2000,23(8):799~805.

［3］左青云，陈鸣，赵广松,等．基于 OpenFlow 的 SDN 技术研究[J]. 软件学报，2013(5):1078-1097.

［4］ Gavras A , Karila A , Fdida S , et al. Future internet research and experimentation[J]. ACM SIGCOMM Computer Communication Review, 2007, 37(3):89.

［5］ Elliott C . GENI: Opening Up New Classes of Experiments in Global Networking[J]. IEEE Internet Computing, 2010, 14(1):39-42.

［6］ Farhady H, Lee H Y, Nakao A. Software-Defined Networking: A survey[J]. Computer Networks, 2015, 81(C):79-95.

［7］Specication OpenFlow,OpenFlow Switch.Version 1.5.0.(WireProtocol0x05)[S].[S.l.],2013.

［8］ Ryu[EB/OL].[2011].http://ryu.readthedocs.io/en/latest/getting_started.html.

［9］ Noskov A A, Nikitinskiy M A, Alekseev I V. Development of an active external network topology module for Floodlight software-defined network controller[J]. Automatic Control & Computer Sciences, 2016, 50(7):546-551.

［10］许名广，刘亚萍，邓文平．网络控制器 OpenDaylight 的研究与分析[J]. 计算机科学，2015, 42(S1):249-252.

［11］Jain S, Kumar A, Mandal S, et al. B4: experience with a globally-deployed software defined wan[C]//ACM SIGCOMM 2013 Conference on SIGCOMM.[S.l.],2013:3-14.

［12］Chi Yaohong, Srikanth Kandula, Ratul Mhajan, et al. Achieving .High Utilization with Software-Driven WAN[C]. Hong Kong, China, SIGCOMM,2013,8.

［13］黄韬，张丽，张云勇,等.基于 OpenFlow 的 SVC 流媒体时延自适应分级传输方法[J]. 通信学报, 2013(11):121-128.

［14]陈志钢．基于 Open Flow 和 MTR 的 IP 网络流量控制方法的研究．电子科技大学，2013.

［15］徐李谦．基于 SDN 的视频流调度机制研究.重庆邮电大学，2017.

［16］李建冲.基于 SDN 的业务服务质量保障技术研究.西安电子科技大学，2017.

［17］K.-W. Kwong, R. Guerin, A. Shaikh, and S. Tao. Improving service differentiation in IP networks through dual topology routing. Proc. Co NEXT, December 2007, pp. 26-28.

［18］Civanlar S, Parlakisik M, Tekalp A M, et al. A QoS-enabled Open Flow environment for

Scalable Video streaming[C]. GLOBECOM Workshops. IEEE, 2010:351-356.

［19］N.Handigol, S.Seetharaman, M. Flajslik, N. Mc Keown, and R. Johari. Plug-n-Serve:Load-Balancing Web Traffic using Open Flow. In ACM SIGCOMM Demo, August 2009, pp.268-270.

［20］Nikhil Handigol,Srini Seetharaman, Mario Flajslik, Aaron Gember,Aster*x: Load-Balancing Web Traffic over Wide-Area Networks. http://www.standford.edu/.

［21］gilmez 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.

［22］Dobrijevic O, Santl M, Matijasevic M. Ant colony optimization for QoE-centric flow routing in software-defined networks[C]//International Conference on Network & Service Management. 2015.

［23］Owens H, Durresi A . Video over Software-Defined Networking (VSDN)[J]. Computer Networks, 2015, 92:341-356.

［24］王嗣平. 基于 SDN 网络的视频流传输控制系统设计与实现[D]. 国防科学技术大学, 2016.

［25］QoS. http://zh.wikipedia.org/wiki/QoS, June 19, 2012.

［26］田冲. 基于 IP 网络的 QoS 队列调度算法研究[D]. 南京邮电大学, 2013.

［27］武莹, 王敬宇.基于 OpenDaylight 的虚拟网络 QoS 控制系统的设计与实现[J]. 2018.

［28］吴慧, 陈秋红,刘国辉. OpenFlow 网络基于 DiffServ 模型的 QoS 管理机制的实现[J]. 电视技术, 2014, 38(5):113-115.

［29］崔潇宇,沈庆国. MPLS 网络中基于 QoS 约束的路由和准入控制[J]. 通信技术, 2018, v.51；No.318(06):102-105.

［30］Kutscher D, Ahlgren B, Karl H, et al. 10492 Executive Summary -- Information-Centric Networking[J]. 2011.

［31］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.

［32］McKeown N. Software-Defined Networking[R/EB]. In:Proc.of the INFOCOM Key Note. 2009. http://infocom2009.ieee-infocom.org/technical Program.htm.

［33］房秉毅,张歌,张云勇,等.开源 SDN 控制器发展现状研究[J].邮电设计技术,2014(07):29-36.

［34］基于 SDN 的数据中心流量工程研究[D]. 电子科技大学, 2016.

［35］许莹. 基于区分服务（DiffServ）的 IP QoS 控制策略研究[D]. 湖南大学, 2003.

［36］雷鸣. 基于 SDN 数据中心的流量调度算法研究[D]. 西安工业大学，2018.

［37］Book R V. Book Review: Computers and intractability: A guide to the theory of NP-completeness[J]. Bulletin of the American Mathematical Society, 1980, 3(2):898-905.
［38］李晓方.混合 SDN 的流量矩阵估计和路由优化研究[D].中国科学技术大学,2015.
［39］董晓林.面向 SDN 的多路径调度算法研究[D].郑州大学,2017.
［40］朱冠宇. 基于遗传算法的 QoS 路由选择策略研究[D]. 华中科技大学, 2004.
［41］文强.SDN 网络业务量工程技术研究[D]. 电子科技大学, 2016.
［42］陈忠. SDN 网络中 QoS 控制技术研究与实现[D]. 2017.
［43］孔智. 基于三层交换机服务质量的研究[D]. 2016.
［44］杨俊超. 基于 OpenFlow 网络的 QoS 管理策略研究[D]. 2015.
［45］林玉侠, 朱慧玲,马正新,等.QoS 路由度量参数的选择问题研究[J]. 电信科学, 2003, 19(7):22-27.
［46］周怡.基于 OpenFlow 的 SDN 网络 QoS 路由策略研究[D]. 2018.
［47］Chang W A, Ramakrishna R S.A genetic algorithmfor shortest path routing problem and the sizing of populations[M]. IEEE Press, 2002.
［48］Team M. Mininet:An instant virtual network on your laptop(or other PC) [DB].2012.
［49］黄家玮,韩瑞,钟萍,王建新.基于 Mininet 的计算机网络实验教学方案[J].实验技术与管理,2015,32(10):139-141.
［50］张白,宋安军.基于 Iperf 的网络性能测量研究[J].电脑知识与技术, 2009,5(36): 27-29.

攻读硕士学位期间发表的论文

［1］才新,李静.基于 SDN 的视频流媒体路由选择[J].国外电子测量技术,2018,(11):87-90.

致 谢

从 2016 年 9 月入学到现在毕业论文完成之际，作者在西安工业大学度过了三个春夏秋冬，日月如梭，研究生的生涯已接近尾声。三年的时间里，有过生活上的失落迷茫，有过科研上的艰难困苦，有过论文完成过程中的重重阻碍，幸而身边有这么多的良师益友，他们给了作者很多的帮助，在这里作者想要向他们表达最真挚的感谢。

首先要感谢作者的导师李静教授，李老师是一名非常负责任的老师，研一时，李老师要求作者要好好学习，并给作者推荐了多个领域的书籍，扩大作者的知识广度。在研二以后的科研工作中，李老师同样给予了作者很大的帮助，帮助作者解答了很多学术上的疑惑。李老师在学习和科研方面对作者严格要求，但在生活上，李老师对作者十分的关心和爱护。读研这三年，李老师严谨的科学态度、宽厚待人的品质对作者的影响较为深刻，在未来的工作和生活中，作者会一直向李老师学习。

还要感谢雷鸣教授，雷老师是一个知识渊博且非常幽默的老师，感谢雷老师在作者论文完成过程中给予的专业性的指导，为作者的论文提出了很多宝贵的意见。

其次要感谢室友和实验室的同学们，在枯燥的科研生活中，我们互相勉励，互相陪伴，一起度过了美好的三年校园时光，感谢他们在我沮丧无助时给予我的力量，让我重拾信心。特别要感谢实验室的朱文斌同学，感谢他在作者论文完成过程中给予的帮助。

最后感谢参加论文评阅和答辩工作的老师们，感谢您们认真评阅我的论文和参加我的论文答辩。

学位论文知识产权声明

本人完全了解西安工业大学有关保护知识产权的规定，即：研究生在校攻读学位期间学位论文工作的知识产权属于西安工业大学。本人保证毕业离校后，使用学位论文工作成果或用学位论文工作成果发表论文时署名单位仍然为西安工业大学。学校有权保留送（提）交的学位论文，并对学位论文进行二次文献加工供其他读者查阅和借阅；学校可以在网络上公布学位论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存学位论文。（保密的学位论文在解密后应遵守此规定）

学位论文作者签名：

指导教师签名：

日期：

学位论文独创性声明

秉承学校严谨的学风与优良的科学道德，本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的成果，不包含本人已申请学位或他人已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了致谢。

学位论文与资料若有不实之处，本人承担一切相关责任。

学位论文作者签名：

指导教师签名：

日期：