**Research on Video Streaming Media Transmission Performance based on SDN Network**

**Discipline：**Pattern Recognition and Intelligent System

**Student Signature：**

**Supervisor Signature：**

**Abstract**

In recent years, with the rapid development of the Internet and multimedia technology, video streaming media applications on the Internet shine brilliantly, which puts forward higher requirements for the transmission performance of the network, However, the drawbacks of traditional network often cause a lot of packet loss or even interruption in the transmission process of video, which seriously affects the service quality of video transmission. As a new network architecture, software-defined Networking (SDN) provides a new train of thought for traffic engineering, QoS routing and other problems in traditional network by taking advantages of the separation of control, centralized control and programmable. In this paper, the performance of video streaming media in SDN is studied by using the characteristics of SDN, The main research contents are as follows:

First of all, this paper makes a priority distinction between four different kinds of businesses, and sets different priority values, and puts forward the QoS control strategy of video streaming media. Then, the QoS routing based on genetic algorithm calculates 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 Dijstra algorithm, and when the controller detects congestion in the transmission path of the video streaming media, take the rerouting measure to better guarantee the QoS of video streaming media. Secondly, the HTB queue rules are used to differentiate schedules for different priority services on the OpenFlow switch, so as to guarantee QoS of video streaming media businesses firstly.

Finally, the Topology Management module, link information measurement module, routing management module and queue scheduling module of QoS control framework are implemented respectively, and the QoS control strategy is tested in SDN network transmission environment built by Mninet, Ryu controller, camera and other software. Through a series of simulation experiments, the feasibility of this control strategy is verified from the aspects of transmission performance parameters of video streaming media such as chain delay jitter and throughput rate. Simulation experimental results show that QoS control strategy can select a path that meets the requirements of video streaming media for transmission and can dynamically route video streaming media, and can distinguish the QoS of different priority business in the data forwarding layer, and ensure the QoS requirement of the highest priority service end-to-end.

**Key Words：**SDN; Video streaming media; QoS routing; Genetic algorithm; Queue scheduling

目 录

[1 绪 论 1](#_Toc534641155)

[1.1 研究背景及意义 1](#_Toc534641156)

[1.2 国内外研究现状 2](#_Toc534641157)

[1.2.1 SDN研究现状 2](#_Toc534641158)

[1.2.2 SDN网络中视频流媒体QoS控制策略的研究现状 2](#_Toc534641159)

[1.2.3 存在的问题 3](#_Toc534641160)

[1.3 论文的主要研究内容及结构安排 4](#_Toc534641161)

[1.3.1 论文的主要内容 4](#_Toc534641162)

[1.3.2 论文的结构安排 5](#_Toc534641163)

[2 SDN中QoS的相关技术分析 6](#_Toc534641164)

[2.1 QoS技术 6](#_Toc534641165)

[2.1.1 QoS的定义 6](#_Toc534641166)

[2.1.2 QoS服务模型及机制 6](#_Toc534641167)

[2.2 传统网络下的流媒体传输分析 7](#_Toc534641168)

[2.3 SDN相关技术 7](#_Toc534641169)

[2.3.1 SDN控制器 9](#_Toc534641170)

[2.3.2 OpenFlow 9](#_Toc534641171)

[2.3.3 OpenFlow对QoS的支持 13](#_Toc534641172)

[2.3.4 智能算法在SDN上运行的可行性分析 14](#_Toc534641173)

[2.4 本章小结 14](#_Toc534641174)

[3 QoS控制策略需求分析与设计 16](#_Toc534641175)

[3.1 需求分析 16](#_Toc534641176)

[3.1.1 应用环境分析 16](#_Toc534641177)

[3.1.2 应用需求分析 16](#_Toc534641178)

[3.2 QoS控制策略框架 17](#_Toc534641179)

[3.2.1 数据流分类 17](#_Toc534641180)

[3.2.2 QoS控制策略的架构设计 18](#_Toc534641181)

[3.3 QoS控制框架模块分析设计 20](#_Toc534641182)

[3.3.1 QoS路由框架 20](#_Toc534641183)

[3.3.2 队列调度策略的分析设计 28](#_Toc534641184)

[3.4 基于遗传算法的多QoS约束路由算法分析与设计 31](#_Toc534641185)

[3.4.1 路由度量数学模型 31](#_Toc534641186)

[3.4.2 遗传算法基本概念 31](#_Toc534641187)

[3.4.3 染色体编码设计 32](#_Toc534641188)

[3.4.4 种群初始化方法设计 33](#_Toc534641189)

[3.4.5 适应度函数设计 33](#_Toc534641190)

[3.4.6 遗传算子的设计 34](#_Toc534641191)

[3.5 本章小结 37](#_Toc534641192)

[4 QoS控制策略实现 38](#_Toc534641193)

[4.1 QoS控制框架实现概述 38](#_Toc534641194)

[4.2 QoS路由的实现 39](#_Toc534641195)

[4.2.1 链路性能测量模块的实现 39](#_Toc534641196)

[4.2.2 拓扑管理模块的实现 44](#_Toc534641197)

[4.2.3 路由管理模块 45](#_Toc534641198)

[4.2.4遗传算法的实现 47](#_Toc534641199)

[4.3 队列调度策略的实现 55](#_Toc534641200)

[4.4 本章小结 58](#_Toc534641201)

[5 实验仿真与结果分析 59](#_Toc534641202)

[5.1 仿真实验环境介绍 59](#_Toc534641203)

[5.1.1 网络仿真软件Mininet 59](#_Toc534641204)

[5.1.2 OpenvSwitch 60](#_Toc534641205)

[5.1.3 其他实验环境及相关软件介绍 60](#_Toc534641206)

[5.2 网络环境的搭建 60](#_Toc534641207)

[5.2.1 拓扑环境搭建 61](#_Toc534641208)

[5.2.2 视频服务器搭建 63](#_Toc534641209)

[5.3 实验仿真与结果分析 64](#_Toc534641210)

[5.3.1 QoS路由计算有效性分析 64](#_Toc534641211)

[5.3.2 动态路由有效性分析 65](#_Toc534641212)

[5.3.3 队列调度的QoS控制性能测试 68](#_Toc534641213)

[5.3.4 QoS控制策略性能分析 70](#_Toc534641214)

[5.4 本章小结 71](#_Toc534641215)

[6 总结与展望 72](#_Toc534641216)

[6.1 全文工作总结 72](#_Toc534641217)

[6.2 不足与展望 72](#_Toc534641218)

[参考文献 75](#_Toc534641219)

[攻读硕士学位期间发表的论文 78](#_Toc534641220)

[致 谢 79](#_Toc534641221)

[学位论文知识产权声明 80](#_Toc534641222)

[学位论文独创性声明 81](#_Toc534641223)

# 1 绪 论

## 1.1 研究背景及意义

多媒体应用的快速普及使网络中视频应用与视频用户数量呈爆炸性增长，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]：截至2018年6月30日，网络视频用户规模达6.09亿，较去年年末增加3014万，占网民总体的76.0%，视频通信正成为多媒体通信中发展最为迅速、竞争最为激烈的领域。

互联网技术的快速发展促进了视频业务的蓬勃发展，用户对视频通信的需求也变得愈来愈强，视频点播、视频直播、视频会议、监控视频、可视电话等一系列流媒体视频应用正引起人们的广泛关注。如何通过网络更好地为用户提供视频服务，成为流媒体视频领域研究的主要方向。一般来讲，高质量的视频业务需要传输网络满足高带宽、低时延、小抖动、低丢包率等条件，但是网络资源是有限的，不可能提供无限的资源来满足不同的视频传输需求。目前，网络通过“尽力而为”的服务方式为某些视频业务提供了良好的服务，但是对于提供实时视频服务还存在着一定的挑战。尽管IETF针对此问题提出了多种QoS模型及机制，但是这些模型及机制都是建立在分布式互联网架构上，收集全局网络状态信息难度巨大，不能从根本上解决问题，无法有效地保证视频流媒体的服务质量（Quality of Service，QoS）[2]，当大量视频流媒体业务请求叠加时，传输路径容易拥堵，造成视频数据包丢失，严重影响视频的服务质量。

互联网中各种新型视频流媒体业务的不断涌现，使得应用对传统的网络架构提出了更高的要求，以现有视频流媒体业务的发展速度，传统的网络架构将难以适应未来视频业务的需求。针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，摒弃传统网络的弊端，提高网络数据传输的效率。软件定义网络（Software Defined Networking ，SDN）[3]作为一种新型的网络结构，为解决传统网络的很多问题提供了新的思路，SDN将网络的控制平面与数据转发平面解耦合，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合适的转发路径的决策，数据转发层根据收到的转发信息转发业务流而不做任何控制决策行为，这大大减少了网络的复杂度并提高了转发设备转发数据的速度。在SDN网络中，网络管理员只需根据特定的业务需求去修改控制层的配置，便可以轻易地改变转发层网络设备的行为，实现所需要的网络特性。对于视频传输而言，应用SDN架构可以让SDN控制器结合视频业务的接入及其网络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

为了解决TCP/IP架构所出现的问题，各国都进行了大规模的网络研究工作，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来进行研究。

从概念提出到校园研究再到真正的实现应用，SDN的发展历经了十几年。目前SDN的标准化工作由开放网络基金会（Open Networking Foundation，ONF）负责[6]，ONF已经发布和定制了多个版本的OpenFlow和SDN的白皮书。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SDN的应用、SDN测试实验以及控制平面的设计等。除此之外，国内外各大互联网公司也纷纷参与到SDN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]，其链路利用率达95%以上；微软通过在数据中心网络部署 software-driven WAN[12]把网络链路的利用率也提高到了99%；中国的腾讯公司也在其广域网中部署了基于SDN 的广域网流量调度方案，用来解决在网络运维中遇到的问题；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研究的浪潮里，如中国移动就在2017中国SDN/NFV大会上表示中国移动未来网络将在架构和网元两个层面进行重构，架构重构重在三层解耦的底层基础设施重构，网元重构重在以面向服务的架构来设计网元。许多厂商也参与到SDN交换机的设计与研发的工作中，如思科、IBM等厂商都推出了具备SDN功能的交换机及芯片；中国的华为在2013年发布了其首款支持 SDN 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]提出了一种针对流媒体的基于OpenFlow的可分级视频编码（SVC）流媒体传输方案，通过构建基础层和增强层2个独立路由，实现了动态网络下SVC流媒体分级自适应高效传输。

文献[14]通过区分QoS流，和非QoS流，对非QoS流直接选择最短路径，对QoS流运行其QoS路由算法，提出了一种新的基于OpenFlow网络的QoS算法。

文献[15]通过建立视频流分类模型，并将区分服务的思想应用到视频流的调度机制当中，以抢占式调度算法和PGPS调度算法为基础，对视频业务提供不同的QoS保障，满足 SDN 中多媒体视频业务的调度需求。

文献[16]利用SDN网络架构从控制器和交换机两个层面实现了基于SDN的业务服务质量保障系统，在全局网络层面为业务提供了QoS保障。

**2）国外研究现状**

K.-W. Kwong，R. Guerin 等人于 2007 年提出了对偶路由算法[17]，这种算法将数据流分成两类，每类对应不同的链路权重，后根据权重来求得路由。

Civanlar 等研究了通过OpenFlow网络进行视频流传输的QoS路由[18]。作者介绍了一种基于线性规划的方法为基于SVC编码的视频流计算QoS路由，旨在降低路径丢包率低同时将时延控制在要求范围内，同时计算流的QoS路由。他的基本实现是，对应 best-effort 业务流按照最短路径进行转发，视频业务按照提出的算法计算的QoS-rich路径进行转发。

Nikhil Handigol，Srini Seetharaman 等人于2009年提出了一种基于OpenFlow网络的路由方法**[**19-20]，对每一个数据流根据当前的网络负载情况计算出路由。

Egilmez 等人设计了一种基于OpenFlow的控制器OpenQoS[21]，用于视频流的QoS 保障。利用基于多媒体视频流的报文头字段与其它类型的数据流中报文头字段存在的差异，Open QoS 把针对于Open Flow 的匹配规则将传输的流量分为多媒体视频流和其他数据流两种类型。针对多媒体视频流，Open QoS结合传输路径在延迟、丢包测量得到的特性，在Open Flow控制器上增加了服务层，为其选择一条满足QoS参数的传输路径，其它数据流仍然维持在原有的最短路径上

Dobrijevic等人将QoS的概念扩展到QoE，通过QoE模型来表征路由的服务质量，基于蚁群算法为不同业务流寻找将其 QoE 最大化的路由[22]。通过对视频，语音和纯数据业务的路由实验与控制器默认的最短路径算法结果对比，验证了在一定网络规模内，该策略相比于最短路径算法路由体系的优越性。

Owens和 Durresi 介绍了另一种控制器架构和协议VSDN，用于SDN网络视频应用的QoS保障，允许视频应用向网络请求端到端的严格QoS保证服务[23]。这需要通过修改 Open Flow 提供的部分交换功能来实现，排队过程根据 VSDN 控制器提供的流量规范来调整每个流的流量。

### 1.2.3 存在的问题

整体上来看国内外对SDN网络的QoS研究未能充分发挥SDN网络架构的优势。具体说来当前研究存在以下问题：

（1）大多数的研究，没有在控制层算法上进行深挖，所采用的路由算法未能充分考虑网络的现状。

（2）大多数的学者对数据层和控制层进行分开研究，没有把他们结合起来，控制层在进行决策时没考虑到数据层的已有策略。

（3）大多数的研究，对网络中资源的分配是静态的，一旦分配好就不再改变，这种方法对网络资源的管理缺乏一定的灵活性，不能动态的适应网络变比如链路损坏，掉线等等，也没有充分发挥出SDN网络集中控制的优点。

## 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

随着互联网的飞速发展，流媒体类业务在网络中所占的比重正在急剧增大，它们对端到端的QoS有着严格的要求，然而传统数据报网络并不能很好的保证端到端的QoS。同时由于网络中业务种类繁多，其他业务类型的数据对QoS也有需求，且需求各不相同。针对这种情况，需要对业务分类并设置不同的优先级，然后依据类别及优先级对业务进行差别传输，以满足不同业务QoS需求。但是传统数据报网络在传输拥塞的情况下，只能提供尽力而为的服务，并没有区分对待不同种类的业务来保证不同的QoS需求。针对上述问题，本文充分利用SDN的优点，研究SDN中的队列调度、拥塞控制和QoS路由，提出并实现了区分业务优先级的QoS控制策略，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：

一是依据网络传输业务所需要的QoS要求高低，对业务进行了优先级的区分，并设定了不同的优先级值。

二是从控制层和数据层制定整套视频流媒体的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，而对其它等级业务使用Dijstra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进行区分调度，保障了视频流媒体优先传输的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。

三是在SDN网络中，对QoS控制框架的功能模块进行了实现。一方面在控制器编写程序实现了基于遗传算法的QoS路由，较好的保障了视频流媒体的QoS。另一方面，在OpenFlow交换机上进行队列的配置，通过控制器下发指令将数据包送入配置好的队列中，实现了不同等级业务数据包的差别调度，在转发端口进一步保证视频流媒体的QoS。

四是搭建软件定义网络模拟实验平台，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，从控制层和转发层分别制定控制策略，旨在满足视频流媒体业务端到端的QoS需求，同时尽最大可能满足其他类型业务QoS需求。围绕着论文的主要工作，本文对内容结构做如下安排：

第一章，绪论。介绍本课题的研究背景和意义以及国内外研究现状，阐述本文的主要研究内容。

第二章，SDN中QoS的相关技术分析。对QoS以及SDN网络进行了概述，以及对传统网络下流媒体传输遇到的问题进行了分析总结。

第三章，QoS控制策略分析与设计。针对视频流媒体传输的特点，提出了基于业务优先级的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设计。

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。搭建网络模拟环境及视频传输平台，对QoS控制策略框架进行测试。

第六章，总结与展望。总结了此次论文所做的工作，指出需要改进的地方并对未来工作做出展望。

# 2 SDN中QoS的相关技术分析

## 2.1 QoS技术

### 2.1.1 QoS的定义

服务质量（Quality of Service，QoS）是指网络利用各种技术保障网络延时和吞吐量等性能的服务要求，向选定的网络通信提供更好服务的一种安全机制[24]。在通信中，QoS 主要通过各种性能参数体现，其中常见的几种QoS性能指标有吞吐量﹑丢包率﹑时延 [25]。

吞吐量：指数据的传输速度，即单位时间内成功传输的数据大小，常用单位 M/s。

丢包率：假如发送方一共发了n个数据包，接收方一共收到m个数据包，则丢包率为（n-m）/n。网络丢包一方面会造成用户体验下降，另一方面丢包所引起的丢包重传会造成网络负荷进一步加大。

时延：指发送方发送数据到接收方收到数据用的时间，常用单位 ms，时延包括发送时延，传输时延，处理时延，排队时延。

### 2.1.2 QoS服务模型及机制

QoS管理从模型及机制上来说可分为尽力而为服务模型（Best-Effort）、综合服务模型（IntServ）、区分服务模型（DiffServ）、多协议标签交换（MPLS）以及QoS路由。

尽力而为服务模型，不能提供任何QoS的保证，对数据包采取先来先服务策略，当网络发生拥塞时，采取尾丢弃等方式丢弃数据包。

综合服务模型（IntServ）[26]通过资源预留协议（RSVP）为业务预留带宽、缓存等资源，保证业务实现端到端的QoS保障。但由于要为业务预留资源，整个网络中的每台设备都要能实现此功能，扩展性较差。IntServ 更换路由时需要断开连接、重新建立连接等一系列操作，将会增加额外的时延，因此难以应用动态QoS路由。

区分服务（DiffServ）[27-28]模型将传输的业务流在进入网络时进行分类和标记，将不同的业务流映射到相应的QoS等级上，后续的网络设备根据已有的QoS标记提供对应的QoS服务。边缘网络设备必须要具备报文分类、报文标记、流量整形等功能，而核心网络设备则只需要根据QoS优先级进行相应类别的转发功能，解决了IntServ 扩展性差的问题，但是由于DiffServ 架构中缺乏端到端系统间的协议通信，不能提供端到端的QoS保障。

多协议标签交换（MPLS）[29]采用分类标记的方法选择传输数据包的路由，基于标签交换技术进行数据转发，而不是基于目的IP地址查找路由表。MPLS分离数据转发与控制，能够有效地支持网络系统升级。MPLS最主要的功能是流量工程，即多条候选路径中进行链路负载平衡。

QoS路由的主要目标是为接入的业务选择满足服务质量（如带宽、时延、丢包等）要求的传输路径，同时保证整个网络资源的有效利用。

## 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网络带来更沉重的负担，为了解决这个问题，在传统网络中加入更多的协议及功能，使得网络更加复杂，这样的恶性循环使得网络会有崩塌的危险；二是处理媒体数据要比普通数据产生更多的开销，除了要进行编解码，客户端播放，还要运行网络状态检测机制和拥塞控制机制，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，尽管目前已经提出了许IP 层的QoS模型和机制，在某个域内或某条特定的网络路径上实现网络层服务质量是完全可行的，但是一方面由于这些方案的域间互通性较差，另一方面由于向IP 路由器引入了较多复杂功能，路由设备的软硬件更新换代的成本过高，在整个Internet 范围内实现任意端节点之间的QoS保证的目标仍然遥不可及。

若流媒体视频流在网络中传输时不满足最低的QoS要求，会出现以下情况：若有视频数据帧延迟，可能造成同一幅画面上各点的时间不同步，视觉上看，物体都是变形的。视频流在传输时，若产生数据丢失，导致同一幅画面各个区域视觉效果不同，有的地方清晰，有的趋于清晰，有的区域模糊。视频传输时产生的延迟比较大，使得画面抖动比较厉害，效果很差。

## 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，新的网络架构中具有代表性的有内容中心网络（Content-Centric Networking CCN）[30]和SDN网络。CCN网络请求数据时直接以内容的名字进行请求而不再是通过IP，网络中有很多节点对内容进行缓存，而SDN 网络则是对网络的控制层和数据层进行分离。相比之下CCN对网络的改变更大一些，替换了网络层，而SDN对网络的变革要小一些，SDN也更容易实现一些，本文以 SDN 网络为研究基础。

OpenFlow概念源于斯坦福大学的Clean Slate计划，由该项目的Nick McKeown教授提出，最初是作为校园的实验平台产生的，之后SDN的概念应运而生，Mckeown 等设计者们开始推广SDN概念[31]**，**并引起业内外广泛关注。

|  |
| --- |
|  |
| 图2.1 传统分布式结构与SDN架构对比图 |

如图2.1所示，SDN集中控制的网络架构与传统网络架构不同，其控制层与数据转发层是分开的。当前SDN的架构遵循于开放网络基金会（Open Network Foundation，ONF）提出的架构，由基础设施层、控制层、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，还可以是软件交换机，如OpenvSwitch。应用层根据不同的业务需求，调用控制层的北向接口，用软件的方式灵活地对基础设施层中的网络设备进行管理和控制。南向接口用于控制平面与数据平面间信息的交互，通过该接口，控制器可以获取数据平面的运行信息，并对数据平面的设备进行配置。北向接口用于控制器与上层应用程序的交互，对网络进行了抽象，这样网络管理者可以不用关心底层网络的细节，通过该接口，进行简单的编程就可以控制网络的运行逻辑，从而方便地管理网络。

SDN的体系架构具有两个特征：控制平面与数据转发平面解耦合，控制平面的易编程性。转控分离的思想使转发设备从繁重的工作中解脱出来，专注于数据的传输，相应的控制决策行为交给控制器来完成。同时SDN出现伊始就被定义为开放性的架构平台，将自由可编程引入控制平面，为简化网络配置、提高网络的性能、鼓励网络应用创新带来了巨大的便利性。

### 2.3.1 SDN控制器

SDN控制器是SDN网络架构的中枢，负责监听、维护和管理网络，并为上层应用提供各类统一的编程接口，通过各类应用来实现网络的动态管理和配置。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，目前已有超过20种控制器的实现，一些常见的控制器如表2.1所示[33]。

表2.1 常见的软件定义网络控制器

|  |  |  |
| --- | --- | --- |
| 控制器名称 | 实现语言 | 简介 |
| NOX | C++/Python | 第一款OpenFlow控制器 |
| POX | Python | NOX的纯Python实现 |
| Beacon | Java | 扩平台的模块化设计，易于部署开发 |
| FloodLight | Java | 企业级控制器，脱胎于Beacon |
| Mul | C | 使用C的多线程框架，提供多层次北向接口 |
| Ryu | Python | 能与OpenStack平台整合，易于开发 |
| Meastro | Java | 跨平台，易部署。支持多线程 |
| NodeFlow | Javascript | 极度简化的OpenFlow控制器，用于编写可扩展的因特网应用 |

Ryu是由日本NTT公司贡献的基于组件的开源SDN框架，组件是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产生；提供GUI界面用于管理和监控网络拓扑和状态，模块清晰，可扩展性好；采用python语言开发，python语法简洁清晰，易于学习阅读；跨平台，并提供大量的第三方库，善于做科学计算；支持多种用以管理网络设备的协议，如OpenFlow、Netconf、OF-config等；支持OpenFlow从1.0到1.5以及Nicira 的全部版本[34]，本文所实现的QoS路由功能就是在Ryu中编写相应模块实现的。

### 2.3.2 OpenFlow

目前，最为主流的南向接口为OpenFlow协议，OpenFlow协议对SDN数据平面中的转发设备（OpenFlow交换机）的基本组件与功能要求进行了定义，同时还规范了SDN控制器与OpenFlow交换机间控制的消息。

**1）OpenFlow交换机**

一个OpenFlow交换机通常由OpenFlow通道以及一个或多个流表、组表组成。OpenFlow安全通道是交换机和控制器的连接接口，控制器与交换机之间是通过安全通道进行通信的，所有的信息必须按照协议规定的格式来执行，根据OpenFlow协议的要求，采用TCP连接以保证可靠性，通过安全传输层协议（TLS，Transpor Layer Security）加密以保证安全性。流表及组表是数据转发的指引者，当数据进入到OpenFlow交换机，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的流表与传统网络设备中的二层MAC地址转发表或者三层IP地址路由表类似，不过OpenFlow在流表项的匹配域中对网络中各个层次的网络配置属性进行了整合，使得网络设备能够更加精确的对网络中的流进行匹配，可以可以对数据流执行更细粒度的操作。交换机中的流表由多个流表项组成，每个流表项都包含匹配域、优先级、计数器等一系列指令，OpenFlow1.3规定的流表项的结构如图2.4所示。

|  |
| --- |
|  |
| 图2.4 流表项结构图 |

匹配域是用来判断数据包与流表项是否匹配，从图2.4可知由16个字段组成，涵盖了1-4层的网络信息，其中每个元组可以是匹配精确值，可以是匹配“ANY”任意值，或者利用掩码匹配某一范围的值。优先级定义了相应流表的匹配顺序，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，比如某端口的接收与发送字节数或某个流的匹配数据包数等信息。

超时主要用于流表项的超时，包括idle timeout和hard timeout，分别表示流表项的活跃时间和硬超时时间。Cookie是由控制器设置的数据值，控制器用这个值来进行过滤流统计数据，流改变和流删除，但交换机并不用此数据来处理数据包。

指令表示匹配的数据包该执行的动作，指令主要有转发（Forward）、丢弃（Drop）、排队（Enqueue）、修改域等指令，一个流表项可以有零个到多个指令，如果有多个指令则可以规定每个指令的执行顺序，如果在这些指令里不含转发，则和这个流表项匹配的数据流就会被丢弃。

转发的七种行为分别是LOCAL，转发给当地的网络协议栈；IN\_PORT，从哪个端口进来，就转发给哪个端口；TABLE，对packet\_out消息执行流表的动作；CONTROLLER，将数据包封装并转发给OpenFlow控制器；ALL，转发给除入口端的所有出端口；FLOOD，遵照最小生成树从设备出端口洪泛发出，但不含入端口；NORMAL，利用传统转发机制如MAC地址和IP 等信息转发数据包。

丢弃（Drop）：若数据包没有和任何流表项匹配，且没有设置 table-missing 项或table-missing 项的动作是丢弃，则会被丢弃。

排队（Enqueue）：设置数据包的队列ID，将数据包放到出端口的对应转发队列，此项操作与QoS相关。

修改域（Modify-Field）可选的字段修改操作有修改IP包头的ToS（服务类型）字段，把新的值写入原ToS字段，该操作可用于支持QoS服务（ToS 的前六位为 DSCP，后两位目前保留）；修改源目的IP地址，把其用指定的值替换，并对IP、TCP或UDP的校验和重新计算；修改TCP或UDP的端口号，既可以修改源端口号，也可以修改目的端口号，修改后对TCP或UDP的校验和重新计算；修改以太网MAC地址，源地址、目的地址皆可；设置VLAN优先级，若被设置的数据包以前没有VLAN首部，则会被加上一个新的首部，设置其优先级，并将VLAN ID 置 0，若数据包头已有VLAN首部，则只需将旧的值替换即可；去除VLAN首部，若数据包中存在VLAN首部，则将其去除。

OpenFlow交换机中，对数据的所有操作都通过流表来定义，交换机只是简单的根据流表处理，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交换机处理数据包的过程如图2.5所示，当一个数据包进入OpenFlow交换机后，交换机首先在编号为0的流表里进行匹配，若流表里存在能匹配该数据包的流表项，则更新流表项的计数器，按照该流表项的指令转发数据，否则数据包转到下个流表项进行匹配，若数据包在最后一个流表里都没有匹配成功，则丢弃该数据包或者携带数据信息转发到控制器进行相应处理。

|  |
| --- |
|  |
| 图2.5 流表匹配流程图 |

**2）OpenFlow协议**

OpenFlow协议运行在OpenFlow交换机中，是交换机与控制器通信需要遵循的标准。OpenFlow交换机和控制器间之间的信息交互依靠OpenFlow协议中的消息，OpenFlow协议中的消息包括控制器-交换机（Controller-Switch）消息、异步消息（Asynchronous）以及同步消息（Symmetric）。

控制器-交换机消息用于控制器对交换机的状态进行查询，控制器通过发送Request请求，分析交换机返回的Reply消息来对交换机的状态进行查询。例如features-request消息可以获得交换机支持的性质；modify-state 消息可以管理交换机端口状态（port-mod）和流表项（flow-mod）；利用read-state消息向交换机发出请求，获取各个流表、流表项、端口的统计信息；利用send-packet消息，通过packet-out消息来响应packet-in消息，控制交换机向指定端口发送数据包。

异步消息由交换机主动发起，向控制器告知网络事件及其状态变化，异步消息包括packet-in、flow-removed以及ports-status等。当进到交换机中的数据没有找到匹配的流表进行转发时，就会触发packet-in消息，其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，作出相应的处置决定。当交换机中的流表项因超时或被修改而被删除时，会触发flow-removed消息，告诉控制器该流表项已不存在。当端口状态变化，例如从up变为down时，交换机会发送ports-status消息，控制器将重新整合底层网络拓扑状态信息。

同步消息可以由控制器和交换机任意一方发起，主要用于建立和保持连接。例如通过hello消息确认连接，通过echo消息确认连接状态或测量数据传输时延。

### 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，其可以将复杂的QoS控制逻辑在控制平面实现，进行全局性的调度。

OpenFlow1.0有一个可选动作enqueue（1.1版本后叫set queue），该动作可以将数据包送入一指定的交换机队列中，与传统网络最大的不同是，SDN架构具有实时更新配置的特点，因此可以利用该动作对QoS进行更加灵活的控制。

在OpenFlow1.1中，添加了VLAN，MPLS标签以及流量类型字段的匹配与操作, 在该版本之后，能够通过OpenFlow协议对VLAN标签进行添加、修改与去除操作，并且添加了多级VLAN的支持。

OpenFlow1.2协议中添加一个消息，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Meter表项组成，结构如图2.6所示。

Meter band的Type Specific Arguments可以是drop操作或DSCP remark操作。即当一个Meter的速率超过预先设定的Rate时，Meter即会采取相应的策略，对于超过设定值的这一部分数据包丢弃或者对DSCP重标记（优先级降低惩罚）。Meter与队列机制结合使用，可以实现非常复杂的QoS控制。

|  |
| --- |
|  |
| 图2.6 Meter表项组成图 |

Open Flow 1.4：该版本提出了流量监控框架，允许控制器实时监控其他控制器对流表的任何子集所做的更改。

Open Flow 1.5：将Meter 指令改为Meter动作。这个改动使得多个meter可以属于一条流表项以及被使用到组桶中。

软件定义网络架构在QoS路由上相对于传统网络具有明显的优势：

1. 统筹决策。能够根据网络的全局视图计算路由，为业务流分配满足QoS的路由。

2. 时效性。实时地对网络监测的能力，使得路由策略的计算都是在最新的数据下进行的，具有比较好的实效性。

3. 细粒度。能够精确控制网络中的每一条数据流，及时控制数据流的转发，能够很好的应对网络出现的问题，比如：链路失效、链路拥塞。

### 2.3.4 智能算法在SDN上运行的可行性分析

在SDN中，采用启发式智能算法解决网络中的流量问题是一种可行并且有效的选择。对于路径选择这种NP问题，一般的智能算法都可以进行优化解决，虽然每种智能算法进行优化计算的方式不同，但是都需要收集部分或者整个系统的信息，通过计算出最优解再进行资源的重新部署安排。因此这些算法都需要能够持续地收集到网络全局的实时信息，并且算法部署实施的代价比较小。而SDN的特性能够比较好的满足以上的要求，使得智能算法在SDN网络中运行成为可行的。

相对于传统网络，SDN提供了控制器对网络的集中控制，这里主要指收集全局信息这一点上，SDN也有分布式控制模型。控制器通过专用的控制链路连接交换机，可以收集全局的状态信息，并且可以简单高效地给底层交换机部署流表指令。

SDN控制器收集到的实时性信息十分丰富，因为SDN是基于流分析的，OpenFlow协议所规范的十六元组基本上囊括了数据流的基本信息，通过不同策略的流分析，能够得到当前流的详细信息，进而对流进行不同的处理。控制器还可以收集整个系统运行时的状态信息，这些信息为智能算法的计算提供了很好的参考数据，使算法做出的优化选择更加智能。并且，通过OpenFlow协议，控制器对流的操作控制可以进行多粒度处理，为算法提供的可操作项也更多，例如控制器检测到传输路径出现拥塞时，能够根据周期获得的网络信息为数据流提供动态路由的机制。

SDN提供了开放的可编程接口，开发者可以利用这些接口编写扩展出自定义的应用或者模块，例如可以在控制器内增加额外的链路性能测量模块，获取网络信息；编写部署客制化的应用，将这些策略或优化结果部署到网络中去。

## 2.4 本章小结

本章首先对QoS技术进行了分析，主要是QoS概念以及QoS服务模型及机制的介绍。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网络的运行机制进行阐述以及对智能算法在SDN上运行的可行性进行了分析。下一章将以本章内容为基础展开，针对传统网络中视频流媒体传输面临的困境，借助SDN网络的优势，提高其传输性能。

# 

# 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

根据SDN的三层架构体系，任何应用都是以SDN控制平台为中心并在其上扩展相关模块而实现的，SDN控制器的性能将直接影响应用的实现效果，因此需要对控制器环境及它能为上层应用提供的编程能力进行研究分析。SDN控制器是上层应用和下层网络设备之间通信的桥梁，向上，它向应用提供数据平面的基本信息，包括网络拓扑、设备资源使用情况和链路状态等；向下，它将应用发出的指令发送给数据平面，对OpenFlow交换机的流表进行编程。

本文的QoS路由功能属于上层应用，需要部署在控制器中，为了实现这些应用，不仅需要调用控制器的相关模块实现一些基本功能，如基本的模块管理、协议适配、事件机制等模块；还需要在控制器上扩展相应的模块来实现本文所需要的功能，比如拓扑探测、链路性能信息获取等模块；同时，也需要注意自定义模块与控制器原有模块之间的调用顺序以及它们的协同工作。队列调度功能属于底层应用，摈弃其固有的先进先出的队列模式，采用新的队列算法，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，也能兼顾其他业务等级流的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，同时兼顾其他等级数据流的传输。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，对于其他等级的业务流则使用以路径跳数为代价的Dijkstra算法进行路由，在数据路传输过程中，实时监测数据流传输的前方路径的链路状态，当监测到传输路径前方发生拥塞时，为视频流进行重路由。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，优先调度视频流媒体，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、重路由路径的选择、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）、计算网络拓扑：策略的正确部署依赖于全网拓扑视图，因此需要动态获取全网拓扑，当有传输节点或链路加入或退出时，及时更新网络拓扑结构，以保障在路径选择时有实时的拓扑参考。

2）、收集链路信息：本文方案中QoS度量参数包括链路时延、链路丢包率，为了使策略能够选取出正确的路径，需要设计相应的模块实时地监控并获取这些资源的使用情况。

3）、保证视频流的服务质量：为了满足视频流的服务质量，需要对数据流流进行划分，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）、自动化：在数据流传输过程中，利用SDN控制器的集中管控能力实时监测链路并进行评估，当发生拥塞时，自动根据全网拓扑、流量分布情况调用动态路由策略选取替代路径，并下发流表。

5）、快速恢复：网络拥塞会增大数据包的传输时延，影响数据端到端的传输，所以动态策略必须要及时发现拥塞并快速恢复链路畅通。当发生拥塞时，要使下发新流表的时延尽量小，同时在下发流表期间，也要保证一定量的数据传输，避免流表下发不一致带来的“流中断”。

## 3.2 QoS控制策略框架

### 3.2.1 数据流分类

实时视频会议、高清晰在线视频、视频监控等流媒体业务对时延和丢包率等性能参数有着严格的要求，为满足这类业务的QoS，并充分利用网络资源，需要对所有业务流进行优先级的分类，通过有区别的业务路由和业务数据包队列调度转发，优先满足此类业务的QoS。

数据流分类的方法有很多，如应用层的深度净荷检测法、传输层的端口分析法等。本文为了能对不同种类业务数据包标记对应的等级值，利用DiffServ网络模型中的DS域[35]来进行业务等级值的存储。DiffServ网络模型将IP数据包头中的8位服务类型字段（ToS）重新定义为区分服务字段DS，用来区分不同业务数据流，表示不同的优先级。虽然ToS字段有8位，但目前只用到了前6位记为 DSCP，从理论上说DSCP值不同则服务优先级不同，这个值越大对应的服务优先级也越高。

根据IETF定义的服务类别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需求相对较低的应用。

表3.1 应用的QoS等级分类及其对应的DSCP值

|  |  |  |  |
| --- | --- | --- | --- |
| 应用类型 | 协议举例 | QoS服务等级 | DSCP值 |
| 实时视频会议、视频监控 | RTP,RTSP | Q1 | 100000（32） |
| 关键数据业务 | NFC,SMB,RPC | Q2 | 011000（24） |
| 事物处理，交互式数据 | SQL | Q3 | 010000（16） |
| 数据同步，大块数据，e-mail | FTP,SMTP | Q4 | 001000（8） |

对流的分类一般的做法是根据包头中的源IP地址、目的IP地址、协议字段、源端口号、目的端口号等字段，并根据分类结果重新标记它的DSCP值。用这种方法能区分大多数的业务，如果想区分的更细可以做包解析，但复杂度较高，故一般采用5元组，或更多维的包头数据直接匹配，本文使用的也是这个方法来对流进行分类。

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（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，而是依靠控制器的流表项中的修改域指令，该指令可以修改IP包头的ToS字段中的DSCP值，把新的值写入原DSCP。在控制器下发流表时，规定匹配该流表的数据流的DSCP值作出相应的修改。此动作需在边缘结点中执行，确保从边缘网络进入的数据流都被打上DSCP标签，网络其他结点则直接根据IP数据包头的 DSCP值进行相应的流分类和标记操作。

### 3.2.2 QoS控制策略的架构设计

由上节介绍可知，本文依据业务类型，将数据流分为四个等级，视频流媒体业务对时延、丢包率等都有着严格的要求，其他等级业务则对QoS的要求较低。为了在多种数据流并存的网络中满足视频流媒体的QoS，并减少其对其他等级业务流的影响，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，提出了视频流媒体的QoS控制策略的框架。该QoS控制策略架构能够满足视频流媒体端到端的服务，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。它提高了网络资源的利用率，细化了业务流控制的粒度。

QoS控制架构如图3.1所示，主要由QoS路由与队列调度两个部分组成，这两个部分在SDN控制器与OpenFlow交换机都有着对应的模块，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控制框架执行流程，当数据流进入OpenFlow交换机时，若没有可匹配的流表，就会触发packet-in消息携带数据包信息进入到控制器，控制器通过解析数据包的头部信息，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值以及将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控制框架执行流程 |

## 3.3 QoS控制框架模块分析设计

QoS控制框架分为QoS路由和队列调度两个部分，下面分别对这两部分的设计进行阐述。

### 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、路由管理模块。QoS路由框架的设计是在控制器原有功能的基础上，充分利用了SDN网络中控制平面和数据平面可以通过OpenFlow协议进行自由灵活交互的特点。

**1）拓扑管理模块**

控制器中的拓扑管理模块用于主动对网络拓扑进行探测，以获取网络中OpenFlow交换机的MAC地址，端口号等信息，并能够周期性地获取网络设备间的链路连接关系，主要是数据平面中交换机与交换机之间的链路，还包括交换机与主机的链路以及控制器和交换机之间的连接情况。同时会对网络链路的变化进行监测，保证模块维护的全网络拓扑连接信息是最新的。该模块所得到的网络拓扑连接结构给控制器提供了一个全局的网络视野，使得控制器能根据全网拓扑进行路由。

另一方面，模块会从链路性能测量模块获取链路的性能信息，包括时延、丢包率、带宽等信息。得到当前网络链路的物理连接信息和链路对应的性能参数后，根据这些信息拓扑管理模块对链路进行抽象封装，因为路由问题原型是图论中的寻路问题，这里将网络全局信息映射为一个邻接链表形式表达的带权图，从而方便后续QoS路由算法进行数据处理。网络中物理的交换机映射为图中的顶点，交换机的连接关系映射为图中顶点之间连接的边，链路的性能参数映射为图中边不同的权值。通过这样的方式完成了对网络物理状态的抽象映射，形成抽象的详细网络全局资源视图，该视图是QoS路由计算模块的数据支撑。

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；信息采集是对交换机信息进行统计（交换机 datapath ID，端口号 port，交换机间的链路连接）；信息分析是根据信息采集模块统计的相关信息，进行运算处理，计算网络的拓扑图；状态轮询模块利用子线程执行拓扑信息的循环查询，当网络拓扑发生变化时，例如交换机接入、离开，链路状态变化时，对相应的拓扑信息进行更新；拓扑图显示模块将对交换机邻接矩阵进行遍历，向用户展示网络的拓扑信息。

|  |
| --- |
|  |
| 图3.3 拓扑发现模块图 |

SDN网络中链路信息采集的工作由控制器统一完成。OFDP（OpenFlow Discovery Protocol）利用链路层发现协议（Link Layer Discovery Protocol，LLDP）进行链路间信息的采集。LLDP的以太网帧如图3.4所示。

|  |
| --- |
|  |
| 图3.4 LLDP以太网帧 |

其中LLDPDU为有效载荷，该部分以TLV（Tag Length Value）集合体的形式将设备标识、接口标识以及地址等相关信息进行存储。LLDP将这些信息转送给网络中相邻的网络设备，接收到LLDP帧的设备能够获取存储在LLDPDU中的相应信息并将其保存。

控制器执行信息采集的过程如图3.5所示,控制器为了获得网络拓扑连接情况，将LLDP数据包封装在packet-out消息中发送给s1和s2。以s1为例，packet-out中下发的流表规则：s1将从控制器端口收到的LLDP数据包从非接收端口发送出去；s2的port1与s1的port1端口相连，因此LLDP数据包会从s1的port1端口发送出去，s2的port1端口收到此数据包；交换机内没有专门匹配LLDP数据包的流表，因此会触发packet-in消息携带LLDP数据包信息发送给控制器；控制器收到packet-in消息后对LLDP数据包进行解析，解析报文中携带的classis ID和port ID，确定发送该LLDP报文的为s1的port1，解析packet-in消息中携带的in-port，定收到该LLDP报文为s2的port1，从而确定交换机s1与s2间的链路连接情况，即s2的port1与s1相连，同理可知s1的port2与s2相连。依照这样的方式控制器获得网络中相连的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**

带宽、时延、丢包率等链路性能参数代表了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，然后通过这些链路组合出满足业务流QoS需求的转发路径。随着OpenFlow协议的不断更新，协议功能逐渐丰富，利用OpenFlow协议可以直接获取少量网络状态信息，但是链路的时延、带宽、丢包率等链路性能信息并不能由OpenFlow协议直接获得，因此本文对控制器功能进行扩展，添加了链路性能信息测量模快，从而高效地对链路的QoS性能参数进行测量。下文将对链路性能信息的测量原理进行具体阐述。

**a、带宽使用情况及丢包率**

OpenFlow协议中对于交换机计数器统计的字段有相应的定义及要求，其中对于交换机端口能够查询到统计信息如图3.6所示。

|  |
| --- |
|  |
| 图3.6 端口消息内部结构图 |

通过该消息可以获得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，通过它们能够计算出一条链路的带宽使用情况及丢包率。

假设系统分别在t1时刻与t2时刻根据OFP\_Port\_Stats消息获取了交换机s1与交换机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端口，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，单位为bit/s。

|  |  |
| --- | --- |
|  | (3.1) |
|  | (3.2) |
|  | (3.3) |

则交换机s1的端口port1被占用的带宽及链路剩余带宽如公式(3.4)和(3.5)所示。

|  |  |
| --- | --- |
|  | (3.4) |
|  | (3.5) |

由于交换机s1的port1与交换机s2的port2对于传输数据包的统计是对称，即除去链路出现故障等意外状况，位于端口port1的统计字段应当与位于端口port2的统计字段数据相同。因此，位于链路两端的两个交换机的端口，其中一个交换机的发送数据包数即为另一个端口的接收数据包数。

链路拥塞率反应了网络业务数据流传输的拥塞程度，拥塞率越大，则拥塞越严重。本文使用公式(3.6)来计算链路的拥塞率，其中为单位换算即bits/s-Mbit/s。

|  |  |
| --- | --- |
|  | (3.6) |

OpenFlow对于一个流表项定义了类似端口统计字段的统计消息，该消息为OFP\_Flow\_Stats，OFP\_Flow\_Stats能够返回一个交换机中某个流表项对于匹配到该流表项的流的相应数据的统计信息，其中同样定义了duration\_sec以及duration\_nsec。由于对于一条流而言，只存在一个方向，因此对于数据包的统计字段只有packet\_count与byte\_count字段，分别表示当前流表项匹配到了多少报文与字节数，此外也可以根据该消息的instructions字段，获知匹配到该条流表项的数据包会被采取何种动作，比如丢弃或者送至某一个端口。

对于一条链路，丢包率的测量与带宽使用情况的测量方法类似。但对于链路丢包率的测量除了使用OFP\_Port\_Stats消息，还需要使用OFP\_Flow\_Stats消息中的instructions字段以及packet\_count字段，通过OFP\_Flow\_Stats消息能够获得OpenFlow交换机中所有流表项的统计数据，并根据instructions字段提供的输出端口，从而能够得到送往当前交换机其中某一个端口的报文总数。对于交换机s1与s2间的链路（s1的port1与s2的port2相连），根据OFP\_Flow\_Stats消息，t1时刻交换机s1所有流表项统计的发送到port1的报文总数为，同理t2时刻为，t1时刻到t2时刻，交换机s1传给交换机2的报文数如公式(3.7)所示。

|  |  |
| --- | --- |
|  | (3.7) |

根据OFP\_Port\_Stats消息中的字段获得t1时刻交换机s2从port2端口接收到的报文总数，同理，t2时刻接收的报文总数，t1时刻到t2时刻，交换机s2实际接收的报文数如公式(3.8)所示。

|  |  |
| --- | --- |
|  | (3.8) |

丢包率的计算如公式(3.9)所示。

|  |  |
| --- | --- |
|  | (3.9) |

**b、时延**

OpenFlow协议本身并未提供直接用于时延测量的机制，但通过利用OpenFlow协议中定义的消息，使消息中携带时间戳的方式，可以对网络中的时延进行测量。本文利用OpenFlow协议中的packet-out消息、packet-in消息、Echo Request消息和Echo Reply消息之间相关的联系计算链路的时延。

时延测量的具体步骤如下：

（1）控制器构造packet-out消息并携带LLDP数据包发送到所有交换机，使消息的data域携带将消息送出控制器的时间戳，该消息指示交换机将LLDP数据包发送到其他端口，交换机s2收到s1发送过来的LLDP数据包，交换机内并没有处理这种数据包的规则，以packet-in消息上报到控制器，记录此时的时间，即为控制器到s1交换机，s1到s2，s2到控制器的大致时延。同理，控制器到交换机s2，s2到s1，s1到控制器的时延T2计算方式也如此。

（2）控制器到交换机的往返时延测量是利用请求回复消息Echo Request，使消息的data域携带将消息送出控制器的时间戳，控制器向交换机s1发送Echo Request消息，交换机收到Echo Request消息会以Echo Reply消息回复控制器，控制器收到Echo Reply报文后会记录获取报文的时间，因此控制器与s1交换机往返时间，同理，控制器与s2交换机之间往返的时间计算方式也是如此。

（3）交换机s1与s2之间时延计算公式如(3.10)所示。

|  |  |  |
| --- | --- | --- |
|  | (3.10) | |
|  | |
| 图3.7 时延测量原理图 | |

**3）路由管理模块**

路由管理模块负责网络的路径决策，是整个QoS路由控制机制中最核心的部分。路由管理模块有四个功能，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；路径的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，对视频流媒体数据流进行QoS路由，对其他低优先级业务数据流使用以跳数为路径代价的最短路径算法进行路由。通过这种方式，实现了不同优先级业务的区分路由，在控制层较好的保证了视频流媒体数据流端到端的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约束的视频流媒体的路由，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Dijstra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]，有些研究通过将多种约束参数用一个混合的参数来度量，通常合成这种参数的相关约束之间的运算不易确定，而且这种混合参数算出的路径可能不能反应其中任何一个约束，并没有足够的实际意义。通常两个以上加性参数的多约束路由问题需要使用近似算法或启发式算法求解，近似算法与启发式算法各有各的优缺点，近似算法能够更好的确保解的质量，但计算却相对缓慢；启发式算法能够较快的计算出最优解，却无法确保解的质量，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。

遗传算法是启发式算法中较为经典的算法，通过模拟生物在自然环境中的遗传和进化机制而提出的一种全局优化概率搜索算法[40-41]，遗传算法是一种迭代方法，通常它会从一组群体开始，模拟生物的进化过程，通过交叉、变异等遗传操作生成下一组群体，通过适应度来调整群体的进化方向，不断迭代，最终找到最优解或者合适的解。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，动态路由的功能是能够根据周期检测的情况，在发现视频流媒体的传输路径出现拥塞时，重新计算符合需求的路径，然后下发新路径的流表，删除交换机中旧路径的流表。本文假定理想情况为触发动态路由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，端口转发一直会使用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。

**（1）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%，即路径中的链路带宽的使用率为固定带宽的80%以上，即判定为拥塞，需要用依据遗传算法为视频流重路由，为其重新选择一条合适的路径进行传输。

**（2）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，需要下发新的流表，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，却涉及新老流表项共存和路径切换过程中丢包的问题，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。

路径切换机制中涉及两个问题：一是新流表项安装的问题，二是如何处理新旧流表共存的问题。需要注意的是，在OpenFlow交换机的流表中，流表项与流表项的区分是以匹配域以及优先级作为依据的，即一个流表项的身份标识为匹配域与优先级。在OpenFlow协议中，优先级的值越大则优先级越高。如果新路径的流表项在安装进交换机之后，其优先级低于老路径的流表项，则数据包进入网络之后，仍将优先匹配到原有的流表项，最终仍然按照原有的传输路径进行数据包的传输。若新的流表项优先级与原有的流表项的优先级相同，则对于数据包的传输会存在一定的风险。而将优先级的设为高于原有的流表项，则新的数据包会优先匹配新路径的流表项，从而新进入网络的数据包会按照新路径进行传输。因此新的流表项的优先级必是要高于旧的流表项。

确定新的流表项的优先级高于旧的流表项后，需要决定流表的安装顺序。如果按“源交换机”到“目的交换机”的顺序依次下发流表，则很有可能产生“流中断”现象，即当数据流到达新路径的某一交换机时，流表更新还没有完成，此时数据包不能匹配到任一流表项，停在了此处，“流中断”不但耽误了数据流的传输时间，还极有可能产生丢包现象，影响端到端的传输。

基于此，采用“倒序式”安装流表，即按照从“目的交换机”到“源交换机”的顺序依次写入流表。在重写全部完成之前，不删除原有路径的流表项，这样在调整过程中，数据流仍继续按照原有路径传输，避免了路径切换不一致情况下造成的数据丢失，尽量保证了流动态调整中数据的连续性。

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，触动重路由机制为流重新选择传输路径为s1-s3-s2，若按照“顺序式”写入流表，T1时刻s1上的流表已经重新成功，新进入到网络中的流依据s1中新的流表项进行匹配转发，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，就会造成数据流停在此处，根据OpenFlow交换机的设置，此时有两种动作，一种是丢弃数据包，一种是送至控制器。对于第一种情况将直接造成丢包，第二种情况，会出现短时间大量packet-in消息送至控制器的状况，对系统的运行产生极大的负载，甚至影响正常工作。因此不论哪一种情况，都不是本文希望发生的。

若采用“倒序式”重写流表，即按照s2到s3再到s1的顺序重写流表，在最后一个交换机s1重写成功之前，数据流仍按照原始的路径传输，避免了“流中断”。

|  |
| --- |
|  |
| 图3.8 流表安装图 |

对于新老流表共存于交换机中的处理方法有两种，一是是不做任何处理，使之自然超时触发流表失效的操作，二是采取主动操作去删除原有路径的流表项。对于第一种处理方式，对于本文的研究并不适合，因为，原有的路径的流表项己经失效，却占据流表的有限资源，这是一种浪费，此外，由于新的路径中的流表项设定了较高的优先级，若原有路径的流表项未被清除，则新路径的流表必须保持较高的优先级，这会带来额外的优先级管理问题。因此本文用了主动删除原有路径的流表项的机制，在新的流表项重新安装完之后，删掉旧的流表。

### 3.3.2 队列调度策略的分析设计

在OpenvSwitch 虚拟交换机中，当数据业务到达出端口时，OpenvSwitch的出端口将按照 FIFO（First In First Out，先入先出）的转发规则把缓冲区中的数据包转发出去。OpenvSwitch交换机在出端口上采用了“尽力而为”的传统转发规则，是一种抢占式的资源分配方法，当某一类型的业务所占的带宽多时，会导致其它类型的业务占用带宽减少，这样有可能会出现重要性低的业务抢占过多带宽资源，而造成重要性高的业务短时间内无法得到转发的情况。因此，为保障流媒体视频流的QoS，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协议能够提供队列配置的功能，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，比如FIFO 、PRIO、HTB等。其中，FIFO只能提供尽力而为的服务，不能对不同业务做区分处理，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，但容易使得低优先级队列中的业务被“饿死”；HTB既能对不同优先等级业务进行区分调度，也能对网络进行流量整形和速率限制。

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。根据本文的需求以及对各个队列调度规则的对比，使用HTB（分层令牌桶，Hierarchical Token Bucket）队列规则来完成队列调度。

HTB队列规定是树状结构，能对业务按照树状结构进行分类[42]。它使用DWRR算法，按照类的优先级进行数据包的差额加权轮询调度，将不同优先级的数据包调度到相应类的队列规定中；使用令牌桶算法对流量进行限速和整形。下面对这两种算法分别进行介绍。

**a、差额加权轮询算法**

DWRR算法是对RR算法与WRR算法的改进[43]，因此先介绍RR算法与WRR算法，然后再阐述DWRR算法的原理。

轮循算法（Round Robin，RR）是最简单的轮询算法，按顺序轮流调度各个队列，若队列为空则跳过，实现调度的公平性。虽然RR队列调度算法能体现公平与工作尽力的原则，但是并不能体现业务的优先等级，实现区分调度服务，这导致了不能满足较高优先级业务的服务质量。

加权轮循算法（Weighted Round Robin，WRR）对RR算法进行了改进[44]，是一种加权轮询报文调度算法。它为每一个队列都分配一个独立的权值和计数器，初始时，每个队列中的计数器值被初始化为队列的权值。在每一次轮循中，当有效队列得到调度机会后，每调度一个报文，队列计数器的值就减小1，当队列没有报文可被调度或者计数器值为0时，计数器的值被重新设置为队列的权重值，然后切换到下一个有效队列中继续进行调度。没有报文的队列不参与调度，带宽被释放以供其他有报文的队列使用。当一个队列的流量较少时，其余队列还可分享其带宽，WRR队列调度算法能在各个队列间灵活分配带宽，为不同等级业务提供差异化的服务，图3.9为WRR算法原理图。

|  |
| --- |
|  |
| 图3.9 WRR算法原理图 |

WRR加权轮询调度算法可以为不同类数据提供不同的优先级服务，但是WRR算法是基于数据包的，拥有较多较长报文的队列会得到更多的带宽资源，出现不公平的现象。为了克服WRR队列调度算法存在的缺陷，算法设计人员们提出了差额加权轮询（Deficit Weighted Round Robin，DWRR）算法。

DWRR队列调度算法是对WRR算法的改进，其队列权值不是报文的个数，而是报文的字节数，为每个队列维护一个权值 Q（Q 为定值），和一个差额 D。算法最初运行时需设置每个队列的Q值，并将每个队列的D值变为0。在每一轮调度中，一个队列需发送尽量多的的数据包，且发送的字节数少于等于 Q+D，并保证按序发送，若发送了N个数据包，则需满足Q+D大于等于这N个数据包长度，且这N个数据包长度加上这个队列下一个要发送的数据包长度需大于 Q+D，发送完毕后，若队列为空则把D值设置为0，不为空则 D=Q+D-send。send为这个队列这一轮发送的字节数。DWRR算法可以为不同业务，提供更为精准的优先级服务。

**b、令牌桶算法**

令牌桶算法能对网络流量进行整形和限制流量速率，可控制发送到网络上数据的数目并允许突发数据的发送。图3.10为令牌桶算法的流程图。

|  |
| --- |
|  |
| 图3.10 令牌桶算法流程图 |

令牌桶有一个容量固定且可以自行以恒定的速率源源不断地产生令牌，数据包经过令牌桶时，如果桶中有足够的令牌，则发送数据包，同时减去相应的令牌数。如果桶中的令牌不足，那么数据包就不能被发送，需要等待，直到令牌桶中有足够的令牌。令牌桶传输突发数据和限制速率分别通过下面的两种方式实现的。

（1）令牌产生的速度大于数据包的传输速率，如果这样一直下去，最终会使得令牌不断增多直至达到令牌桶的容量，然后开始溢出。当大量突发数据包到来时，令牌桶中就会有多余的令牌来处理这些突发的峰值流量。

（2）令牌产生的速率小于数据包的传送速率，这样一直下去就会导致令牌桶变空，就会使得数据包的发送速率等于令牌的产生速率，起到了限制流量的作用。

**2）队列调度策略的设计**

图3.11展示了队列调度策略的设计，数据流在从交换机的端口进入后，经过流表的匹配执行端口转发、数据包标记、入队列动作，不同优先级业务报文进入不同的队列，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，也可以优先被调度出端口，转发到下一个交换机上。

|  |
| --- |
|  |
| 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

本文将遗传算法应用到SDN网络中解决QoS路由问题，根据本文的应用场景，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[45]分为可加性指标，乘性指标，凹性指标。可加性指标，即路径的 QoS为该路径每条链路QoS要求值的和，比如跳数、链路成本、时延等。乘性指标，即路径的QoS为该路径每条链路QoS要求值的乘积，比如丢包率、链路利用率等。凹形指标，即路径的QoS为该路径每条链路QoS要求值的最小值或最大值，最小值如带宽。

网络QoS路由选择问题的理论基础是图论中的寻路问题，通常将网络描述为有向图G (V,E) ，其中，V代表图中顶点的集合，用于描述网络中的Open Flow交换机或主机，E 代表网络中边的集合，描述连接各个网络设备的链路（控制器和交换机的链路不包括在此集合内）。每条边。

设有一条路径p,为路径的起点，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：

|  |  |
| --- | --- |
|  | (3.11) |
|  | (3.12) |
|  | (3.13) |

公式(3.11)为该条路径端到端时延，公式(3.12)为传输路径总的丢包率，公式(3.13)为传输路径中剩余的带宽。

对于一个端到端的传输需求，需要找到一条路径，分别满足以下约束条件：

|  |  |
| --- | --- |
|  | (3.14) |
|  | (3.15) |
|  | (3.16) |

其中D、L、B分别表示传输对时延、丢包率及带宽的需求，D的值为10ms，L的值为，B的值为0.8Mbps。

### 3.4.2 遗传算法基本概念

遗传算法是基于自然选择和遗传进化学的自适应启发式随机搜索算法，虽然强调随机化，但是遗传算法并不是完全随机的，而是利用历史遗留信息将搜索范围引导到可能获得更优解的搜索空间中。遗传算法基本概念如下：

（1）种群：种群是算法待解决问题的所有可能解的编码方案的一个子集。

（2）染色体：一个染色体代表种群中的一个个体，对应待解决问题的一个解，个体与染色体概念等同。

（3）基因：基因是染色体一个位点上的具体元素。

（4）染色体编码：指种群中每个染色体内部基因的编码方式，用于建立实际问题解与计算机可以理解的数据结构之间的映射。

（5）适应度函数：适应度函数是用于对种群中染色体对于当前环境适应程度的一个量化函数，计算结果代表一个染色体的适应值，反映解空间一个解对当前问题解决程度的优劣。

（6）遗传算子：遗传算子是遗传算法中的进化操作，包括选择算子、交叉算子和变异算子等。

遗传算法模拟的是生物进化的过程，算法执行时首先需要得到一个初始种群，选择种群中的部分染色体执行交叉和变异算子从而生成新的子代染色体。然后用子代中的适应值高的染色体替代父代种群中的适应值较差的染色体，重复这个过程直到算法收敛。

### 3.4.3 染色体编码设计

染色体编码是实际问题与计算机联系的桥梁，是遗传算法的基础和关键。合适的染色体编码对问题的解决有很大的助益，可以将实际问题空间直观地映射到具体的机器空间，同时又能兼顾计算机对算法执行的效率。常用的染色体编码有符号编码、二进制编码和基于树的表示。

由于路由经过的节点数目可以不同，即染色体的长度可以是变化的，QoS路由算法染色体编码选择变长符号编码的方式。网络中每个交换机都有唯一的datapath.id的标识，将一路径上的每个交换机的datapath.id按照途经的顺序排列作为一条染色体的编码，染色体第一个位置的基因代表路由的源节点，最后一个基因代表路由的目的节点，基因的排列顺序表示路由经过的网络节点的顺序。染色体长度不能超过网络中节点的总数，因为路由永远不需要比网络所有节点更多的节点来构成。以图3.12网络拓扑为例，用箭头标出的路径用染色体编码表示的方式为[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 种群初始化方法设计

种群初始化是算法的第一步，执行初始化会产生一定数量的染色体，它们将构成算法运行的初始种群。通常种群初始化需要考虑三个要素[46-47]：种群模型、种群初始方式、种群数量规模。

遗传算法的种群模型分为稳态和代数两种方式。稳态模式中，算法每次迭代产生一或两个新的染色体，并用它们替换原种群中的一个或两个染色体，因此也被称为增量GA。代数模式中，每次迭代产生n个新的染色体，其中n就是种群的规模的值，然后用新的染色体集合替换原先整个种群。本文采用稳态模式作为QoS路由算法的种群模型。

初始种群生成通常有随机和启发两种方式，随机初始化用完全随机的方式来生成初始种群，启发式初始化利用实际问题的已有启发式方法填充初始种群。为使QoS路由算法有更大的搜索空间，我们采用随机初始化的方法。

种群规模表示遗传算法中每代种群中的染色体总数量，较小的种群规模，算法在运行时计算量小，但是可能导致算法搜索空间陷入局部最优区域，降低解的质量；较大的种群规模，算法搜索空间相对较大，有助于寻找全局最优解，但是每代种群进化的计算量相应增大，计算量的增加会增加计算时间，进而会降低算法的时效性，为了遗传算法有最佳表现，种群规模的需要设置一个平衡的值，兼顾计算效率和解的合理性。针对本文的拓扑设计情况，将拓扑中所有路径的30%作为该拓扑的初始种群。

### 3.4.5 适应度函数设计

适应度是评价一条染色体所代表路径的优劣程度的依据，选择算子依据适应度选择染色体，适应度更好的染色体将有更高的概率被选入下一代的染色体之中去。染色体适应度由适应度函数来计算，因此适应度函数的设计尤为关键，其直接影响到算法收敛的速度以及能否找到优化解。

常用的度量参数有带宽、时延、丢包率等，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测度个数。

|  |  |
| --- | --- |
|  | (3.17) |

为染色体ｉ关于时延度量参数的适应度计算函数，如公式(3.18)，其中为染色体i所表示的路径的时延总和，m为一代群体的数量。

|  |  |
| --- | --- |
|  | (3.18) |

为染色体ｉ关于丢包率测度的适应度计算函数，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)。

|  |  |
| --- | --- |
|  | (3.19) |
|  | (3.20) |

时延与丢包率测度的适应度计算函数，都除以了该代所有染色体相应测度和，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，时延对于路径的选择的影响会大于丢包率，这样选择的路径可能过多的考虑了时延，而忽略了丢包率。做了相应的处理后取值范围在同一范围，被考虑的可能性就相对均等。

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 遗传算子的设计

**1）选择算子设计**

从群体中选择优胜的个体，淘汰劣质个体的操作叫选择。选择的目的是把优化的个体直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。

一条染色体根据适应度函数求得的适应度的值越小，说明该条染色体所代表的路径更符合要求。为了防止当前群体的最优个体在下一代发生丢失，导致遗传算法不能收敛到全局最优解，本文的选择算子使用“精英选择(elitist selection or elitism)”策略，该策略的思想是把群体在进化过程中迄今出现的最好个体(称为精英个体（elitist)）不进行配对交叉而直接复制到下一代中。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，将其染色体按照适应度由小到大顺序排列，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) 将剩余的染色体执行交叉算子，并传入下一代。

**2）交叉算子设计**

交叉指把两个父代个体的部分结构加以替换重组而生成新个体的操作。交叉算子将种群中的两个个体随机地交换某些基因（本文指的是交换机节点），能够产生新的组合，交叉算子将产生新的个体，扩大遗传算法的搜索空间。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，对剩下的所有染色体执行交叉算子，且只有除头尾基因外有相同的基因的双亲染色体才能进行交叉，因此不设置交叉率，只对交叉方式进行设计。

遗传算法中的交叉算子一般是通用的，但设计算法时，需要根据具体问题选择相适应的交叉算子。算法实践中常用的交叉算子有单点交叉，多点交叉，融合交叉等，其中单点交叉是在染色体中随机选择一个点，作为交叉点，然后在该点将父代染色体部分基因交换，得到新的染色体，如图3.13所示。

|  |
| --- |
|  |
| 图3.13 单点交叉示意图 |

QoS路由算法中交叉算子的设计借鉴了单点交叉的思想，将两条路由的部分路径片段进行交换，产生新的路由，具体是先从两条路由中选出一个节点，将路由分为两部分，一部分是从源节点到选出的节点，另一部分从选出的节点到目的节点，然后相互交换选出的节点后面的路由片段。

同时，路由算法中采用的单点交叉算子同上文提到的常规单点交叉又有些区别。本文算法要求双亲的染色体头尾基因必须相同，即源目的节点要相同，且双亲的染色体除了头尾基因外，必须包含至少一个相同的基因片段，但是并不要求该相同基因片段在染色体中的位置也必须相同。按照物理上讲，即二者染色体代表的路径除去起点和终点，至少包含一个相同的节点，但是路径经过该相同节点的顺序可以不同。这些相同的基因片段作为潜在可行的交叉点，此时交叉算子的执行是等概率的随机选择潜在的交叉点，并以该交叉点为中心，将两条染色体的前后部分分别互换，从而生成两条新的染色体，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**

变异算子的目的是在种群中保持和引入基因多样性，扩大搜索范围，能够一定程度的防止局部最优解的出现。变异算子的设计包括变异概率的设置和变异方式的设计。

变异概率决定了一个染色体发生变异的概率，遗传算法的变异率并没有明确的值，但有些变异概率确是比其他变异概率提供好很多的结果。较高的变异率允许种群中有更多的遗传多样性，这样扩大了搜索范围，有助于避免陷入局部最优。然而，变异率过高会导致过多的遗传特性变异，可能导致失去前代种群中的优良解。最好的变异概率应该设置为这样一个值，该值允许足够的多样性，以防止算法搜索停止不前，同时算法在变异中又可以较好地保留原种群中有价值的遗传信息。通过改变变异概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，故将变异概率的值设为1%。

常见的变异算子有翻转、随机重置和交换突变。本文的变异算子选择的是随机重置，随机重置是在染色体中随机选择一个基因，并赋值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，遗传算法的终止条件一般是迭代次数，但作为QoS控制这一应用场景下，需要提高算法收敛的速度，因此设置了另一终止条件，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。N的值可以根据具体的场景进行设置，若是最初的QoS服务请求，N可以设定一个较大的值，从而可以提供更好的服务，若是视频流传输时出现劣化需要进行QoS控制时，则将Ｎ设置为一个较小的值，从而确保算法能够在较短的时间内收敛，更快寻找到重路由的路径，减少拥堵时间太长造成视频流的吞吐量下降等问题。

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为下一代的种群数，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

本章首先阐述了SDN网络中QoS控制策略的需求分析，对整体的QoS控制框架的设计进行了介绍。然后依据 Ryu控制器、OpenFlow 协议和OpenvSwitch，对框架中各个模块的作用和设计原理进行了具体阐述，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策略控制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模块实现了周期地测量网络带宽、丢包率、时延、链路拥塞率等QoS性能参数的功能。Topology Manager模块能够获取网络拓扑视图，并将链路性能信息与链路的连接关系结合起来，让控制器拥有更详细和准确的网络全局资源信息，形成全局QoS拓扑视图。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中的实现 |

在OpenvSwtich软件交换机中实现了QoS控制策略框架的队列调度模块，主要负责在数据转发层面为业务提供QoS保障。下面将阐述QoS控制框架中各个模块在Ryu控制器和OpenvSwitch软交换机中的具体实现。

## 4.2 QoS路由的实现

### 4.2.1 链路性能测量模块的实现

SDN中OpenFlow协议作为南向接口协议被广泛应用，由第三章可知OpenFlow协议并不是专门为测量网络链路性能而设计的，且OpenFlow交换机不会向控制器直接提供交换机链路的性能指标信息，但是支持OpenFlow协议的交换机内部有相关统计运行信息的计数器，用于记录每个Port、Flow 等信息，这些计数器记录的信息足够支持完成链路性能测量工作。下面将详细介绍链路性能测量模块的实现。

**1）链路性能测量模块框架**

链路性能测量模块周期地对链路进行测量，获取全网链路QoS性能信息。图 4.2展示了链路性能测量模块相关类和接口之间的关系，下面说明其中各个类和接口的功能。

|  |
| --- |
|  |
| 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()方法用以周期性地向网络中的所有交换机发送Port\_Stats\_Request和Flow\_Stats\_Request请求报文。类中的\_flow\_stats\_reply\_handler()方法以及\_port\_stats\_ reply\_handler()方法是在交换机收到请求报文后，将需要的流信息和端口统计信息上报给控制器。

BandWidth类定义了\_get\_allbandwidth()、\_get\_freebandwidth()和\_get\_bandwidth\_ congestion ()三个方法，分别用于计算网络拓扑链路的总带宽、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类定义了\_get\_delay()方法用以计算全局网络拓扑链路的时延。

BandWidthDisplay、PacketLossRateDisplay、DelayDisplay三个接口分别定义了show\_bandwidth()、show\_packet\_loss\_rate()、show\_delay()三个方法用以展示链路间的带宽、丢包率、时延等信息。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类负责缓存交换机端口统计报文中与测量相关统计数据，time变量保存测量的时间戳，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**

由于带宽、链路拥塞率以及丢包率性能信息都是通过分析和计算OpenFlow交换机端口或流统计信息获得的，本节将链路带宽、链路拥塞率和丢包率测量的实现合并介绍。

Traffic类中的\_monitor()方法是一个周期执行的线程，该线程每2s向网络中的所有交换机发送Flow\_Stats\_Request和Port\_Stats\_Request请求报文，交换机接收到这两个请求报文后，将端口的流量统计信息使用Port\_Stats\_Reply、将流的统计信息使用Flow\_Stats\_Reply应答报文进行包装后发送给Ryu控制器，控制器收到应答报文后进行解析，然后得到端口的流量以及流的统计信息。本文使用统计信息中交换机端口发送的字节数与接收的字节数，以及两次统计的时间差来计算得到端口的速率，见公式(3.3)；使用OFP\_Port\_Stats消息中的字段，OFP\_Flow\_Stats消息中的字段和字段来计算得到链路两端发送与接收的packets，见公式(3.7)和(3.8)。

OpenvSwitch端口的带宽为20Gbps，为了符合实际情况，需要对交换机端口的实际带宽进行限制，以作为连接该端口链路的带宽。本文在Ryu启动配置文档中统一设置所有链路的带宽值，这个值对应于交换机端口所设置的实际带宽值。Ryu控制器启动后，链路性能测量模块会发送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。

在得到速率与端口带宽后，链路性能测量模块会使用公式(3.5)和(3.6)计算得到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和拥塞率以及丢包率的流程图。图4.4、图4.5以及图4.6分别为链路剩余带宽和拥塞率以及丢包率的终端展示，可以更直观的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

|  |
| --- |
| C:\Users\Dell\Desktop\性能参数图\剩余带宽 (3).png |
| 图4.4 剩余带宽展示图 |



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是链路间时延信息的展示图，下面阐述每个类的功能。

OFChannelHandle类用以处理OpenFlow交换机和Ryu控制器间信息交互的连接，它的send\_echo\_request()方法向交换机发送EchoRequest请求报文，同时携带有发送请求信息时的时间，当交换机收到请求报文后会向控制器应答携带有控制器请求信息时间的EchoReply 报文，控制器使用process\_echo\_reply()方法处理 Echo Reply报文后，会提取所携带的请求报文时间，Ryu控制器接收EchoReply报文的时间与发送EchoRequest 请求报文的时间的差值就是交换机和控制器间往返通信的 RRT时间。

LinkDiscoveryManager类发送LLDP消息来发现OpenFlow交换机之间的链路连接并维护链路的状态。它的 generate\_LLDPmessage()方法向网络中的OpenvSwitch软交换机发送LLDP报文并在报文中插入报文下发的时间戳，对应的软交换机收到 LLDP 报文后会向它相邻的交换机发送该LLDP报文，当邻近交换机收到LLDP 报文后把这个报文返回给Ryu控制器；在收到这个LLDP报文后，控制器会使用 handle\_LLDPmessage()方法来处理这个LLDP报文，然后解析该报文中Ryu控制器插入的时间戳并存储这个时间，控制器收到LLDP报文的时间记为 ；与的差值即为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()方法是根据LinkDiscoveryManager类和OFChannelHandle类计算获得的时间数据并通过公式(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 |



图4.8 时延信息展示图

### 4.2.2 拓扑管理模块的实现

拓扑管理模块负责为全局提供拓扑结构，并为QoS路由算法提供带有最新的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。图4.9展示了拓扑管理模块相关类的关系，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类是物理网络拓扑的抽象图，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()方法是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路由算法提供最新资源，Dijstra()方法将拓扑结构转换成邻接矩阵供Dijstra算法使用。

Edge类保存链了路的性能参数信息，其中s、t、bandwith、loss Rate、delay以及congestion rate分别表示保存当前链路的源交换机编号、目的交换机编号、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Dijstra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，并将需要的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。路由管理模块所涉及到的类和接口关系如图 4.10所示，下面详细阐述这些类和接口。

PacketInMessageHandler类是对PacketIn报文的处理，其中do\_drop()、do\_flood()、do\_forwarding\_flow()方法分别为丢弃处理、泛洪处理、转发处理，转发处理即是对有路由请求的数据流进行路径计算及转发。

RoutePlanner接口定义了calculate()方法，用以路由的计算。QoSRoutePlanner类和DijstraRoutePlanner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，其他等级数据流的基于Dijstra算法的路由。

Graph类是为QoSRoutePlanner类和DijstraRoutePlanner类提供带有链路性能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，add\_path()与del\_path()方法分别为添加新的路径和删除旧的路径。

|  |
| --- |
|  |
| 图4.10 路由管理模块类与接口图 |

**2）路由管理模块实现**

SDN网络中，当有数据包进入到OpenFlow交换机时，交换机会对数据包进行流表匹配操作，若匹配流表项成功，执行该流表项中规定的动作；若匹配失败，交换机会将该数据包封装到packet\_in报文中发送给Ryu控制器，控制器路由管理模块中的PacketInMessageHandler类会对该packet\_in进行不同的处理。为了对最高优先级的视频流媒体数据流使用QoS路由，本文对do Forwarding Flow()方法进行了扩展，使得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，其他等级的业务数据流基于Dijstra算法进行最小跳数路由。

图4.11是修改后的do Forwarding Flow()方法的执行流程。当PacketInMessageHandler类收到packet\_in 报文后，经解析是路由请求，将其交给do Forwarding Flow()方法，do Forwarding Flow()方法首先提取报文中的有效负载数据，随后会依次解析有效数据中的源交换机、目的交换机和应用层协议，若识别出为非最高等级业务数据流，Ryu使用以链路跳数为代价的Dijkstra算法来求取源交换机到目的交换机之间的路径，然后根据该路径向交换机下发流表，以指导非最高等级业务数据流在网络中的传输。如若识别出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，删除旧的路径，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，此部分是队列调度的基础。

|  |
| --- |
|  |
| 图4.11 do Forwarding Flow()方法的执行流程 |

### 4.2.4遗传算法的实现

**1）遗传算法模块框架**

本节对基于遗传算法的QoS路由的实现进行阐述，算法的设计在前一章节已经进行了介绍，图4.12展示了路由算法实现的相关类之间的关系，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，随机生成种群中的个体；population()实现初始种群的生成，利用链路保存一组染色体，该类是遗传算法进化计算的作用对象；fitness()是适应度函数的计算；evolve()是遗传算法逻辑主体，内部实现了各种遗传算子。

|  |
| --- |
|  |
| 图4.12 遗传算法模块框架图 |

**2）种群初始化策略实现**

Genetics类是种群初始化的核心类，其中individual()方法随机生成从起点到终点的路径作为初始种群，它也是QoS路由算法的重要方法，算法的流程如图4.13所示。该算法是对深度优先遍历算法的一种改进算法，考虑到路径生成时需要尽可能保证路径的随机性，因此方法执行流程中，每次选择未被标记的邻接节点之前，先对其进行随机排列，保证下一个加入路径的节点是完全无规律的。同时，方法对已经访问过的节点进行标记，保证生成的路由没有环路。当遇到一个节点所有邻接节点都已经被标记时，执行回退操作，直到找到一个点有邻接节点未被标记时停止，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，直到达到种群规模。这样第一代种群中的每个染色体都是不同的，有利于保证种群基因的多样性，扩大搜索空间。种群初始化流程如图4.14所示。

|  |
| --- |
|  |
| 图4.14 种群初始化流程图 |

**3）适应度值的计算**

适应度值的计算由 Genetics类的fitness()方法实现。计算适应度值的流程如图4.15所示，计算染色体适应度值时，首先设置适应度计算公式，取得保存路径的链表，即路径包含的链路。然后通过拓扑管理模块提供的每条链路具体的QoS性能信息，获得路径的QoS 性能指标，最后依据适应度函数计算每条染色体的适应值。由于时延与丢包率测度的适应度计算函数都除以了该代所有染色体相应测度和，所以在获得路径链表后，应计算出所有染色体相应测度和，以便每个测度的适应度值的计算。

|  |
| --- |
|  |
| 图4.15 适应度值计算流程图 |

**4）遗传算子的实现**

本节阐述基于遗传算法的QoS路由用到的遗传算子的实现，包括了选择、交叉、变异等，这些算子都通过Genetics类进行实现。

**a、选择算子**

精英选择策略实现流程如图4.16所示,精英选择策略以当前种群数为参数，首先计算种群中所有个体的适应度的值，并将适应度值的大小进行排序，然后根据设定的保留率，选择群体中较好的精英个体，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是在0-1之间。

|  |
| --- |
|  |
| 图4.16 精英选择法流程图 |

**b、交叉算子**

交叉算子实现流程如图4.17所示，对经过精英选择策略后剩下的个体进行交叉，在这些染色体中随机选择两个染色体，若染色体中有相同的基因，则随机选取基因点进行交叉，由于在交叉之后形成的路径可能会出现环路的情况，因此需要对交叉后形成的新的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**

变异算子使用的是随机重置，随机重置方法的实现如图4.18所示，遍历所有经过选择与交叉后的下一代，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，变异的位置为该染色体中随机选择的基因位置，并随机重置为新的值。

|  |
| --- |
|  |
| 图4.18 变异算子实现流程图 |

**d、终止校验算子**

终止判断流程如图4.19所示，本文实现了两种终止判断方式，一是迭代数，算法执行过程中种群代数达到设置的最大进行代数，算法将退出搜索过程，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。这样做是因为在QoS控制这一应用场景中对时间要求非常敏感，提高算法收敛的速度，可以提升路由效率，提高QoS控制的性能。

|  |
| --- |
|  |
| 图4.19 终止判断流程图 |

## 4.3 队列调度策略的实现

实时视频会议与监控视频等视频流媒体业务对时延、带宽等QoS参数有较高的要求，其他业务对QoS要求依次降低。依据表3.1，本文将实时视频与监控视频、关键数据业务、事物处理与交互式数据、数据同步与e-mail优先级分别映射为32、24、16、8。根据第三章提出的队列调度的框架，将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，控制器根据其四层协议将数据流进行分类，并根据分类结果，将数据流的IP头中的DSCP值标记为对应的优先级，如表3.1。对DSCP值的修改是利用了OpenFlow动作集中的修改域，修改域可选的字段部分如图4.21所示。

|  |
| --- |
|  |
| |  | | --- | | 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等级的数据包放入不同的对列，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。

|  |
| --- |
|  |
| 图4.22 修改DSCP值的程序图 |

HTB队列调度规则是通过对 Open vSwitch 的linux-htb队列模块进行配置来实现的,具体配置如图4.23所示。

|  |
| --- |
|  |
| 图4.23 HTB队列规则配置图 |

其中，type是配置OpenvSwitch端口队列的类型；dscp用于配置进入队列的业务流的优先级，具体值见表3.1；max-rate是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表示桶大小，指能处理的突峰发送速率；protiory用于配置DWRR的优先级，用来保证重要的数据进入优先队列而优先发送；quantum表示在进行 DWRR 算法调度时，每次最多调度的字节数。图4.24、图4.25与图4.26分别是是端口列表查询、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。

|  |  |  |
| --- | --- | --- |
| |  | | --- | | C:\Users\Dell\Desktop\2019-01-02 04-37-42屏幕截图.png | | 图4.24 端口列表查询图 | |
| |  | | --- | | C:\Users\Dell\Documents\Tencent Files\1762871539\Image\C2C\Image1\)W6LZK@~]B42@YN4Y5U@ROM.png | | 图4.25 端口qos查询图   |  | | --- | | C:\Users\Dell\Documents\Tencent Files\1762871539\Image\C2C\Image1\G~7{C(]ZQ{TP)}DKTI7H09O.png | | 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控制策略框架中各个模块的实现，主要是QoS路由模块中的拓扑管理模块、链路性能测量模块以及路由管理模块的实现过程，分别对各个模块的具体实现类和重要方法的执行流程做了详细介绍。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]采用轻量级的虚拟化技术使得单一的系统可以仿真一个完整的网络，即使在一台电脑上，也可以很容易的创建出包含主机和交换机的网络，模拟的网络拓扑最多可有4096台主机的网络结构。Mininet支持OpenFlow协议各种版本以及OpenvSwitch，可以很方便的进行SDN应用的验证和测试。Mininet支持系统级别的回归测试，除了本身提供带有基本参数的常见拓扑结构之外，还支持用户自定义拓扑结构，可以创建任何需要的网络拓扑结构。除此之外，基于Mininet开发的控制器相关的代码可以直接地迁移到真实的网络环境中，即由OpenFlow 交换机组网的硬件环境，这样极大地方便了用户开发与 SDN 相关的网络应用，以非常低的成本进行新应用的开发、测试和验证，而不需要浪费时间去搭建真实的网络，大大缩短了SDN应用开发测试的周期，具有很高的灵活性。

Mininet提供多个python API (Application Programming Interface,应用程序编程接口)，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的命令行接口(CLI)调用，在CLI上，通过nodes查看网络的节点，使用net查看网络链接情况，通过links获取节点间的连接状态，通过link禁用或启动两个节点间的链路。

表5.1 Mininet网络相关函数功能

|  |  |
| --- | --- |
| 函数 | 功能 |
| addHost() | 为网络设置主机，可配置相关参数 |
| addSwitch() | 为网络设置交换机，可配置相关参数 |
| addLink() | 为网络设置链路，可配置相关参数 |
| start() | 开启网络 |
| pingall() | 控制所有节点相互执行ping命令，测试连通性 |
| stop() | 停止网络运行 |
| net.hosts | 网络中所有主机 |

### 5.1.2 OpenvSwitch

SDN架构中基础设施层的转发设备也可由软件的形式实现，目前软件交换机也已能够满足很多应用场景中的数据传输，因此以软件交换机的形式构建SDN网络，已成为可选方案之一。

OpenvSwitch[50]是一个开源的多层虚拟交换机，它支持现有的标准管理接口和协议（如 CLI，LACP，802.1ag 等）。OpenvSwitch既能够在虚拟机环境中作为软件交换机(soft switch)运行，也能够嵌入到硬件交换机切片中作为控制软件运行，其支持OpenFlow协议及用于虚拟化的其他扩展。

### 5.1.3 其他实验环境及相关软件介绍

仿真实验所用的操作系统是Windows7专业版64位；安装内存（RAM）：8.00GB；处理器：Intel(R)Core(TM)i5-3230M [CPU @ 2.60GHz](mailto:CPU@2.60GHz) 2.60GHz。

虚拟机：VMware Workstation10.0.1build-1379776；虚拟机系统：Ubuntu14.04。

多媒体播放器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 网络环境的搭建

视频传输的仿真平台主要由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，网络中包含s1，s2，s3，s4，s5，s6，s7，s8 八个虚拟交换机，它们由共同的远端Ryu控制器控制；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和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创建一台Ubuntun虚拟机，安装Ryu和Mininet。为了在Mininet上成功地创建如图5.2所示的实验拓扑，首先需要编写自定义的 Python 脚本 cai-topo.py，如图5.3所示，并放在/home/mininet/custom 路径下，然后在Mininet中运行此脚本，实现与指定控制器相连，若运行窗口出现图5.4所示的情况，且运行拓扑管理模块出现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。

|  |
| --- |
|  |
| 图5.3 自定义拓扑程序图 |

|  |
| --- |
| C:\Users\zhu\Documents\Tencent Files\1762871539\FileRecv\2018-12-17 19-24-26屏幕截图.png |
| 图5.4 运行结果及链路信息图 |

|  |
| --- |
| C:\Users\zhu\Documents\Tencent Files\1762871539\FileRecv\2018-12-17 20-19-36屏幕截图.png |
| 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 USB2.0 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。

|  |
| --- |
| C:\Users\zhu\Documents\Tencent Files\1762871539\FileRecv\MobileFile\50D3C333ED7ABE73A8433A328570AD49.png |
| 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控制策略对视频流传输性能的提高。

为了模拟真实网络中的流量情况，在每个实验开始之前，分别使用iperf[51]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，具有高QoS要求的视频流媒体业务应尽量选择时延小、丢包率低的链路，以保障其对服务质量的需求。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Dijstra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。

|  |
| --- |
| C:\Users\Dell\Documents\Tencent Files\1762871539\Image\C2C\Image1\RXK}Y6$3M~K4(M]TP$HELCC.png |
| 图5.7 时延变化情况图 |

|  |
| --- |
| C:\Users\Dell\Documents\Tencent Files\1762871539\Image\C2C\Image1\I(@%_YC~IW(}E39`J~RM5GV.png |
| 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，虽然两者都逐渐趋于稳定，但是使用遗传算法所选路径的时延和丢包率都小于使用Dijstra算法所选路径，并且使用遗传算法所选路径的时延和丢包率均在设定的QoS指标范围内。即丢包率在以内，时延在10ms以内。综上所述，基于遗传算法的QoS路由计算提高了视频流的传输性能。

### 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QoS保障，但是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将会受到极大影响，此时必须使用动态路由为其优化路径，才能保证其QoS。动态路由的有效性分析，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监控网络中的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，由于网络中流量的增加，视频流媒体的吞吐量由于链路的拥塞迅速下降，其时延抖动也随之增加，若使用动态路由策略，控制器监测到链路拥塞后，通过控制器的动态路由机制，使用遗传算法重新为视频流媒体计算一条符合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，只是在最初为视频流媒体选出一条符合需求QoS需求的路径，并未在网络状态发生变化时动态改变传输路径，各个等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，而延时抖动也是经历了两次跳跃才恢复至稳定。

峰值信噪比PSNR（Peak Signal Noise Ratio）是评价视频质量最通用的标准。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，即传输路径中注入大量其他类型的业务流导致视频流传输出现拥塞时，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Dijstra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。

|  |
| --- |
|  |
| 图5.12 队列规则配置图 |

server1使用iperf工具向client1发送6组UDP报文，每组报文发送速率从1000kbps递增到7000kbps，递增的梯度为1000kbps，组内不同优先级业务报文发送速率相同，每组报文传输时间为 2 分钟，最后记录客户端的平均接收速率。图5.13展示了未配置调度策略的情况下不同等级业务报文的接收速率；图5.14展示了配置了调度策略的情况下不同等级业务报文的接收速率。



图5.13 未配置调度策略接收速率图



图5.14 配置调度策略接收策略图

从图5.13中可以看到，在没有配置端口调度策略时，不同种类业务报文的接收速率和发送速率基本保持一致，当发送速率为 4000Kbps 的时候，传输链路出现了拥塞，接收速率达到了最大值；在发送速率超过 4000Kbps 后，随着发送速率的增加，4种业务报文的接收速率在 3800 Kbps 上下摆动。这表明，在未配置端口调度策略时，网络对4种不同优先级业务的报文仅仅只提供了尽力而为的服务，并没有区分保障它们的QoS。

从图5.14中可以看到，在配置调度策略后，随着报文发送速率的增大，优先级最低的数据同步类业务报文，它的接收速率最先降低；优先级第三的事务处理类报文在发送速率超过3000Kbps时，其接收速率开始下降；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；而优先级最高的视频流媒体业务报文，其接收速率随着发送速率的增加一直持续增涨。

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的时候，其接收速率也为2000Kbps，是因为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，但是随着发送的报文的速率逐渐增大，而同步类业务报文却逐渐减小，是因为网络中的剩余带宽逐渐减少，为优先保证高等级的报文传输，逐渐增加了对同步类报文的限制，且由于其优先级最低，所以其接收速率最先被限制。

综合以上分析，等级越高的业务报文，它的QoS越能优先得到保障，这说明区分业务级别的端口调度策略可以保障不同等级业务报文的不同QoS需求。

### 5.3.4 QoS控制策略性能分析

上述三个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，本实验将其综合起来，以验证QoS控制框架的整体传输性能。

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、其他三个等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%。

|  |
| --- |
|  |
| 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。其他类型的数据流相关设置如表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监控网络中业务流的吞吐量变化情况，图5.16为网络中业务流的吞吐量变化情况。

|  |
| --- |
|  |
| 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保证其QoS。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的路径进行传输。

## 5.4 本章小结

本章是对针对本文提出的QoS控制策略进行仿真实验分析。首先对实验的网络拓扑及相关设置进行说明。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 全文工作总结

随着网络技术的升级创新，网络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。然而传统的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，与传统的网络相比，为提高流媒体视频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顺序，解决了流表一致性问题，避免由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优先转发的“权利”，在底层保证视频流媒体业务QoS；且在带宽充足的时候，不同优先级的业务可以相互借带宽，保障不同优先级业务的QoS。

第三，通过 Ryu、Mininet 等软件成功模拟了网络实验环境，在文中给出了详细的逻辑架构以及物理拓扑和软件实现的过程。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。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 不足与展望

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进一步完善和改进。主要包括：

第一，在交换机层面队列调度中只实现了 DWRR 调度算法，今后重点改进的地方是加入随机公平队列 SQF 等队列设置，并且在交换机端口转发中添加可动态配置队列调度算法的功能。

第二，本文使用的网络拓扑结构单一，而真实的网络拓扑规模较大且结构相对复杂，如典型数据中心网络结构，因此本文所涉及视频流传输控制还需要在其他复杂网络拓扑中进行验证。

第三，整个控制策略只是在模拟环境进行了实现，没有实际在真实环境中去部署，可以考虑移植到真实网络中进行测试改进。