学校代号 10532 学 号 S141000887

分 类 号 TP393 密 级 普通

湖南大学
HUNAN UNIVERSITY

硕士学位论文

软件定义的 VANET 下流表用量感知的 QoS 路由机制研究

学位申请人姓名 查理佳

培 养 单 位 信息科学与工程学院

导师姓名及职称 付彬 讲师

学 科 专 业 计算机科学与技术

研 究 方 向 软件定义网络

论文提交日 期 2017 年 05 月 10 日

学校代号：10532

学　　号：S141000887

密　　级：普通

湖南大学硕士学位论文

软件定义的 VANET 下流表用量感知的 QoS 路由机制研究

学位申请人姓名：　查理佳

导师姓名及职称：　付彬 讲师

培　养　单　位：　信息科学与工程学院

专　业　名　称：　计算机科学与技术

论 文 提 交日 期：　2017 年 5 月 10 日

论 文 答 辩日 期：　2017 年 5 月 21 日

答辩委员会主席：　邝继顺 教授

Study on Flow Table Usage-aware QoS Routing Mechanism in Software Defined VANET

by

ZHA Lijia

B.E. (Shenyang Normal University) 2014

A thesis submitted in partial satisfaction of the

Requirements for the degree of

Master of Engineering

In

Computer Science and Technology

in the

Graduate School

Of

Hunan University

Supervisor

Lecturer FU Bin

May, 2017

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　日期：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密□。

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日

摘　要

VANET 可以为车辆提供丰富的安全和非安全相关的网络应用，但现有的 VANET 难以保障应用的 QoS 需求。许多研究者提出将软件定义网络的架构引入 VANET，从而分离 VANET 数据平面与控制平面，实现网络的系统化灵活控制，并带来可编程性。然而在软件定义的 VANET 架构下不同应用的 QoS 需求也各不相同，因此如何保障软件定义 VANET 下各类应用的 QoS 是一个值得研究的问题。许多研究人员进行了相关的理论和实践研究，当前研究在以下几方面有待进一步完善：其一，面向单业务，并发性差；其二，缺乏对业务动态接入和退出的处理；其三，不具有流表用量感知能力。本文针对以上不足，完成的主要工作如下：

首先，通过实验分析了流表溢出对控制器和 QoS 路由机制的影响，并提出了一种流表用量感知的 QoS 路由机制。由于 SDN 路由过程的具体实现依赖于交换机的流表管理，因此流表策略对路由机制性能有着重要影响。本文首先分析了流表溢出的现象及原因，接着通过实验验证了流表溢出对控制器和 QoS 路由机制的性能的影响。在此基础上，为了避免流表溢出带来的不良影响，本文提出了一种流表用量感知的 QoS 路由机制，实验结果表明该机制在路由过程中能够及时感知流表状态的变化，从而极大地减轻了控制器的负载，同时有效地保障了应用服务的 QoS。

其次，本文设计了一种面向异构多网接入的软件定义的 VANET 架构，并提出了一种流表用量感知的动态 QoS 保障框架，该框架允许使用模块化的方式管理网络，并支持业务流的动态加入和退出。在此基础上，建立了多业务流多约束条件下流表用量感知的 QoS 路由模型。该模型不仅考虑了丢包、时延和吞吐量等链路状态参数，还考虑了业务需求以及交换机的流表使用情况，从而为 VANET 应用提供并发的动态 QoS 路由。实验表明，本文提出的 QoS 路由机制不仅能够满足多业务对各自丢包、时延和吞吐量的要求，还能够感知流表用量，从而避免流表溢出对 QoS 路由的影响，进一步提高了网络 QoS 保障的性能。

关键词：软件定义网络；车辆自组织网络；服务质量；路由模型；流表

Abstract

VANET can provide a wide range of security and non-security related services. However, the existing VANET is difficult to guarantee the QoS of these application services. Many researchers proposed that the architecture of software defined network should be introduced into the VANET, which can separate the VANET data plane and control plane, realize the systematic and flexible control of the network, and make it programmable. However, the QoS requirements of different application services are also different under the software defined VANET architecture. So it is worth exploring how to protect the application of the software defined VANET QoS. In order to provide QoS guarantee for application service, many researchers have conducted theoretical and practical research, but most of the existing studies were facing the problems of single-function service with poor concurrency， lack of dynamic service access and exit and having no the ability to perceive amount of flow table. To solving the aforementioned problems, the main work of this paper is as follows:

Firstly, experiment was conducted to analyze the influence of flow table overflow on SDN controller and QoS routing mechanism. Since the implementation of SDN routing process has to rely on the management of the flow table of the switch, the impact of the flow table policy on the performance of QoS routing mechanism is important. The OpenFlow network flow table overflow phenomenon and reasons were analyzed, and then it was verified by experiments that the flow table overflow mechanism had important influence performance on SDN controller and QoS routing mechanism. In order to avoid the flow of QoS routing table overflow adverse effect, a flow table usage-aware QoS routing mechanism was established, which can the perceive the change of the state of the routing table. The experimental results showed that the mechanism can sense the overflow of the table overflow, reduce the load of the controller, and protect the application service QoS effectively.

Secondly, VANET architecture defined by heterogeneous multi access software was designed and a dynamic QoS support framework based on the

perception of flow table was proposed, which allowed us to manage the network in a modular way and supported the dynamic entering and exiting of the service flows. On this basis, a flow table usage-aware QoS routing model was established with multi-service flows and multi-constraints, which considered not only the parameters of link state such as packet loss, delay and throughput, but also the service requirements and the flow table usage, providing concurrent dynamic QoS routing for VANET application service. Experiments showed that QoS routing mechanism proposed in this paper can not only meet the service requirements of their packet loss, latency and throughput, but also was capable of perceiving the usage of flow table so as to avoid the influence of flow table overflow for QoS routing mechanism, which can further improve the performance of network QoS.

Key Words: Software defined network; Vehicular ad hoc network; QoS; Routing model; Flow table

目 录

学位论文原创性声明和学位论文版权使用授权书 I
摘 要 II
Abstract III
插图索引 VII
附表索引 VIII
第 1 章 绪论 1
1.1 研究背景 1
1.2 研究现状 2
1.2.1 VANET QoS 路由研究现状 2
1.2.2 SDN 研究现状 3
1.2.3 基于 SDN 的 QoS 路由机制研究现状 4
1.3 研究意义 5
1.4 本文主要工作 6
1.5 论文结构 6
第 2 章 相关技术与理论基础 8
2.1 引言 8
2.2 SDN 架构 8
2.3 OpenFlow 技术概述 10
2.3.1 OpenFlow 交换机 11
2.3.2 OpenFlow 协议 12
2.3.3 OpenFlow 控制器 13
2.4 Floodlight 控制器 14
2.5 Mininet 仿真器 15
2.6 小结 17
第 3 章 流表溢出对 QoS 路由机制的影响 18
3.1 引言 18
3.2 基于 OpenFlow 的流表溢出影响分析 18
3.2.1 OpenFlow 流表更新机制 18
3.2.2 OpenFlow 流表溢出的影响 20
3.3 流表溢出对 QoS 路由机制的影响 23
3.4 流表用量感知的 QoS 路由机制 25

3.4.1 流表用量感知路由模型 25
3.4.2 性能评估 30
3.5 小结 33
第 4 章 软件定义的 VANET 动态 QoS 路由机制 34
4.1 引言 34
4.2 软件定义的 VANET 架构 34
4.3 面向多业务多约束的动态 QoS 路由机制 36
4.3.1 流表用量感知的动态 QoS 保障框架 36
4.3.2 流表用量感知的多业务多约束 QoS 路由模型 40
4.4 性能评估 43
4.5 小结 47
结　论 49
参考文献 51
附录 A 攻读硕士学位期间发表的学术论文 56
附录 B 攻读硕士学位期间所参与的项目 57
致　谢 58

插图索引

图 2.1 SDN 网络架构 9
图 2.2 OpenFlow 系统架构 10
图 2.3 OpenFlow 包头域 11
图 2.4 Floodlight 控制器架构 15
图 2.5 Mininet 仿真实例 16
图 3.1 OpenFlow 流表溢出实验网络拓扑图 20
图 3.2 OpenFlow 流表溢出对控制器工作量的影响 21
图 3.3 OpenFlow 流表溢出对业务平均吞吐量的影响 23
图 3.4 OpenQoS 选路结果 23
图 3.5 OpenFlow 流表溢出对 QoS 路由机制工作量的影响 24
图 3.6 OpenFlow 流表溢出对业务吞吐量的影响 25
图 3.7 lp 格式的线性规划模型 28
图 3.8 路由矩阵示例 28
图 3.9 路由计算流程图 30
图 3.10 流表溢出场景下 FUQR 选路情况 31
图 3.11 流表溢出场景下 FUQR 与 OpenQoS 选路情况 31
图 3.12 FUQR 与 OpenQoS 的路由调整次数变化情况 32
图 3.13 FUQR 与 OpenQoS 对客户端吞吐量保障情况 32
图 4.1 软件定义的 VANET 架构 35
图 4.2 流表用量感知的动态 QoS 保障框架 36
图 4.3 链路可用宽带矩阵 37
图 4.4 5*5 邻接矩阵 37
图 4.5 5*5 带权矩阵 38
图 4.6 路由计算流程图 38
图 4.7 业务管理流程图 39
图 4.8 流表用量感知的动态 QoS 保障框架的工作流程 40
图 4.9 链路拥塞时启用和关闭 QoS 路由机制的选路情况 44
图 4.10 启动和关闭 QoS 路由机制时客户端吞吐量的变化情况 45
图 4.11 启动和关闭 QoS 路由机制时视频受影响情况 46
图 4.12 FTP 业务动态接入的场景 46
图 4.13 业务并发度与系统吞吐量变化的关系 47

附表索引

表 3.1 OpenFlow1.4 采用的 15 元组.....19
表 3.2 线性规划模型参数.....26
表 3.3 路由矩阵查找算法伪代码.....29
表 4.1 交换机端口连接关系表.....37
表 4.2 交换机流表使用情况表.....38
表 4.3 业务信息表.....39
表 4.4 线性规划模型参数.....41

第 1 章 绪论

1.1 研究背景

车辆自组织网络（Vehicular Ad Hoc Network，VANET）实现车与车（Vehicle-to-Vehicle，V2V）、车与基础设施（Vehicle-to-Infrastructure，V2I）、车与人及互联网的广泛通信，从而能够提供各类安全和非安全相关的服务。VANET 的应用场景非常广阔，具有重要研究价值，目前 VANET 主要应用在以下几个方面[1]：

(1) 安全驾驶。VANET 的安全报文传输能够为车辆提供即时的交通事故状态与道路拥堵信息，让车辆有充足的时间作出相应的措施，从而有效地避免危险行为与交通事故的发生，提高人们出行的安全性。

(2) 便利驾驶。车辆可以通过 VANET 获取道路状况信息，以便选择适合的行驶路径，此外，通过 VANET 还能查询停车场、加油站、修车站的地理位置，甚至可以获取停车场的空位数量，从而方便人们的出行。

(3) 车载娱乐。VANET 可以使得车辆之间共享多媒体信息，比如音乐、视频、游戏等等，丰富人们的驾驶生活。

(4) Internet 接入。VANET 可以通过路边的基础设施与 Internet 接入，以便车辆也能够接入 Internet。

虽然 VANET 能够支持许多服务和应用，但是在大规模地部署 VANET 服务和应用的时候仍然面临着许多挑战：

(1) 体系结构的挑战：当前的 VANET 架构主要是基站和路侧单元（Road Side Unit，RSU）的 V2V 与 V2I 混合架构，该架构缺乏灵活性，导致大规模地部署和设置新的服务或协议是很困难的。

(2) 通信流量的挑战：车辆的移动性使得网络通信流量的动态性和不均衡性特征突出，现有 VANET 缺乏全局的视图，在数据转发的过程中不能动态的调度网络资源来合理分配通信流量，从而提升网络传输效率。

(3) 网络服务质量（Quality of Service，QoS）的挑战：不同的 VANET 应用对于网络提供的 QoS 有着不同的要求。当前 VANET 主要依赖于尽力而为服务模式的传统 IP 网络，难以保障应用服务的 QoS。

软件定义网络（Software Defined Network，SDN）[2-3]以系统化的灵活控制网络的方式出现，其分离的数据与控制平面为网络带来了可编程性。传统计算机网络采用分布式控制，对业务需求进行修改时，会产生巨大的开销，同时对数据的控制和转发也都依赖于具体的网络设备，不便于网络维护。SDN 通过控制

平面来控制多个底层网络设备来实现路由计算和数据转发等功能，实现对网络资源灵活的按需调配，改善了传统网络维护的难度与灵活度。

当前 VANET 中的众多服务和应用主要依赖于传统 IP 分组网络，该网络的尽力而为服务模式不能为数据传输的带宽、端到端时延和包到达率等性能提供服务质量保证。因此，面临着大规模部署 VANET 服务和应用时候的诸多挑战，利用 SDN 转发与控制相分离的优势，将 SDN 的架构引入到 VANET 中来增强许多原有的 VANET 服务，例如能够保留或限制一些频段，让安全服务使用这些频段，或者让不同的流选择不同的频段来传输。软件定义的 VANET 架构充分利用了数据转发与控制相分离的优势，能够有效地解决在部署 VANET 应用服务时所遇到的问题。

1.2 研究现状

1.2.1 VANET QoS 路由研究现状

当前的 V2V 与 V2I 主要是以无线网络进行通信，也就带来了无线传输所存在的诸多问题，例如丢包率较大，时延较高，因此难以保障 VANET 应用服务的 QoS。为了有效保障 VANET 应用服务的 QoS 需求，国内外研究者对 VANET 中的 QoS 路由进行了相关的研究。

文献[4]提出的 QoS-OLSR 是一种解决 VANET 聚类问题的优化链路状态的路由协议，该协议能够兼顾 QoS 和车辆移动状态。文献[5]提出的 ViCoV 关注感知无线电网络 VANET 下的视频流传输，在完全和间歇连接的网络中广播安全和娱乐内容。文献[6]提出了 IPTV，即因特网协议电视，与需要高带宽和高质量 QoS 的 IPTV 业务的实时流传输相配合，可以通过 VANET 拓扑变化来区分通过 VANET 的 IPTV 业务流。文献[7]设计了一种应用于 VANET 的合作协议，该协议对多跳网络十分有效，能够获得较好的 QoS。文献[8]提出一种新型的适应性多媒体流分发系统，该系统使用本地高度缓存来保存 VANET 中的资源，避免了从 Internet 下载资源所产生的时延。文献[9]提出了一个综合分析模型，同时考虑到 EDCA 的 QoS 特征和车辆移动性， 基于该模型分析了差分服务流量的吞吐量性能和平均传输延迟，最佳地调整 EDCA 的参数为车辆提供可控的 QoS。文献[10]提出 MAC/PR，其 QoS 路由算法的核心思想是进行资源预留，这里的资源主要是指带宽。

上述对 VANET QoS 路由的相关研究主要依赖于尽力而为服务模式的传统 IP 网络，而 SDN 为 VANET 提供了一种新型的网络架构，有效地提高了网络的可编程性和灵活性，能够进一步保障 VANET 应用的 QoS。

许多研究者提出将软件定义网络架构引入 VANET，从而分离 VANET 数据平面与控制平面，实现网络的系统化灵活控制，带来可编程性。例如：文献[11]

提出了一种基于 SDN 的 VANET 架构，指出新架构将提高路径选择、频率/信道选择、功率选择方面的性能提升，并展望了新架构的 VANET 应用；文献[12]提出了基于 SDN 的 5G 网络 VANET 架构，并说明了 SDN 对下一代 5G 网络 VANET 的重要性；文献[13]提出了面向无线异构接入的软件定义的 VANET 架构，在该架构下，无线异构设备诸如汽车、RSU 都可以抽象为 SDN 交换机；文献[14]提出了 FSDN，一种基于 SDN 并支持雾计算的 VANET 架构，且该架构也支持无线异构接入。在软件定义 VANET 架构下可以考虑更多的网络接入方式，也能更有效地管理网络的资源，从而保障 VANET 应用服务的 QoS。

1.2.2 SDN 研究现状

SDN 的概念在提出时就受到了学术界和工业界的普遍关注和广泛研究[15]。SDN 的思想起源于斯坦福大学 Nick McKeown 教授等人的科研项目 Ethance，2007 年，Nick McKeown 教授团队提出 SDN 的概念，并与 Martin Casado 博士、Scott Shenker 教授等人共同创立了 Nicira 公司专注于网络虚拟化的发展。2008 年，Nick McKeown 教授在 SIGCOMM 发表文章《OpenFlow: Enabling Innovationin Campus Networks》[16]，在此之后，Nick McKeown 提出的 OpenFlow 技术获得两届 SIGCOMM 的最佳演示奖。至此，SDN 成为当前全球网络领域最热门的研究方向[17]。

2011 年，由 Google、Facebook 等多家公司倡议成立了开放网络基金会（Open Networking Foundation，ONF），ONF 是一个非营利性组织，其宗旨将 SDN 技术规范化、商业化。2013 年是 SDN 发展最迅猛的一年，有多篇有关 SDN 的学术研巧出现在 SIGCOMM 大会上。同时，工业界在 SDN 研究方向也投入大量的人力物力，比如思科、IBM、HP、BigSwitch、Juniper 等厂商成立了 OpenDaylight[18]开源项目，该项目以厂商为主致力于进行 SDN 的开源研发以及 SDN 控制器和交换机的研制。在互联网企业中，以 Google、Amazon、Yahoo、Microsoft、Facebook 为代表的国际互联网公司正在积极进行 SDN 技术的探索与研究，其中 Google 构建的 B4 网络[19]毫无疑问是最具代表性的 SDN 技术成功应用的范例，Google 的基于 OpenFlow 协议构建的 B4 网络可以实现自动化构建网络以及重新配置数据中心的连接的功能，B4 网络通过其动态的改变流量路径来保护高优先级的流量，从而将它的资源利用率由 30%提高到了 90%以上，在极大程度上节省了资源。

国内企业对 SDN 的探索与研究也取得了显著的成果。华为、盛科、华三、锐捷等企业都在该领域投入巨大的科研力量，很多互联网公司也把 SDN 当作一次革命性变革，积极投入力量[20]。2014 年，SDN 产业联盟在北京正式揭牌成立。同年，腾讯在 SDN/NFV 大会上提出了构建边缘智能网络的概念。2015 年，百度在 SDN/NFV 大会提出了其 SDN 解决方案 Baidu Unified Net 5.0，一

种基于 SDN 的统一网络架构。由工信部电信研究院联合业界 15 家单位协力创立的 SDN 产业联盟秉承“开放、创新、协同、落地”的宗旨，关注 SDN 商用价值，推动 SDN 产业发展。SDN 产业联盟由来自全球的 33 家成员组成，将定位于全球 SDN 产业的推动。33 家成员中包括中国三大运营商以及阿里巴巴、腾讯、奇虎 360、华为、中兴等知名企业。

1.2.3 基于 SDN 的 QoS 路由机制研究现状

传统的 QoS 模型不具备对网络信息的采集与分析功能，也无法支持动态实时更新配置信息，也就无法基于网络当前的情况动态地选择合适的路径，SDN 的出现有效地提高了网络的可编程性和灵活性，能够动态地收集网络当前的参数，实时响应业务需求，从而为应用服务提供动态的 QoS 保障。SDN 下的 QoS 路由机制的研究主要包括 QoS 保障框架与 QoS 路由算法的研究。QoS 保障框架是基于 SDN 控制器的北向接口来提供一种网络 QoS 管理框架，或者通过修改和添加 SDN 控制器的内置模块来提供 QoS 保障，QoS 保障框架的特点是能够实时监控网络的状态并合理地管理网络的资源，从而达到保障 QoS 的目的。QoS 路由算法则是 QoS 保障框架为业务分配合适路径所采用的选路算法，一旦 QoS 路由机制发现业务的 QoS 得不到保障，QoS 保障框架会根据 QoS 路由算法为业务调整满足其 QoS 的路径。

在 SDN 的架构下，为了给应用服务提供有效的 QoS 保障，许多研究人员进行了相关的理论和实践研究。其中，文献[21-28]提出了 QoS 路由算法，文献[29-33]提出了 QoS 保障框架。

文献[21]针对域内网络提出了面向 SDN 的可扩展路由和资源管理模型，SDN 控制器基于此模型实施路由计算，权限控制和资源管理。文献[22]提出了基于 SDN 的 QoS 管理系统，并提出了动态路由算法。该系统能够根据实时监控的网络数据和代价模型来生成最小代价的路径。文献[23, 24]都是关注视频业务的 QoS，其中文献[23]提出了基于 SDN/OpenFlow 的分析型架构来优化转发策略，文献[24]与[23]类似，但[24]定义了自己的受限最短路径问题，可以动态地调整路由策略。文献[25]专注于修改 Floodlight 控制器的内置模块来优化其对 QoS 的支持性能。由于 Floodlight 控制器采用的路由算法没有考虑链路之间的代价，因此作者在此基础上修改了路由算法，考虑了链路负载和其他 QoS 指标。文献[26]描述了 SDN 的网络架构并分析了提供多媒体服务的机遇和必要性，因此作者提出了一种 SDN 框架，该框架集成了 OpenFlow、网络虚拟化、功能盒以及测试算法的接口。其中的“QoS 路由算法”模块可以作为运行不同 QoS 路由算法的容器。文献[27]基于 SDN/OpenFlow 实施了一套 QoS 保障模型，可以实时地监控网络的 QoS 指标并动态地改变业务的路由。此外，作者还提出了面向多业务多约束条件下的路径模型，由此来为多业务分配满足其 QoS

需求的路径。文献[28]提出了一种基于 QoS 和动态负载均衡的路由策略，该算法兼顾了流的 QoS 需求，并用近似算法进行多 QoS 约束的最优路径选择。

文献[29]提出了 OpenQoS，一种 SDN/OpenFlow 控制器的设计方案，其目的是保障多媒体业务端到端传输的 QoS。多媒体流的传输路径是可以根据当前网络的 QoS 指标来进行动态调整的。文献[30]提出了 OpenNetMon 来监控网络参数，它作为 SDN 控制器内置模块添加到 POX 中去，不但可以监控每一条流的 QoS 指标，比如吞吐量，时延和丢包率，还能在线地判断端到端的 QoS 参数是否满足业务要求。文献[31]基于 SDN/OpenFlow 与 OpenNaaS 提出了网络控制层的概念。网络控制层包括 SDN 控制器，SDN 监控器和 SDNApp，网络控制层的目标是为服务提供端到端的动态 QoS 控制。文献[32]作者提出了 QoS 管理机制（AQSDN），能够让 QoS 指标可以自动地配置在 OpenFlow 交换机里。此外，作者基于 AQSDN 设计了数据包上下文感知的 QoS 模型（PCaQoS）来提高网络的 QoS，此模型在数据包被标记和进入转发队列管理状态时能够考虑数据包的上下文。文献[33]提出了 OpenFlow 增强的 QoE 保障框架，当多个用户竞争网络资源的时候最大化每个用户的 QoS，此框架关注的是用户级别的公平性和网络的稳定性。

综上所述，当前 SDN 下 QoS 路由机制的研究还存在以下几个方面的不足：

首先，由于 SDN 路由过程的实现不得不依赖于交换机的流表管理，当前研究者对交换机流表的管理机制进行了相关研究，但是流表的使用率对 QoS 路由机制的影响却未见具体分析。

其次，大多数 QoS 保障框架下的路由模型的研究重点都是受限最短路径问题，无法满足软件定义的 VANET 下多业务并发处理的需求。

最后，考虑了多业务并发的 QoS 模型的研究却没有考虑业务动态接入和退出，而软件定义的 VANET 下的业务随机接入和退出的动态性很强。

1.3 研究意义

传统的 QoS 路由主要依赖于尽力而为服务模式的 IP 网络，由于只具备网络局部信息的采集与分析功能，因此，无法根据网络全局的情况动态地选择合适的路径。SDN 的出现有效地提高了网络的可编程性和灵活性，能够动态地收集网络当前的参数，实时响应业务需求，从而为应用服务提供动态的 QoS 保障。

软件定义的 VANET 架构充分利用了数据转发与控制相分离的优势，而在软件定义的 VANET 架构下，VANET 提供了具有不同特点的应用服务，有的实时性比较高，但带宽占用量比较小，比如交通信息，新闻，天气等等；而有的属于带宽密集型服务，比如视频，语音通话等等。因此，如何保障软件定义 VANET 下应用服务的 QoS 是一个值得研究的问题。

1.4 本文主要工作

软件定义的 VANET 架构充分利用了数据转发与控制相分离的优势，而在软件定义的 VANET 架构下，VANET 提供了具有不同特点的应用服务，为了给应用服务提供 QoS 保障，许多研究人员进行了相关的理论和实践研究，但这些研究存在以下几点不足：其一，面向单业务，无法满足软件定义的 VANET 下多业务并发处理的需求；其二，缺乏对业务动态接入和退出的处理，而软件定义的 VANET 下的业务随机接入和退出的动态性很强；其三，不具有流表用量的感知能力，而 SDN 路由过程的实现不得不依赖于交换机的流表管理。本文针对以上不足，进行了如下研究工作：

(1) 总结了 VANET、SDN 以及 SDN 下 QoS 路由机制的研究现状，并分析了 SDN 下 QoS 路由机制相关研究的不足，发现了目前的 QoS 路由机制不仅缺乏对多业务并发处理的能力，还无法感知流表用量这一研究现状。

(2) 分析了 SDN 架构以及 OpenFlow 协议，同时还分析了主流的开源控制器 Floodlight 以及网络仿真平台 Mininet 的架构与使用。

(3) 分析了 OpenFlow 流表更新机制、流表溢出对控制器和 QoS 路由机制的影响。由于 SDN 路由过程的实现不得不依赖于交换机的流表管理，因此流表策略对 QoS 路由机制性能的影响至关重要。本文通过实验分析了 OpenFlow 协议流表溢出对控制器负载、业务平均吞吐量和 QoS 路由机制工作量的影响，并提出了流表用量感知路由，能够避免流表溢出带来的不良影响，极大地减轻了控制器的负载，同时有效地保障了网络应用的 QoS。

(4) 设计了一种面向异构多网接入的软件定义 VANET 架构，完善了现有工作中数据平面的设计，接着提出一种流表用量感知的动态 QoS 保障框架，该框架允许使用模块化的方式管理网络，并支持业务流的动态加入和退出，同时本文建立了多业务流多约束条件下流表用量感知的 QoS 路由模型，该模型不仅考虑了丢包、时延和吞吐量等链路状态参数，还考虑了应用服务的需求以及交换机的流表使用情况，为 VANET 应用服务提供并发的动态 QoS 路由。实验结果表明，流表用量感知的动态 QoS 路由机制不仅能够满足多业务对各自丢包、时延和吞吐量的要求，而且能够感知流表用量从而避免流表溢出对 QoS 路由的影响，进一步提高了网络 QoS 保障的性能。

1.5 论文结构

本文共由 5 个章节组成，全篇围绕软件定义的 VANET 下流表用量感知的动态 QoS 路由机制展开，论文各个章节的具体内容安排如下：

第一章，首先简要介绍了本文的研究背景与意义，然后分析了 VANET、SDN 以及 SDN 平台下 QoS 路由机制的研究现状，最后简述本文的主要研究工

作以及论文的组织结构。

第二章，分析了 SDN 架构及 OpenFlow 协议，同时还介绍了主流的 SDN 控制器 Floodlight 的架构与使用，以及网络仿真平台 Mininet。

第三章，分析了 OpenFlow 流表更新机制、流表溢出对控制器和 QoS 路由机制的影响，提出了流表用量感知路由并通过实验评估其流表感知性能。

第四章，设计了一种面向异构多网接入的软件定义 VANET 架构，在该构架下提出一种流表用量感知的动态 QoS 保障框架，允许使用模块化的方式管理网络，并支持业务流的动态加入和退出，同时建立了多业务流多约束条件下流表用量感知的 QoS 路由模型，为 VANET 应用服务提供并发的动态 QoS 路由。

第五章，结论和展望。对本文所做的工作进行了总结，对存在的问题和今后的研究工作做出展望。

第 2 章 相关技术与理论基础

2.1 引言

传统计算机网络采用分布式控制，对业务需求进行修改时，会产生巨大的开销，同时对数据的控制和转发也都依赖于具体的网络设备，不便于网络维护。SDN 是一个将网络控制平面和数据平面进行分离的网络架构，通过控制平面来控制多个底层网络设备以实现路由计算和数据转发等功能，并实现对网络资源灵活的按需调配。本文的工作围绕 SDN 网络，对 SDN 控制器的路由机制进行扩展。因此，本章首先介绍了 SDN 的基本架构以及数据转发的原理，然后介绍当前最为广泛使用的数控平面交互协议 OpenFlow，接着介绍一款主流的 SDN 控制器 Floodlight，主要包括 Floodlight 的架构和模块功能，最后介绍本文实验所采用的 SDN 仿真平台 Mininet。

2.2 SDN 架构简介

SDN 的思想起源于斯坦福大学 Nick McKeown 教授等人的科研项目 Ethance，在提出时就受到了学术界和工业界的普遍关注和广泛研究。发展至今，国内外知名企业包括 Facebook、Yahoo、Google 等互联网公司以及德国电信、华为、中兴等网络设备商都对 SDN 展开了积极的研究。SDN 以系统化的灵活控制网络的方式出现，其分离的数据与控制平面为网络带来了可编程性。

传统 IT 架构中的网络是分布式控制，如果业务需求发生改变则需要重新修改相应网络设备，例如修改路由器上的信息，这就为网络的维护带来了巨大的开销，尤其是在业务需求变化量大的场景中。

SDN 架构将网络的控制平面与数据转发平面进行分离，从而通过集中的控制器中的软件平台可编程化地去控制底层设备，实现对网络资源灵活的按需调配。此外，传统 IP 网络中对数据的控制和转发都依赖于网络设备的实现，且设备中集成了与业务特性紧耦合的操作系统和专用硬件，这些操作系统和专用硬件都是各个厂家自己开发和设计的，而在 SDN 网络中，网络设备只需要负责数据的转发，不需要关心具体的控制逻辑是如何实现的，因此可以采用通用的硬件，负责对不同业务特性进行适配。

图 2.1 描述了 SDN 架构的逻辑视图，SDN 的典型架构共分三层，最底层的基础设施层负责数据的转发和处理以及状态收集；中间的控制层主要负责管理网络资源，对业务逻辑进行控制；最上层为应用层，主要是不同特性的应用。

SDN 三层的架构的具体介绍如下：

(1) 基础设施层。基础设施层在位于 SDN 架构的最底层，也可以称为数据平面，由南向接口和网络单元组成，每一个网络单元都能提供网络流量，是一个逻辑上的网络设备，展示了对转发和数据处理能力的控制能力和网络设备的可视化。南向接口是定义在数据平面与控制平面之间接口以便两个平面之间的信息交互，当前最为广泛使用的南向接口是 OpenFlow，它能够对转发操作提供程序化的控制、静态数据汇报和事件通知。

(2) 控制层。控制层位于 SDN 架构的中间层，也即 SDN 控制器所在的层次，控制器是 SDN 网络的核心组件，负责合理地管理网络的资源，能够转换应用程序的网络需求，在 SDN 数据平面使用低级别的网络控制，同时给 SDN 应用程序提供相关信息反馈。网络设备通过南向接口接受控制层发来的指令，产生转发表项，并可以通过南向接口主动将一些实时事件上报给控制层。该层内的 SDN 控制器可能有一个，也可能有多个，可能是一个厂家的控制器，也可能是多个厂家的控制器协同工作。

(3) 应用层。应用层在 SDN 架构的最顶层，SDN 应用程序通过北向接口与控制层通信，比如 REST API[34]就是目前广泛使用的北向接口协议规范，通过 REST API 向控制器发送应用控制请求，然后由控制器代理完成，但是应用程序对控制器的请求不是没有限制的，控制器只为应用程序提供用户级别的网络操作，以保证网络的安全。

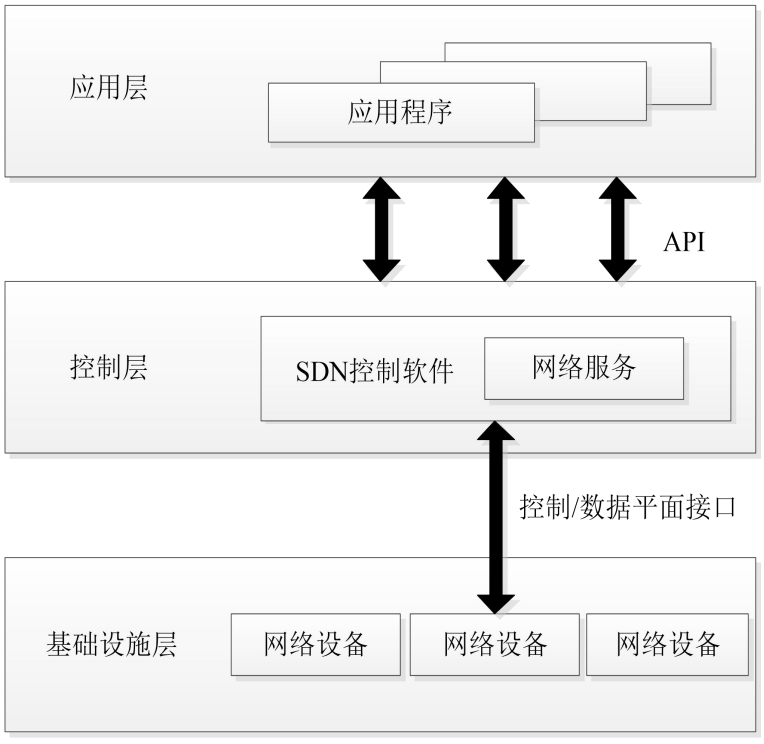


图 2.1 SDN 网络架构

综上所述，与传统网络相比，SDN 的特点与优势有以下三点：

(1) 数控分离。数据平面与控制平面分离，数据平面由通用网络设备组成，有利于解除专用网络设备的限制，这些网络设备的转发方式以及业务逻辑都由运行在控制平面上的控制逻辑所控制。

(2) 控制平面与数据平面之间的开放接口。SDN 为控制平面提供开放可编程接口，通过这种方式，控制应用只需要关注自身逻辑，而不需要关注底层更多的实现细节。

(3) 逻辑上的集中控制。与传统网络分布式控制相反，SDN 的集中控制也就是控制整个物理网络，因而可以获得全局的网络状态视图，并根据该全局网络状态视图实现对网络的优化控制。

2.3 OpenFlow 技术概述

OpenFlow 是目前广泛使用于 SDN 架构中的开放标准协议，OpenFlow 协议定义了 SDN 网络架构中控制平面与数据平面进行信息交互的标准，也叫做南向接口协议。OpenFlow 协议定义了一系列的数据结构，用于描述协议的动作和内容。自 2009 年底发布第一个正式版本 v1.0 以来，OpenFlow 协议已经经历了 1.0、1.1、1.2、1.3 和 1.4 等多个正式商用版本，而在当前应用中以 OpenFlow1.0[35]和 OpenFlow1.3[36]两个版本为主。随着 OpenFlow 协议的发展，网络的数据平面和控制平面的功能都在逐渐增强。一个典型的 OpenFlow 系统中至少包括两个组成部分，分别是 OpenFlow 交换机与 OpenFlow 控制器，如图 2.2 所示：

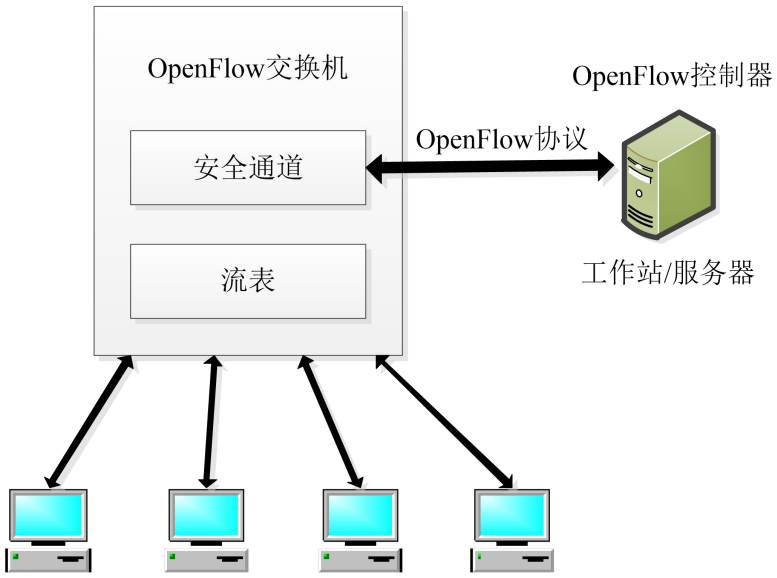


图 2.2 OpenFlow 系统架构

从图 2.2 中可以看出，OpenFlow 控制器通常为运行在 PC 机器上的程序，OpenFlow 控制器通过安全通道与交换机通信，安全通道使用 SSL 协议，控制器

的主要功能包括：维护网络拓扑，并基于网络拓扑对外提供流的相关操作，比如转发行为决策。 应用程序则通过特定的 API 与 OpenFlow 控制器交互，基于控制器提供的 API，可以开发具有特定功能的应用程序，这些应用程序根据特定的策略为每个网络流制定每个 OpenFlow 交换机的转发规则，并通过控制器下发给 OpenFlow 交换机，从而完成对网络流转发路径的决策。

2.3.1 OpenFlow 交换机

OpenFlow 交换机作为 OpenFlow 网络数据平面的核心组件，主要负责数据平面的数据转发功能。按照对 OpenFlow 的支持程度，OpenFlow 交换机可以分为两种：一种是专用 OpenFlow 交换机，仅仅支持 OpenFlow 协议的转发模式，不支持当前交换机的处理流程；另一种是支持 OpenFlow 转发的交换机，在传统交换机中添加具有 OpenFlow 协议功能的模块，使其具备 OpenFlow 交换机转发的要求，主要是为了实现传统交换机向 OpenFlow 交换机的过渡。

OpenFlow 协议具有开放性、灵活性等特点，许多主流设备厂商都已经推出了各自的 OpenFlow 交换机，比如 Broadcom、Dell、HP、IBM 等等。同时，OpenFlow 软交换机和调试软件也在渐渐增多，比如 OpenVSwitch 交换机、OpenFlow 调试工具 liboftrace 等等。

OpenFlow 交换机主要由流表（Flow Table）、安全信道（Secure Channel）和 OpenFlow 协议(OpenFlowProtocol)三部分组成。

(1) 流表。流表是 OpenFlow 对网络设备的数据转发功能的一种抽象，OpenFlow 交换机的转发控制的处理单元都是基于流表。流表是由多个流表项组成的，每个流表项记录了数据包的转发规则。流表项包括了包头域（header fields）、计数器（counters）和动作集（actions）。

- 包头域。 图 2.3 中所示的包头域是一个多域包，包含多个匹配项，涵盖了数据链路层、网络层和传输层的大部分标识。因此，在 OpenFlow 的网络下，统一用 OpenFlow 交换机来表示，网络设备不再需要区分路由器和交换机，每一个进入交换机的包将不再关注它在哪一层传输，而是根据包头域的正确匹配来安排转发。

图 2.3 OpenFlow 包头域

- 计数器。计数器主要用于统计数据流量相关的信息，例如针对每张流

表，统计当前活动的表项数、数据包查询次数、数据包匹配次数等；针对每个数据流，统计接收到的数据包数、字节数、数据流持续时间等；针对每个设备端口，除统计接收到的数据包数、发送数据包数、接收字节数、发送字节数等指标之外，还可以对各种错误发生的次数进行统计。可以通过计数器提供的信息来分析包匹配策略的效率、链路的利用率等，以此作为控制器路由决策的部分依据。

- 动作集。动作集表明了与该流表项匹配的数据包应该执行的下一步动作。与传统交换机转发表只需要指明数据包的转发出端口不同，OpenFlow 交换机因为缺少控制平面的能力，所以对匹配数据包的处理不仅仅是简单的转发操作，而需要用动作来详细说明交换机将要对数据包所做的处理。每个流表项可以对应有零至多个动作，如果没有定义转发动作，那么与流表项包头域匹配的数据包将被默认丢弃。不同的动作有不同的优先级，转发将按照动作的优先级顺序执行，但是动作的执行并不保证包的有序性。

(2) 安全通道。安全通道是连接 OpenFlow 交换机和控制器的接口，通过 SSL 协议控制器可以配置和管理交换机，同时还可以接收来自于交换机的事件和向交换机发出数据包等操作。

(3) OpenFlow 协议。OpenFlow 协议是 OpenFlow 交换机与 OpenFlow 控制器之间的信息交互协议。下一节将会对 OpenFlow 协议做详细说明。

2.3.2 OpenFlow 协议

OpenFlow 协议定义了 SDN 网络架构中控制平面与数据平面进行信息交互的标准，也叫做南向接口协议。OpenFlow 协议不仅规定了交换机和控制器之间的通信的消息类型和规格，还定义了交换机的转发能力。

SDN 控制器与 OpenFlow 交换机通过安全通道建立连接，通道里传送的都是控制协议消息，因此通道里的所有流量转发都无需查询交换机中的流表。当控制器与交换机之间的连接建立起来之后，双方首先要发送携带本方支持的最高协议版本号的 HELLO 消息给对方，接收方将采用双方都支持的最低协议版本进行通信，如果双方发现所支持的最低协议版本不匹配，则会发送 ERROR 消息，并终止连接。当交换机与控制器之间的连接发生异常时，OpenFlow 交换机应尝试连接备份控制器，当多次尝试均失败后，OpenFlow 交换机将进入紧急模式，并重置所有的 TCP 连接。

OpenFlow 协议支持三种类型的消息，分别为 Controller-to-Switch、Asynchronous 和 Symmetric。

(1) Controller-to-Switch。控制器与交换机之间的消息是由控制器发起，对 OpenFlow 交换机进行状态查询和修改配置等操作，OpenFlow 交换机接收并处

理可能发送或不需要发送的应答消息。在 OpenFlow 安全通道刚建立的时候，控制器会向交换机发出一个名为“features”的消息请求，交换机需要应答自身支持的功能。控制器可以通过“configuration”消息设置或查询交换机上的配置参数，交换机仅需要应答查询消息；当控制器需要对交换机的流表做修改时，可以使用“modify-State”消息，同时该消息还能设置交换机的端口性能；通过“read-state”消息向交换机请求诸如流表、端口、各个流表项等方面的统计信息；“Send-packet”消息通过交换机指定端口发出数据包；控制器通过“barrier”请求及相应报文，确认相关消息已经被满足或收到完成操作通知。

(2) Asynchronous。异步消息由 OpenFlow 交换机发起，用来通知交换机上发生的某些异步事件，消息是单向的，不需要控制器应答。主要用于交换机向控制器通知收到报文、状态变化及出席错误等事件信息。当一条流的第一个分组达到交换机时，如果没有匹配的流表项，交换机会向控制器发出“packet-in”消息。如果交换机有足够的缓存，交换机会临时将包头域的部分片段保存在缓存空间，控制器通常会通过“packet-out”消息来处理缓存的网包，或者缓存的网包会在一定时间后因过期而被丢弃。如果交换机缓存不足，那么交换机会将整个包发送至控制器，让控制器决定该如何处理这个包。OpenFlow 交换机中的流表项因为超时或收到修改/删除命令等原因被删除掉，会触发“flow-removed”消息。OpenFlow 交换机端口状态发生变化时，触发“Port-status”消息。OpenFlow 交换机通过“Error”消息通知控制器发生的问题。

(3) Symmetric。对称消息不必通过请求建立，控制器和交换机都可以主动发起，并需要接收方应答。这些都是双向对称的消息，主要用来建立连接、检测对方是否在线等。包括“hello”、“echo”和“Vendor”。“hello”消息用于在 OpenFlow 交换机和控制器之间发起连接建立；“echo”消息用来协商延迟、带宽、是否连接保持等控制器到 OpenFlow 交换机之间隧道的连接参数；“Vendor”消息作为 OpenFlow 交换机的附加消息功能，主要是为了未来 OpenFlow 版本的修订而预留的消息。

2.3.3 OpenFlow 控制器

传统计算机网络采用分布式控制，对业务需求进行修改时，会产生巨大的开销，同时对数据的控制和转发也都依赖于具体的网络设备，不便于网络维护。网络操作系统（Network Operating System，NOS）实现了对网络设备的抽象，用户不再关心具体的网络设备，而是通过 NOS 与抽象设备进行交互。OpenFlow 控制器实际上就扮演了 NOS 的角色，通过 OpenFlow 控制器用户可以用自己熟悉的语言或者工具对网络设备或者网络状态进行管理。

NOX[37]是第一个 OpenFlow 控制器，随后 NEC、Big Switch、Beacon 等厂商也推出了自己的 OpenFlow 控制器。主流的控制器主要有 NOX、POX[38]、

Beacon[39]、Floodlight[40]、Maestro[41]、Helios[42]和 Ryu[43]等等。

(1) NOX。NOX 是斯坦福大学在 2008 年提出的第一款 OpenFlow 控制器，它的早期版本（NOX-Classic）由 C++和 Python 两种语言实现，只能支持单线程操作。NOX 的新版本完全由 C++实现，支持 OpenFlow1.0 协议，并且提供了多线程的支持。最新版本的 NOX 只有 switch 一个应用，实现了 learning switch 的功能。

(2) POX。NOX 团队从其旧版本中分离出 Python 语言实现的内容之后，又实现了一款完全使用 Python 语言的控制器 POX。POX 得以快速发展，并且得到了广泛的应用。

(3) Beacon。Beacon 是一款基于 Java 语言的开源控制器。Beacon 还具有很好的跨平台特性，并且支持多线程，可以通过 Web 的 UI 进行访问控制。Beacon 采用 Java 的 Spring 和 Equinox 编程模型，使用者可以通过用户界面动态地进行模块的添加和删除，在使用和部署上很方便。

(4) Floodlight。Floodlight 采用 Java 语言实现，在 Apache 开源标准许可下可免费使用。Floodlight 的稳定性、易用性已经得到 SDN 专业人士以及爱好者们的一致好评，并因其完全开源，这让 SDN 网络世界变得更加有活力。

(5) Maestro。Maestro 是莱斯大学 2011 年的一篇论文，提出并用 Java 实现了一款基于 LGPLv2.1 开源协议标准的多线程控制器，应用于科研领域。

(6) Helios。Helios 是闭源的 SDN 控制器。Helios 是由 NEC 公司开发的基于 C 语言的可扩展控制器，它主要应用于科研环境，并且提供了一个可编程的界面来进行实验。

(7) Ryu。Ryu 是由日本 NTT 公司负责设计研发的一款开源 SDN 控制器。同 POX 一样，Ryu 也是完全由 Python 语言实现，使用者可以用 Python 语言在其上实现自己的应用。Ryu 目前支持 OpenFlow1.0、1.2 和 1.3，同时支持在 OpenStack 上的部署应用。

2.4 Floodlight 控制器

Floodlight 采用 Java 语言实现，在 Apache 开源标准许可下可免费使用，Floodlight 不仅是基于 OpenFlow 协议的 SDN 控制器，也是一个 Floodlight 控制器的应用程序集。Floodlight 体系结构如图 2.4 所示。

Floodlight 由控制器模块和应用模块组成，控制器模块实现了核心的网络服务并为应用程序提供接口，应用模块根据不同的目的实现不同的方案。

(1) 控制模块。控制器运行所需要的模块：

FloodlightProvider：处理 SDN 控制器和交换机的连接及 OpenFlow 消息分发机制的管控。

设备管理模块：管理网络中的主机等终端设备。

链路发现模块：发现和维护 OpenFlow 网络中的链路状态。

拓扑服务模块：维护拓扑信息。

REST 服务模块：提供 REST API 服务。

静态流创建模块：支持静态流的增加和删除。

内存存储源模块：提供数据存储以及变更服务。

流高速缓存：网络中需要处理某个范围内不同的事件，就定义了流缓存。

数据包流服务：该模块是一个数据包流服务，使用此项服务可以让任何交换机、控制器和它的观察者之间有选择的交换数据。

(2) 应用模块。研究者和用户根据需要添加的模块：

虚拟网络过滤器：该模块是二层上一个简单的虚拟化网络，可以使用它在一个二层的域中建立多个逻辑的二层网络。该模块可以独立使用也可以在 OpenStack 上使用。

转发模块：路由决策以及流表下发。

防火墙模块：防火墙应用程序实现了一个 floodlight 模块，该模块通过检测 PacketIN 行为使用流在 OpenFlow 使能的交换机上强制执行 ACL。ACL 是一系列的条件，这些条件允许或者拒绝接入交换机上的流量。

端口关闭处理模块：该模块为了在端口关闭的时候处理网络中的流。

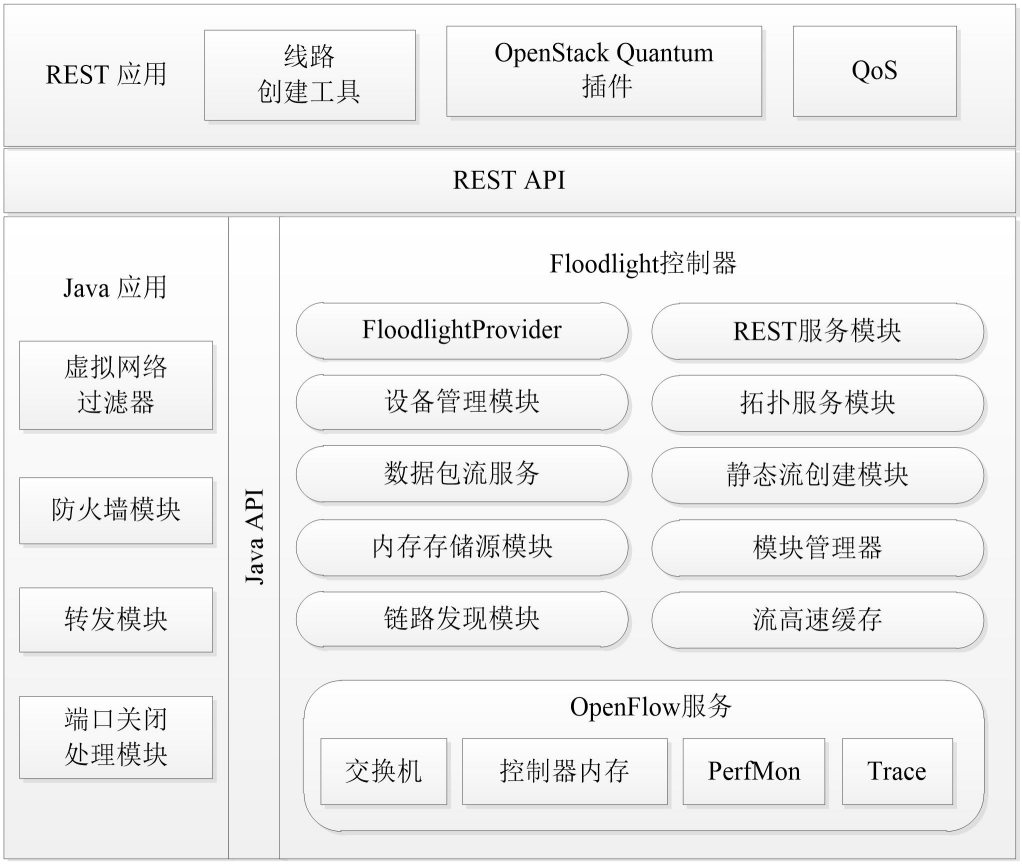


图 2.4 Floodlight 控制器架构

2.5 Mininet 仿真器

Mininet[44] 是支持 OpenFlow 协议的轻量级的软件定义网络系统仿真平台，由 Stanford 大学 Nick McKeown 的研究小组基于 Linux Container 架构开发出的一套进程虚拟化的平台。

通过 Mininet 可以轻易地在自己的笔记本电脑上测试一个软件定义网络，对基于 OpenFlow、OpenvSwitch 的各种协议等进行开发验证，或者验证自己的想法。Mininet 很大的一个优势在于所有的代码几乎可以无缝迁移到真实的硬件环境中，学术界跟产业界再也不是那么难以沟通了，在实验室里，一行命令就可以创建一个支持 SDN 的任意拓扑的网络结构，并可以灵活的进行相关测试，验证了设计的正确后又可以轻松部署到真实的硬件环境中。

Mininet 中自带交换机、主机、控制器，同时在 Mininet 上还可以安装 OpenvSwitch 以及多种控制器（NOX\POX\RYU\Floodlight\OpenDaylight 等等），同时，Mininet 有很强的系统兼容性，可以在当前主流的各种操作系统上运行，例如 windows、linux 以及 Mac OS 等等。图 2.5 为 Mininet 仿真 SDN 网络的简单拓扑实例，c0 为控制器，可以是 Mininet 内置的控制器，也可以是外接控制器，c0 通过 6633 端口与 OpenFlow 交换机 s1 通信，s1 与 h1、h2、h3 三台主机相连，并设置好主机的 IP 地址。

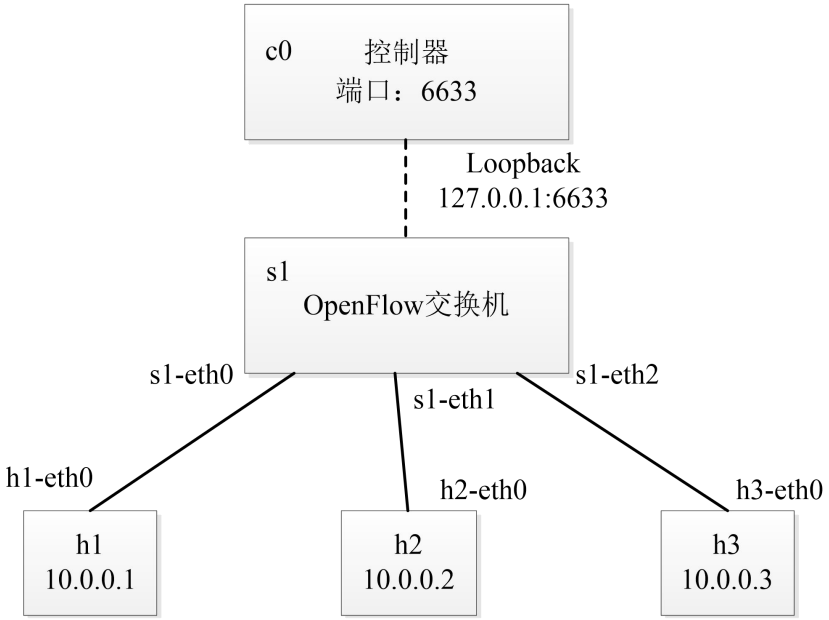


图 2.5 Mininet 仿真实例

2.6 小结

本章首先介绍了 SDN 的三层架构以及数据转发的原理，然后介绍当前最为广泛使用的数控平面交互协议 OpenFlow，包括 OpenFlow 交换机、OpenFlow

控制器和 OpenFlow 协议，接着介绍一款主流的 SDN 控制器 Floodlight，主要包括 Floodlight 的架构和模块功能，最后介绍一款网络仿真器 Mininet，描述了通过 Mininet 如何建立一个 SDN 网络。

第 3 章 流表溢出对 QoS 路由机制的影响

3.1 引言

软件定义网络的提出有效地提高了网络应用的性能，但有限的流表资源带来了新的问题。由于流表使用率的不均衡，给原本容量就十分有限的交换机带来了资源耗尽的压力，最终会导致瓶颈交换机产生流表溢出的现象。SDN 路由过程的实现依赖于交换机的流表管理，因此，流表溢出对 QoS 路由机制性能的有重要影响。本章首先分析了基于 OpenFlow 协议的 SDN 网络中流表溢出的现象及原因，接着通过仿真实验分析了流表溢出的影响，不仅包括流表溢出对 SDN 控制器的工作量以及业务平均吞吐量的影响，还包括流表溢出对 QoS 路由机制的工作量以及业务平均吞吐量的影响，由此说明了 OpenFlow 协议的流表溢出对 QoS 路由机制的性能有至关重要的影响。而后，为了避免流表溢出对 QoS 路由机制的影响，本章提出了一种流表用量感知的路由机制，能够让 QoS 路由机制在路由的过程中感知流表状态。实验结果表明，流表用量感知路由能避开瓶颈交换机，极大地减轻了控制器的负载，同时有效地保障了网络应用的 QoS。

3.2 基于 OpenFlow 的流表溢出影响分析

3.2.1 OpenFlow 流表更新机制

OpenFlow 是一种目前被广泛使用的南向接口协议，实现控制器与数据平面的信息交互。在传统的 IP 网络中，路由表用于保存路由信息，路由器对于到达的分组是通过查找路由表来转发的，而查找路由表实际上就是查找与分组相匹配的路由表项，匹配的内容主要包括源 IP 地址、目的 IP 地址、源端口号、目的端口号和 IP 协议，因此 IP 网络中采用 5 元组来匹配分组。OpenFlow 交换机采用流表来进行流处理，通过查找流表来转发分组，但是与 IP 网络不同，OpenFlow 流表一般采用 12 元组来匹配分组，而 OpenFlow1.3.0 版本增加了 3 个元组，一个流表项最多采用了 15 元组，表 3.1 所示的是 OpenFlow1.4 中采用的 15 元组，虽然增加了流操作的灵活性，但无疑也增加了对流表容量的需求。OpenFlow 交换机流表主要采用 TCAM（Ternary Content Addressable Memory）来实现分组匹配，但是网络设备生产商考虑到 TCAM 高功耗和高成本的问题，从而对 OpenFlow 交换机的流表容量作了限制，比如 NEC PF5820 只支持 750 条 12 元组的流表项，HP 5406zl 只支持 1500 条 12 元组的流表项。

通常在一个 SDN 网络中都会存在流表利用率不均衡的现象。比如在一个最

短路径优先的路由环境中，同一类型业务的所有请求很可能都会选择同一条最短路径，这就造成该路径上的流表利用率远远高于其他交换机流表。因此，当业务量比较多的时候，流表溢出现象就可能会发生，但控制器依然按照最短路径为新到达的业务选路。

表 3.1 OpenFlow1.4 采用的 15 元组

元组	标示	比特数
输入端口	Ingr	32
Metadata	Metadata	64
源以太网地址	Eth_src	48
目的以太网地址	Eth_dst	48
以太网类型	Eth_type	16
VLAN ID	VID	12
VLAN 优先级	Vprty	3
MPLS 标签	MPLS_lbl	20
MPLS 流量类	MPLS_tfc	3
源 IP 地址	SA	32
目的 IP 地址	DA	32
IP 协议	Ptrl	8
IP 服务类型	ToS	6
源端口号	SP	16
目的端口号	DP	16

OpenFlow 交换机的流表用于保存转发的相关信息，一条流表项代表了一条转发规则，但是实现流表转发的 TCAM 是高功耗和高成本的，并且随着协议版本的升级流表项的元组也在增加，因此流表的容量是有限的，也就意味着在大量业务的场景下，OpenFlow 交换机很有可能存在流表资源不足的问题。

除了 TCAM 所带来的流表容量限制之外，OpenFlow 流表更新机制也可以成为流表溢出的原因。每条流表项都可能超时，timeout 就是表示该流的超时时间。OpenFlow 定义了两种超时时间，hard-timeout 和 idle-timeout。流表项会在以下三种情况被删除：

(1) Hard-timeout：从流表项创建开始，经过 timeout 之后，无条件删除，不论 timeout 内有无任何的分组匹配过该流表项。

(2) Idle-timeout：经过了 timeout 后如果还没有任何的分组匹配该流表项，则该流表项会被删除。

(3) Command：控制器通过 delete 消息删除流表项。

因此，在 OpenFlow 中 hard-timeout 通常用来删除超时流表项（一般超过

比较长的时间，比如 1 分钟），而 idle-timeout 则是按秒来设置，但是这也比短业务流的生命周期要长[45]，所以流表资源很容易就会被用完，如果有新的业务流到达，由于这种超时机制的设置，一部分本该被删除的业务流依然驻存在流表中，消耗了宝贵的流表资源，降低了流表的利用率[46]，最终会导致流表溢出。

3.2.2 OpenFlow 流表溢出的影响

通过前面的分析可以知道 OpenFlow 流表利用率的不均衡、OpenFlow 流表更新机制都可能成为流表溢出的原因。本节将通过仿真实验分别分析流表溢出对控制器工作量以及业务平均吞吐量的影响。

如图 3.1 所示，通过 Mininet 生成一个 SDN 网络，网络中共有 12 台交换机，设定链路带宽为 30Mbps，丢包率 1%，时延 10ms，接着设置交换机 *S*8 的流表容量为 10，即当第 11 条流到达网络中的时候就会发生流表溢出。SDN 控制器采用目前较为主流的开源控制器 Floodlight，服务器通过交换机 *S*8 接入网络，以便交换机 *S*8 会发生流表溢出，客户端通过交换机 *S*1 接入网络。随后，每隔 2s 向网络中添加一条业务流，业务向服务器请求服务，通过服务器配置每条业务流的传输速率为 1Mbps。

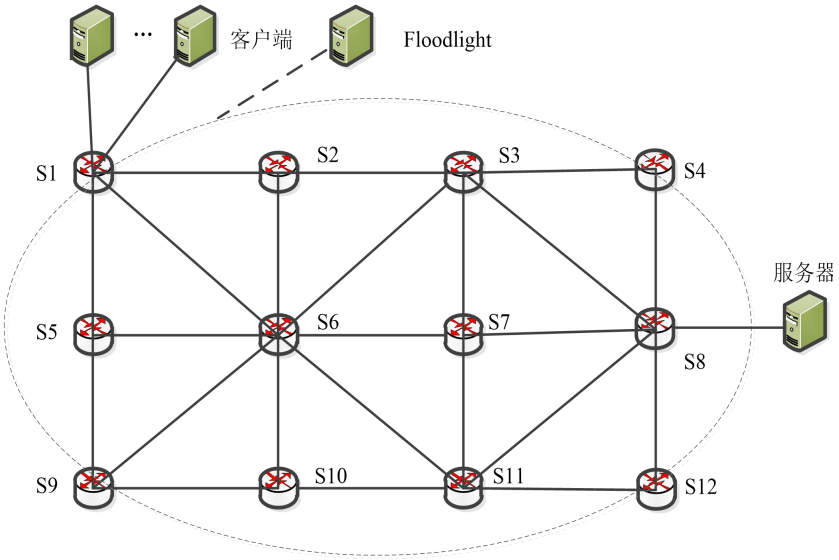


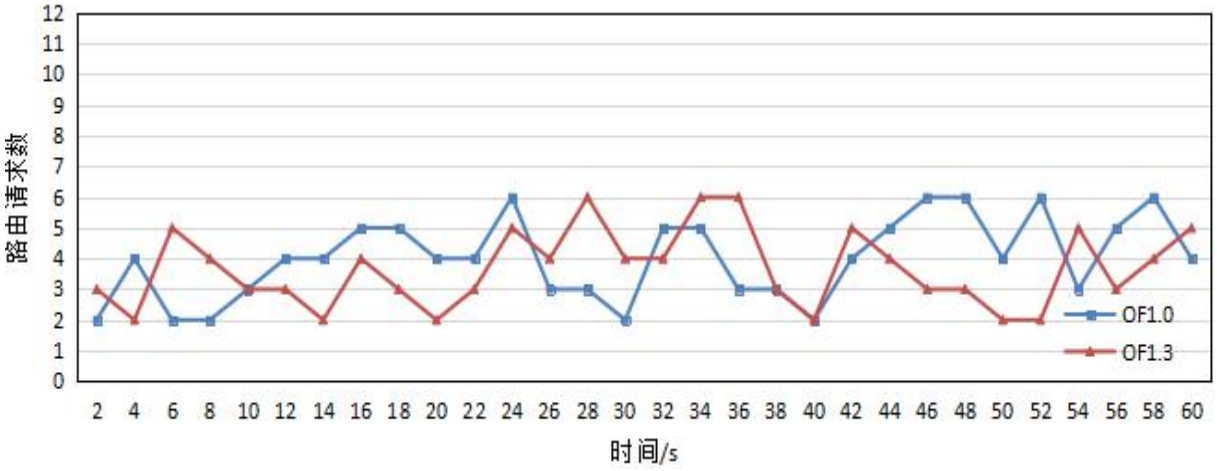
图 3.1 OpenFlow 流表溢出实验网络拓扑图

由于目前广泛使用的 OpenFlow 协议版本分别 OpenFlow1.0、1.3，其中 OpenFlow1.0 是 OpenFlow 的经典版本，也是实验室内使用最为广泛的版本之一，而 OpenFlow1.3 是当前厂商广泛支持的协议版本，因此本节将采用这两个协议版本分别进行实验。实验数据均通过 10 次实验求均值得到。

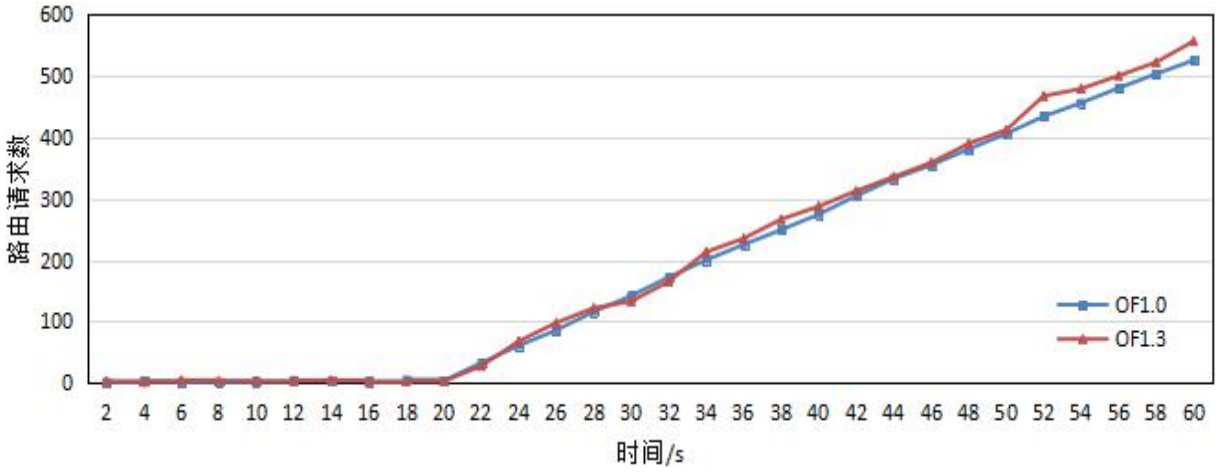
(1) 流表溢出对控制器工作量的影响

首先通过仿真实验来分析流表溢出对控制器工作量的影响，控制器工作量定义为路由请求次数，即当一个新的业务流的第一个分组到达交换机时会引发控

制器产生的路由请求次数。本次实验分为两个部分，在仿真实验(a)中最多会有30条流在同时传输，设置*S*8的流表容量为30，因此不会发生流表溢出；在仿真实验(b)中最多会有30条流在同时传输，但是限制*S*8的流表容量为10，会发生流表溢出。图3.2所示的是流表溢出对控制器工作量的影响。



（a）流表未溢出时控制器的工作量



（b）流表溢出时控制器的工作量

图 3.2 OpenFlow 流表溢出对控制器工作量的影响

通过图3.2(a)可以看到，随着新的业务流不断地加入，OpenFlow1.0与OpenFlow1.3版本下的路由请求数量基本保持在10次以内，原因是只有当一个新的业务流的第一个分组达到交换机时才会产生路由请求，而路由请求数量也受到当前网络状况的影响，但是并不会产生数量过大的请求，但请求的数量也不是固定的，同时也发现了不同协议版本对路由请求数量没有影响，即随着协议复杂度的增加，并没有使路由过程中交换机与控制器的数据交互过程变得复杂。相反，通过图3.2(b)可以看到，在流表溢出前，虽然OpenFlow1.0与OpenFlow1.3版本下的路由请求数量基本保持在10次以内，但是在20s的时候发生流表溢出，两个协议版本下的路由请求次数以平均27次/秒的速度增加，这是因为当流表溢出后，OpenFlow1.0和OpenFlow1.3都对新加入的流的分组采

取了丢弃措施，导致新的流不断向控制器发起路由请求，但控制器下发的流表信息仍然被交换机丢弃，因此造成了路由请求迅速地增加。此外，在流表溢出后新加入的流越多，对控制器造成的工作负载也会越来越重，正如图 3.2(b)中所体现的，从 20s 开始发生流表溢出，每隔 2s 的路由请求都比前 2s 的路由请求增加接近 2 倍的数量。

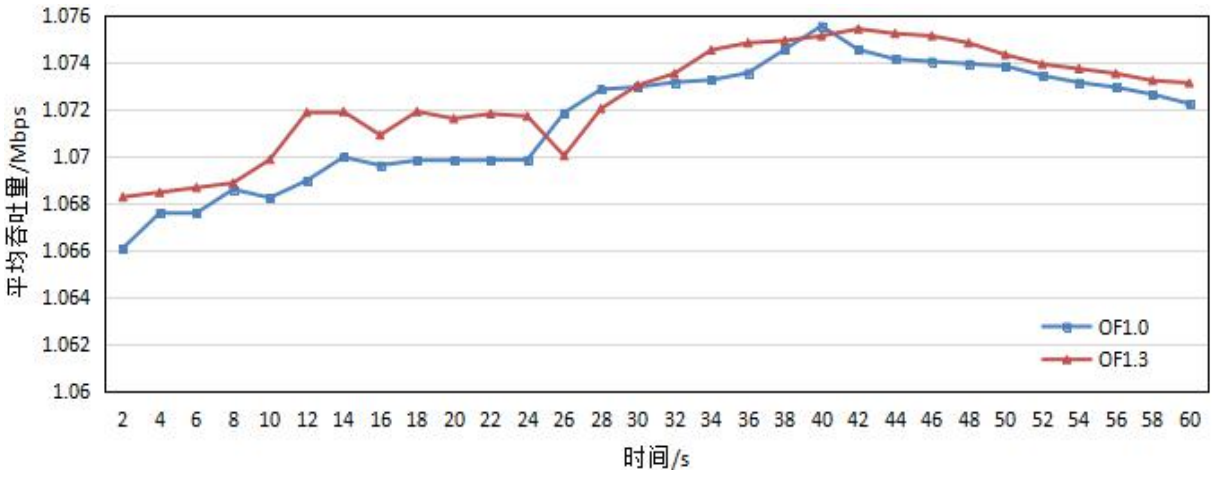
当流表溢出的情况出现时，OpenFlow 所采取的措施因版本而不同，OpenFlow1.0 至 1.3 的流表更新机制是拒绝新到达的业务，而 OpenFlow1.4 可以自定义流表项的丢弃规则，比如可以选择重要性更低的业务流表项进行丢弃，然而这同样也会导致正在传输的流重新对控制器发起路由请求，控制器下发新的流表又会替换掉重要性更低的流表项，同样会造成仿真实验(b)中所描述的情况，并不能够避免流表溢出对控制器工作量带来的不良影响。

因此，本实验说明了流表溢出会增加控制器的工作负担。

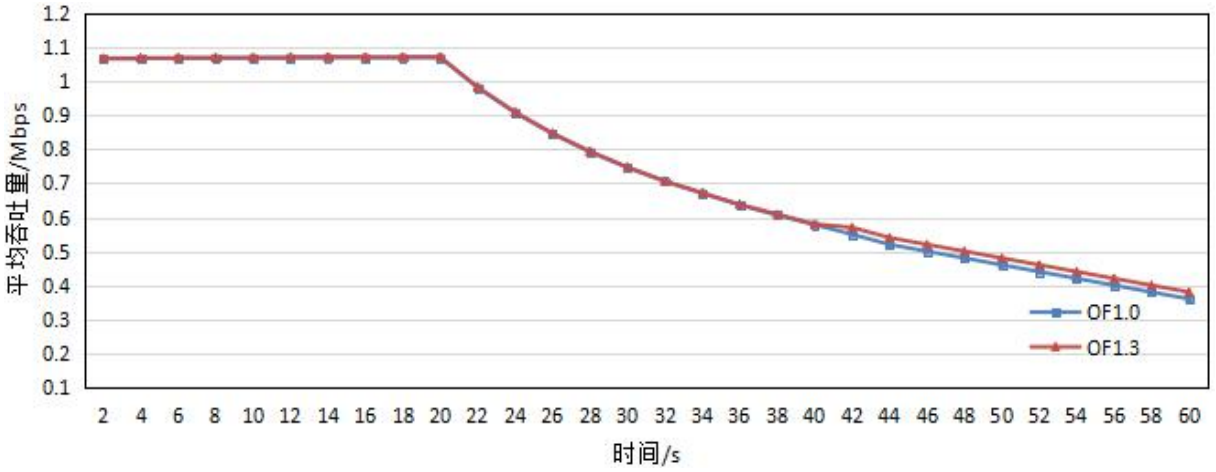
(2) 流表溢出对业务平均吞吐量的影响

接着通过仿真实验来分析流表溢出对业务平均吞吐量的影响。由(1)中的实验可知当流表溢出后，新加入的流的路径无法被添加到交换机中，因此，会影响业务的吞吐量。本次实验分为两个部分，在仿真实验(a)中最多会有 30 条流在同时传输，设置 *S*8 的流表容量为 30，因此不会发生流表溢出；在仿真实验(b)中最多会有 30 条流在同时传输，但是限制 *S*8 的流表容量为 10，会发生流表溢出。图 3.3 所示的是流表溢出对业务平均吞吐量的影响。通过图 3.3(a)可以看到，随着新的业务流不断地加入，OpenFlow1.0 与 OpenFlow1.3 版本下的业务平均吞吐量呈现逐渐上升的趋势，直到系统最大值，然后逐渐下降，能够满足业务正常的传输带宽 1Mbps。相反，图 3.3(b)中在 20s 时发生了流表溢出，新加入的流的路径无法被添加到交换机中，导致了业务平均吞吐量呈现逐渐下降的趋势，远达不到 1Mbps 的需求。

因此，本实验说明了流表溢出会降低业务平均吞吐量。



（a）流表未溢出时的业务平均吞吐量



（b）流表溢出时的业务平均吞吐量

图 3.3 OpenFlow 流表溢出对业务平均吞吐量的影响

3.3 流表溢出对 QoS 路由机制的影响

通过 3.2 节的分析可知流表溢出增加了控制器的工作负担，而 QoS 路由机制的实现依赖于控制器。因此，本小节分析 QoS 路由机制的路由过程是否受到流表溢出的影响以及受影响的程度，此外还分析了在流表溢出的场景下，QoS 路由机制对业务服务质量保障能力的影响程度。实验数据均通过 10 次实验求均值得到。

实验继续采用了 3.2.2 节的实验环境，不同的是关闭了控制器的默认路由模块，在控制器上实现了文献[29]提出的 QoS 路由机制 OpenQoS，OpenQoS 能够为业务动态地调整满足其 QoS 的路径。如图 3.4 所示，客户端向服务器请求服务，OpenQoS 路由机制的选路结果为 *S*1—*S*2—*S*3—*S*8。

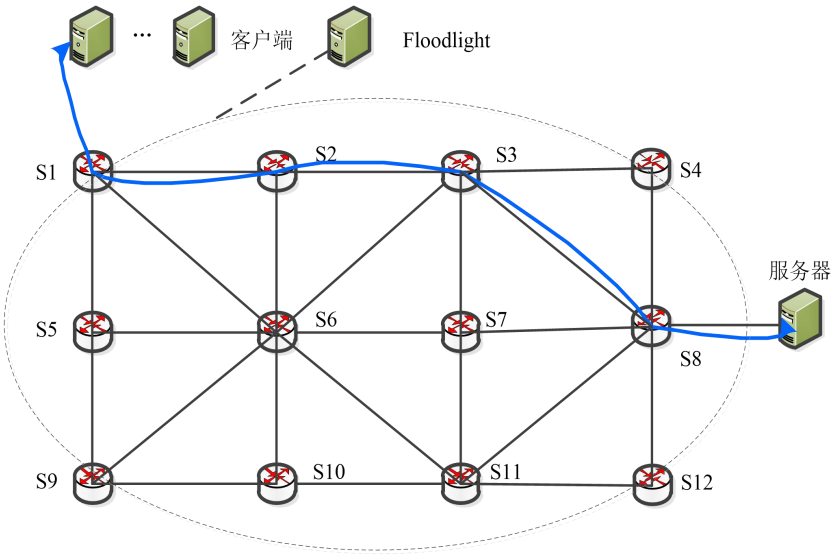


图 3.4 OpenQoS 选路结果

(1) 流表溢出对 QoS 路由机制工作量的影响

首先通过仿真实验来分析流表溢出对 QoS 路由机制工作量的影响，这里的工作量指的是路由调整次数，即新加入到网络中的流向 QoS 路由机制请求分配路径的次数与在传输过程中更改路由的次数之和。在实验中最多有 30 条业务流在同时传输，每隔 5s 加入一条新的业务流，设置 *S*8 的流表容量为 10，在第 11 条业务流加入进来时会使 *S*8 发生流表溢出。图 3.5 所示的是流表溢出对 QoS 路由机制工作量的影响，统计的是每 5s 间隔内 QoS 路由机制调整路由的次数。

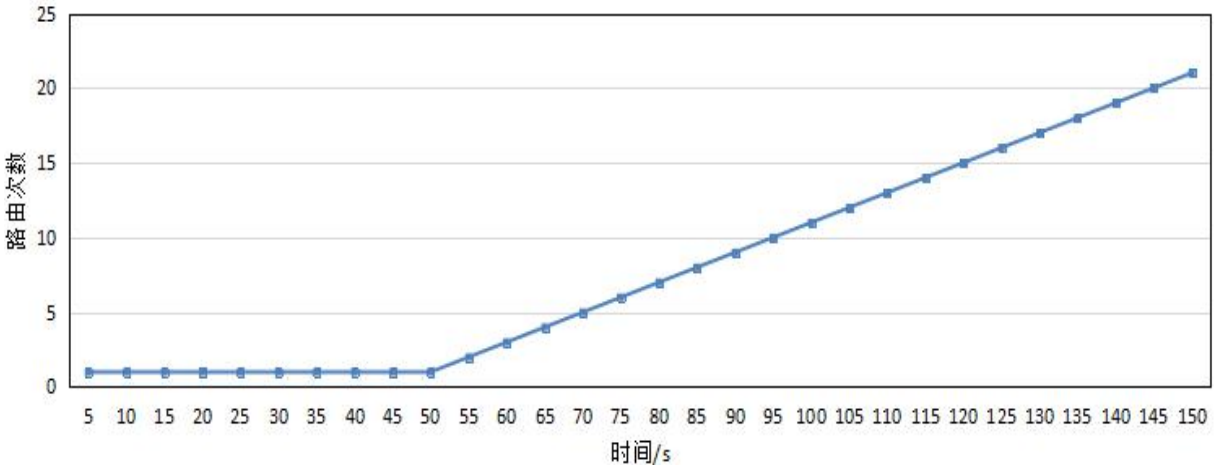


图 3.5 OpenFlow 流表溢出对 QoS 路由机制工作量的影响

从图 3.5 中可以看出，在流表溢出之前，随着新的业务流的加入，QoS 路由机制为每条流路由一次，由于此时网络资源充足，没有发生业务流 QoS 不满足的情况，所以每条流的路由调整次数为 1 次。当发生流表溢出后，新加入的业务流无法成功添加路径，等到下一条新的业务流到来的时候，QoS 路由机制不仅要为当前的新业务流路由，还需要为由于流表溢出没有成功添加路径的业务流路由。这是由于当新业务流的路径无法被添加的时候，显然就不能满足其传输的 QoS，但是 QoS 路由机制只能判断出是由于当前所选路径不满足新流的 QoS，并不能感知到是由于流表溢出造成的，因此会不断为其调整路径，也就造成了巨大的路由开销。从图 3.5 中可以看出，在网络资源充足的情况下，路由调整的次数与流表溢出后流的数量呈线性增长关系。

因此，流表溢出给 QoS 路由机制增加了大量的工作负载。

(2) 流表溢出对业务 QoS 的影响

通过(1)的分析可知在流表溢出后 QoS 路由机制要给那些由于流表溢出导致没有成功添加路径的流路由，在上述实验中是在第 11 条流加入到网络中时发生的流表溢出，因此选择观察第 11 条流传输 20s 内的吞吐量变化情况。如图 3.6 所示，流表溢出后吞吐量出现逐渐下降的趋势。吞吐量始终得不到提升的原因是：当 QoS 路由机制发现当前路径不满足客户端的 QoS 需求时会重新启动选路过程，然而，QoS 路由机制并不具备流表溢出感知能力，根据当前选路指标（丢包率、时延、吞吐量）所选出的路径仍然与之前的路径相同。QoS 路由机

制不断为其重新选路，但客户端业务的吞吐量却始终得不到提升，造成了巨大的路由开销。

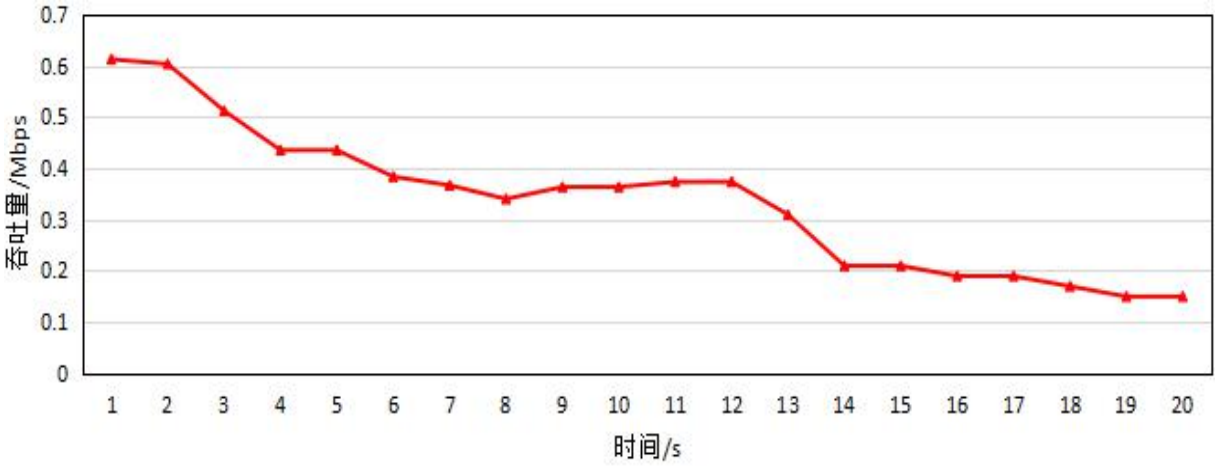


图 3.6 OpenFlow 流表溢出对业务吞吐量的影响

3.4 流表用量感知的 QoS 路由机制

面对流表溢出带来的不良影响，本节提出一种流表用量感知的多约束 QoS 路由机制。如何为业务选择满足其要求的路径是路由算法要解决的问题。路由算法需要在众多能够满足业务要求的路径中选择一条最优路径的问题就是多约束优化路径问题(MCOP)，因此本节通过改进文献[29]提出的 OpenQoS 路由机制来实现选路。文献[29]构建的 MCOP 模型主要优化的指标是时延和丢包率，而本节构建的 MCOP 模型考虑了时延、丢包率、吞吐量多项优化指标，同时将流表利用率也作为选路因素，避免了流表溢出带来的影响。此外，本节在 OpenQoS 路由机制的基础上添加了流表用量感知模块以配合选路。

3.4.1 流表用量感知路由模型

本节提出了 FUQR(Flow-table Usage-aware QoS Routing)，文献[29]提出的 OpenQoS 主要由拓扑管理模块、路由计算模块、流管理模块、接入管理模块构成，FUQR 在 OpenQoS 的基础上添加了流表用量感知模块，由于修改了 MCOP 模型，因此也修改了路由计算模块。

(1) 流表用量感知模块

流表用量感知模块能够获得当前网络中 SDN 交换机的流表使用情况，并将流表使用情况告知给路由计算模块。

(2) 路由计算模块

路由计算模块根据当前链路的状况与流表使用情况为业务选路，在之后的传输过程中如果业务的 QoS 得不到满足，路由计算模块会重新计算一条合适的路径来保障业务的 QoS。

假设网络以有向图 $G=(N, E)$表示，N 是节点集合，E 是节点间链路集合，节点 i 和节点 j 之间链路带宽表示为 b_{ij}，时延为 d_{ij}，丢包率为 p_{ij}，链路开销为 c_{ij}，节点 i 的流表利用率为 u_i。因此 MCOP 问题可以转化为以下线性规划模型，模型的参数如表 3.2 所示。优化的目标是要找到满足所有业务约束条件的最小代价和的路径。

表 3.2 线性规划模型参数

参数	定义
$b_{ij} \geq 0$	链路(i, j)的可用带宽
$c_{ij} \geq 0$	链路(i, j)的代价
$s \in N$	流的源节点
$t \in N$	流的目的节点
$P_{\max} \geq 0$	可接受的最大丢包率
$p_{ij} \geq 0$	链路(i, j)上的丢包率
$D_{\max} \geq 0$	可接受的最大时延
$d_{ij} \geq 0$	链路(i, j)上的时延
$U^i_{\max} \geq 0$	节点 i 最大流表容量
$u_i \geq 0$	节点 i 的流表用量
$B \geq 0$	流所需带宽

变量：$x_{ij} \in \{0,1\}$，选或不选(i,j)；

目标函数：

$$\min : \sum_{(i,j)\in A} c_{ij} x_{ij}; \tag{3.1}$$

约束条件：

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = \begin{cases} 1, \text{if } i = s, \\ -1, \text{if } i = t, \ \forall i \in N, \\ 0, \text{if } i \neq s, t, \end{cases} \tag{3.2}$$

$$\sum_{(i,j)\in A} p_{ij} x_{ij} \leq p_{\max}, \tag{3.3}$$

$$\sum_{(i,j)\in A} d_{ij} x_{ij} \leq D_{\max}, \tag{3.4}$$

$$B \leq b_{ij}, \forall (i,j) \in A, \tag{3.5}$$

$$x_{ij} \in \{0,1\}, \text{if } u_i < U^i_{\max}, \forall (i,j) \in A, \tag{3.6}$$

$$x_{ij} = x_{ji} = 0, \text{if } u_i = U^i_{\max}, (i,j) \in A, (j,i) \in A. \tag{3.7}$$

式(3.1)是目标函数，需要找到满足所有业务约束条件的最小代价和的路

径。对于路径的代价的计算，主要考虑了丢包率和时延 2 个因素，并分别赋予它们对路径代价影响的权重：

$$c_{ij} = \alpha d_{ij} + \beta p_{ij}, \forall (i,j) \in A, \tag{3.8}$$

其中，α 和 β 分别为时延和丢包率对链路代价影响的权重，可以通过管理这些参数来满足不同类型业务的需求。时延包括 4 个组成部分，定义如下：

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}, \tag{3.9}$$

总时延由 d_{nodal} 表示，d_{proc}，d_{queue}，d_{trans}，d_{prop} 分别表示处理时延、排队时延、发送时延和传播时延：

(1) d_{proc}:处理时延是交换设备转发处理所花费的时间，通常是微秒级，但是在 SDN 网络中还需要考虑外部控制器的处理时延，例如当一条新业务流的第一个分组到达交换设备的时候需要向控制器发送信息。

(2) d_{queue}:排队时延是数据包在输入队列中排队等待处理，在输出队列中等待转发所花费的时延。通常排队时延取决于队列当中数据包的数量，因此排队时延的波动会比较大，一般可以从微秒级变动到毫秒级。

(3) d_{trans}:发送时延是从发送数据帧的第一个比特算起，到该帧的最后一个比特发送完毕所需的时延。通常是毫秒级。

(4) d_{prop}:传播时延是数据包在链路传输所花费的时延，传播的速度相当于光速，在广域网中一般为毫秒级。

因此，对总时延 d_{nodal} 的估计大概为毫秒级，仿真实验中在 10ms 至 100ms 之间选取链路时延。同时仿真实验中对丢包率的选取大约为 1%。

式(3.2)为流量守恒定律，保证流的输入和输出守恒；式(3.3)(3.4)(3.5)分别为最大可接受丢包率、最大可接受时延和链路的带宽容量限制；式(3.6)(3.7)是对变量取值范围的限制，对于一条链路而言，有选和不选两种选择，分别用 1 和 0 表示。此外式(3.6)(3.7)还考虑了对节点 i 流表溢出时的约束，即当节点 i 的流表溢出时，不再选择经过节点 i 的链路。

业务吞吐量是通过交换机端口的数据发送速率来计算的，计算方式如式(3.10)所示，业务吞吐量就是时间间隔 t_{interval} 内数据包发送的速率。

$$\begin{aligned} bandwidth &= (b_{packetsCounted} - b_{lastPacketsCounted}) / t_{\text{int}erval} \\ bandwidthKBps &= bandwidth / 1024 \\ bandwidthMBps &= bandwidthKBps / 1024 \\ bandwidthMBps &= bandwidthMBps * 8 \end{aligned} \tag{3.10}$$

实时业务通常会对丢包率、时延、吞吐量、业务代价等多个参数同时提出要求，当这些参数相互独立时，选择满足多个参数限制的路由就成为 NP - Complete[47]问题，对于求解上述线性规划模型的算法时间复杂度的评估，变量数量与边的数量和业务数量有关，等于 $|A|$，而约束条件的数量也与边数和业务

数量相关，为$|N|$。本节将采用 IP_Solve[48]求解器来求解上述线性规划模型，求解器采用了改进的分支定界法求解线性规划模型，平均时间复杂度达到指数级，虽然在大规模网络中计算开销较大，但是能够在一般规模网络中快速找到最优解，而计算开销相对较小的智能优化算法一般只能找到近似解。

以上内容是通过经典的 MCOP 问题构建了 QoS 路由模型，接下来需要解决两个问题，分别是 QoS 路由模型的求解以及路由信息的管理：

(1) QoS 路由模型的求解

在用 IP_Solve 求解模型之前需要先建模，常用的建模语言包括 AMPL、Java、Python、PHP、R 等等，IP_Solve 也支持直接读取 lp 格式的 ASCII 编码文件来读入模型，本节将采用读取 lp 格式的 ASCII 编码文件的方式来读入模型，一个典型 lp 格式的线性规划模型如图 3.7 所示：

```
min: -x1 -2 x2 +0.1 x3 +3 x4;
r_1: +x1 +x2 <= 5;
r_2: +2 x1 -x2 >= 0;
r_3: -x1 +3 x2 >= 0;
r_4: +x3 +x4 >= 0.5;
x3 >= 1.1;

int x3, x4;
```

图 3.7 lp 格式的线性规划模型

图 3.7 中第一行表示目标函数，求解目标可以是满足约束条件的最大值或最小值，图 3.7 的目标是需要使变量的带权和最小；r_1、r_2、r_3、r_4 表示约束条件，可以看出这是一个多约束问题；最后需要声明变量的类型，这里声明的变量类型为整数，所以是一个整数线性规划问题。

(2) 路由信息的管理

在 IP_Solve 求解完模型之后需要将其结果转换成控制器能够操作的命令，例如在图 3.4 中选路结果为 $S1—S2—S3—S8$，其中包括三段链路 $S1—S2$、$S2—S3$、$S3—S8$，此时需要将此结果转化为流表项结构。如果使用一个矩阵来保存路由结果，同时使用一定的查找规则来查找路由矩阵，那么就能在查找的过程中生成正确的路由序列。

图 3.8 所示的是一个路由矩阵的例子，为了例子简单，假设网络中只有 5 台交换机，那么将会使用一个 5*5 的矩阵 A 来保存路由信息。

$$\begin{pmatrix}0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}->\begin{pmatrix}0&0&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}->\begin{pmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&1&0\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}->\begin{pmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}$$

图 3.8 路由矩阵示例

假设路由结果为 *S*1—*S*2—*S*3—*S*4，即包括三段链路 *S*1—*S*2、*S*2—*S*3、*S*3—*S*4，因此就会在 A[1][2]、A[2][3]、A[3][4]的原值上加 1。在读取一条路由结果的时候会从源交换机开始读，假设首次读到 A[i][j]>0，记为 *S*i—*S*j 并添加到路径集合 S 中去，然后将 A[i][j]=A[i][j]-1，接着直接跳转到矩阵的第 j 行查找第一个大于 0 的数，以此类推，直到查找到目的交换机，因此找到了从源交换机到目的交换机的路径。在图 3.8 中，源交换机是 *S*1，所以从第一行开始查找，由于 A[1][2]>0，便会令 A[1][2]=A[1][2]-1 之后查找第 2 行，令 A[2][3]=A[2][3]-1，接着查找第 3 行，又令 A[3][4]=A[3][4]-1，查找第 4 行的时候已经达到目的交换机 *S*4，查找结束，最后将路径集合中的三次结果 *S*1—*S*2、*S*2—*S*3、*S*3—*S*4 按照顺序组合起来，最后只需要将路径转化为流表项下发即可。整个路由矩阵查找算法的伪代码如表 3.3 所示：

表 3.3 路由矩阵查找算法伪代码

算法 1：路由矩阵查找算法

```
输入：路由矩阵 Routes[N+1][N+1]，N 表示交换机数量
      源交换机 startSwitch>0 && startSwitch<=N
      目的交换机 endSwitch>0 && endSwitch<=N
01 开始
02 nextSwitch = startSwitch;
   // 更新下一跳交换机
03 while nextSwitch !=  endSwitch
    // 下一跳不是目的交换机，继续查找
04      i = nextSwitch;
        // 从下一跳交换机开始查找
05      for j = 1 : N
        // 查找首个不为零的元素
06          if Routes[i][j] > 0
07              Routes[i][j] = Routes[i][j] - 1;
                // 取出(i,j)这段路径
08              添加 Si—Sj 到集合 S 中;
09              nextSwitch = j;
                // 更新下一跳交换机为 j
10              break;
11          end if
12      end for
13  end while
14  结束
输出：路径集合 S
```

综上所述，可以总结出路由计算整体流程，如图 3.9 所示，启动 QoS 路由机制之后，开始收集链路状态信息与流表用量信息用来生成 QoS 路由模型，路由模型使用建模语言来描述，然后通过线性规划求解器 IP_Solve 求解模型，将

求解的结果保存在路由矩阵之中，接着通过查找路由矩阵来得到路径集合，最后再通过路径集合生成流表项信息，控制器就可以下发流表了。同时需要注意的是由于 OpenFlow 协议版本的不同，由路径集合生成的流表项格式是不相同的，随着 OpenFlow 协议复杂度的增加，流表项格式也变得更为复杂。

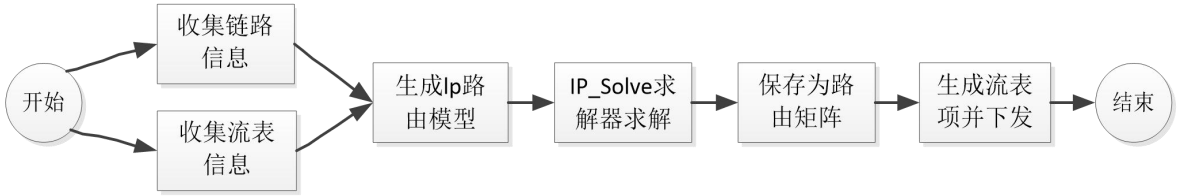


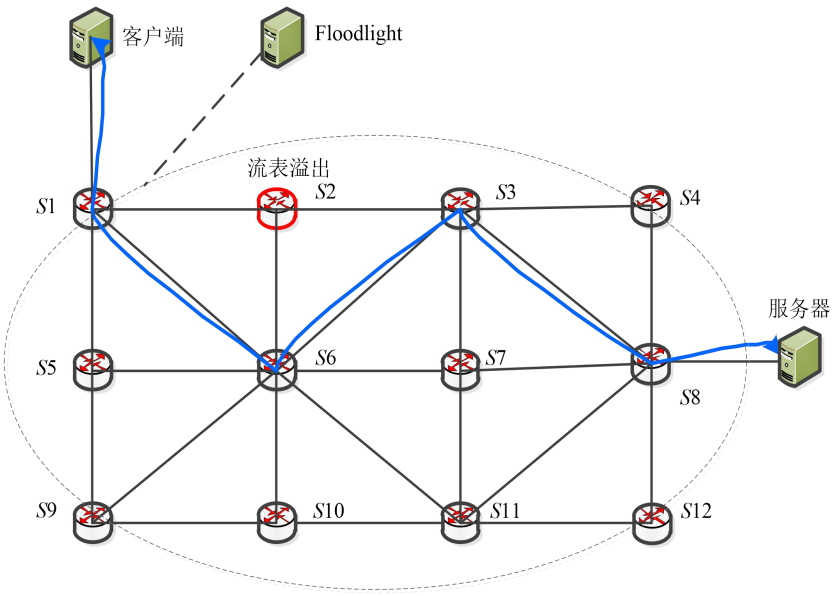
图 3.9 路由计算流程图

3.4.2 性能评估

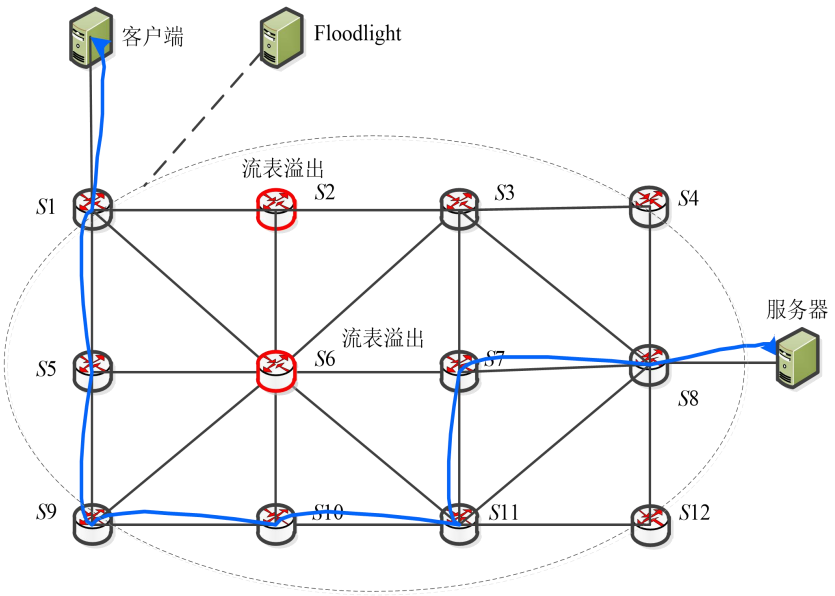
本节在流表溢出的场景下，通过仿真实验分析了 FUQR 的流表感知能力，并比较了 FUQR 与文献[29]提出的 OpenQoS 的性能。

如图 3.10 所示，通过 Mininet 生成一个 SDN 网络，网络中共有 12 台交换机，设定链路带宽为 30Mbps，丢包率 1%，时延 10ms，SDN 控制器采用较为主流的开源控制器 Floodlight，服务器通过交换机 *S*8 接入网络，客户端通过交换机 *S*1 接入网络，通过服务器设置业务流的传输速率为 1Mbps。

实验首先分析了 FUQR 对流表用量感知的性能，在没有流表溢出的场景下，FUQR 选路结果为 *S*1—*S*2—*S*3—*S*8；接着清除所有流表项，设置交换机 *S*2 的流表容量为 30，并向 *S*2 的流表中插入 30 条流表，此时如若再向 *S*2 添加流表项则会发生溢出，启动 FUQR 重新为业务流选路，路由结果如图 3.10(a)所示，为 *S*1—*S*6—*S*3—*S*8，可以看到 FUQR 在选路过程中避开了将会发生流表溢出的交换机 *S*2。



（a）*S*2 流表溢出时 FUQR 选路情况



（b）S2、S6 流表溢出时 FUQR 选路情况

图 3.10 流表溢出场景下 FUQR 选路情况

然后在 S2 流表溢出的场景下，清除 S1、S6、S3 与 S8 的流表项，并采用同样的方法使交换机 S6 流表溢出，重新启动 FUQR 为业务选路，选路结果如图 3.10(b)所示，为 S1—S5—S9—S10—S11—S7—S8，避开了 S2、S6。上述实验结果能够说明 FUQR 具有良好的流表感知能力，进一步保障了业务 QoS。

接着实验比较了在流表溢出场景下 FUQR 与 OpenQoS 的路由性能。图 3.11 所示的是 S2 流表溢出时 FUQR 与 OpenQoS 的选路情况。

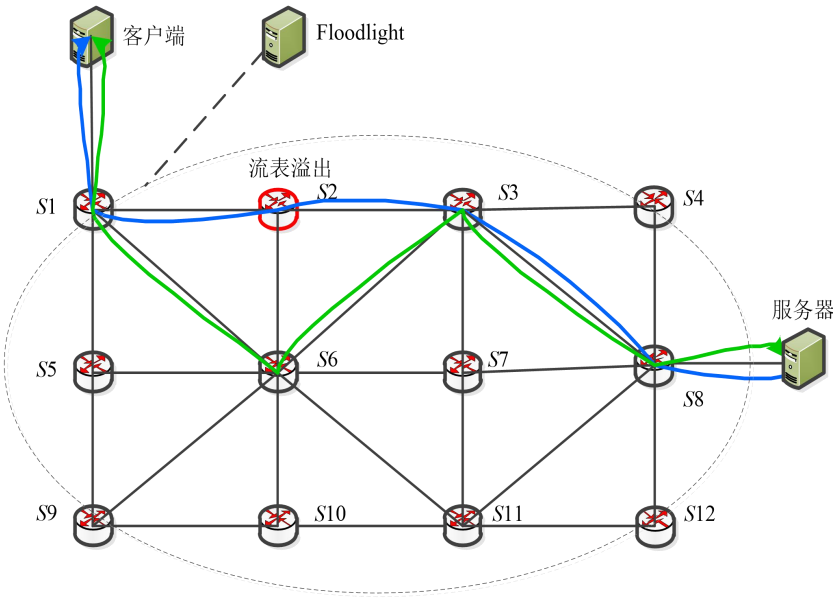


图 3.11 流表溢出场景下 FUQR 与 OpenQoS 选路情况

文献[29]提出的 OpenQoS 没有流表用量感知能力，选路结果为 S1—S2—S3—S8，当控制器向 S2 添加流表项时发生了溢出，但 OpenQoS 不会检测到流

表溢出的情况，会保持当前的选路结果，造成了巨大的选路开销；而改进后的 FUQR 能够检测到流表溢出，并及时地为客户端更换了路径，所选路径为 *S*1—*S*6—*S*3—*S*8。

图 3.12 和图 3.13 分别表示 FUQR 与 OpenQoS 的路由调整次数变化情况以及客户端吞吐量变化情况。从图 3.12 可以看到，OpenQoS 不会检测到流表溢出的情况，会保持当前的选路结果，在 20s 内一共为客户端业务路由 5 次，但仍然无法保障客户端业务的 QoS，而 FUQR 在 20s 内只为客户端业务路由 1 次，同时还保障了客户端业务的 QoS，具体情况可以从图 3.13 看到，尽管 OpenQoS 不断为客户端业务选路，但是客户端吞吐量始终呈现下降的趋势，而 FUQR 只为客户端路由 1 次，同时还保障了客户端的正常传输。原因在于 *S*2 流表的溢出，*S*1 向 *S*2 转发的数据包大量丢失，从而造成了客户端吞吐量持续下降，但是 OpenQoS 会不断重新为其调整相同的最短路径。而 FUQR 及时感知到了流表的使用情况避开了 *S*2，能够维持其吞吐量水平，也避免了重路由的巨大开销。

因此，本节提出的 FUQR 具有流表用量感知能力，能够在保障业务 QoS 需求的同时避免了流表溢出带来的影响。

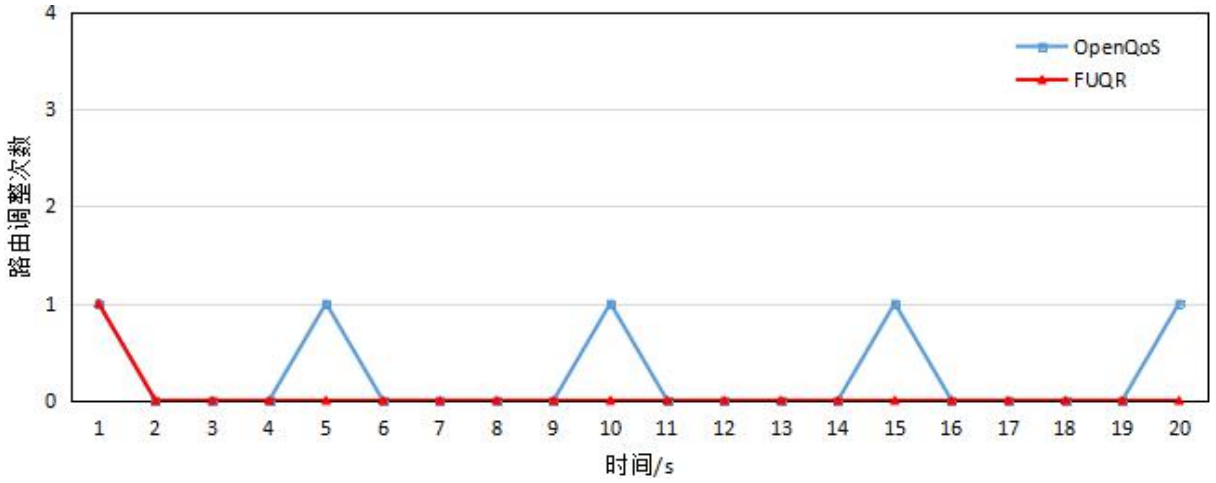


图 3.12 FUQR 与 OpenQoS 的路由调整次数变化情况

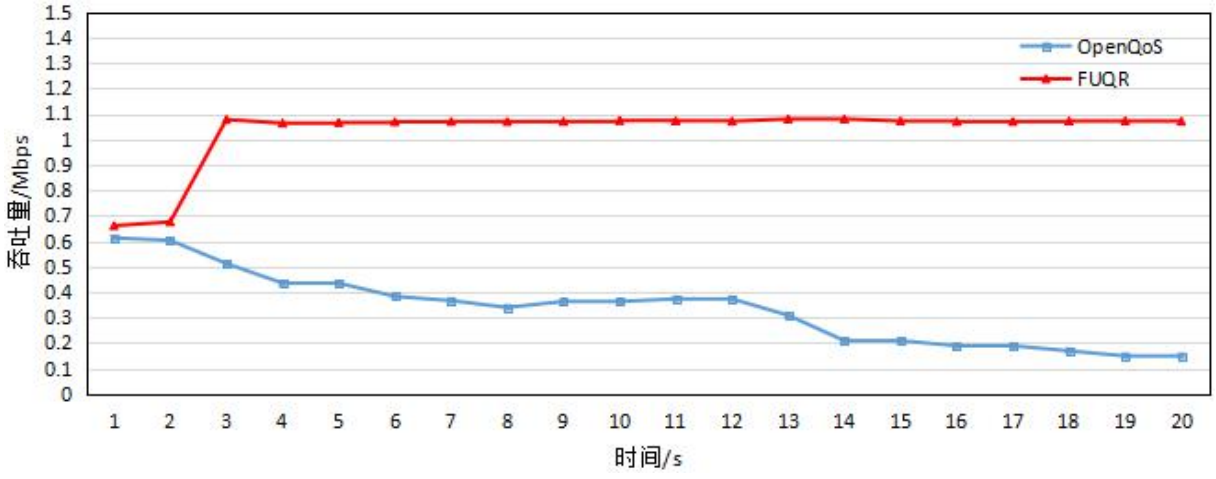


图 3.13 FUQR 与 OpenQoS 对客户端吞吐量保障情况

3.5 小结

SDN 的提出有效地提高了网络应用的性能，但有限的流表资源带来了新的问题。由于流表使用率的不均衡，给原本容量就十分有限的交换机带来了资源耗尽的压力，最终会导致瓶颈交换机产生流表溢出的现象。本章首先分析了 OpenFlow 网络中流表溢出的现象及原因，接着通过仿真实验分析了流表溢出的影响，证实了 OpenFlow 协议的流表溢出对 QoS 路由机制的性能有至关重要的影响。为了避免流表溢出对 QoS 路由机制产生的影响，本章提出了流表用量感知路由模型，能够让 QoS 路由机制在路由的过程中避开瓶颈交换机，极大地减轻控制器的负载，同时有效地保障了网络应用的 QoS。

在保障应用 QoS 的同时，FUQR 也带来了网络开销。网络开销主要包括 FUQR 启动时采集网络拓扑和链路信息所带来的开销以及启动后定时采集业务 QoS 指标和流表用量的开销，该开销不仅与交换机、链路的数量有关，还与系统的并发负载能力有关。

第 4 章 软件定义的 VANET 动态 QoS 路由机制

4.1 引言

现有的 VANET 架构难以保障应用服务的 QoS，因此许多研究者将 SDN 引入到 VANET 中形成软件定义的 VANET 架构。本章首先设计了一种面向异构多网接入的软件定义 VANET 架构，完善了现有工作中数据平面的设计，接着提出一种流表用量感知的动态 QoS 保障框架，允许使用模块化的方式管理网络，并支持业务流的动态加入和退出，最后还建立了多业务流多约束条件下流表用量感知的 QoS 路由模型，该模型不仅考虑了丢包、时延和吞吐量等链路状态参数，还考虑了应用服务的需求以及交换机的流表使用情况，为 VANET 应用服务提供并发的动态 QoS 路由。本章通过仿真平台实现了所提出的保障框架的各个模块并进行了仿真实验，实验结果表明本章提出的动态 QoS 路由机制不仅能够满足多业务对各自丢包、时延和吞吐量的要求，还能够感知流表用量从而避免流表溢出对 QoS 路由的影响，进一步提高了网络 QoS 保障的性能。

4.2 软件定义的 VANET 架构

VANET 可以提供各类安全和非安全相关的服务，但现有的 VANET 难以保障应用服务的 QoS 需求。软件定义网络（SDN）以系统化的灵活控制网络的方式出现，其分离的数据与控制平面为网络带来了可编程性。许多研究者提出将软件定义网络架构引入 VANET，从而分离 VANET 数据平面与控制平面，实现网络的系统化灵活控制，带来可编程性。

软件定义的 VANET 能够增强许多原有的服务：

(1) 安全服务：能够保留或限制一些频段，让安全服务使用这些频段。

(2) 监控服务：控制器简单地下发流表就能让监控数据到达请求节点。

(3) 虚拟化服务：SDN 可以让不同的流选择不同的频段。

本章利用软件定义 VANET 的控制与转发相分离以及集中控制的优势，设计了一种面向异构多网接入的软件定义 VANET 架构，完善了现有工作中数据平面的设计。ONF 组织提出的 SDN 系统架构主要由数据平面，控制平面和管理平面构成。数据平面只需要实现转发和处理数据的功能；控制平面主要负责逻辑控制部分；管理平面由若干个 SDN 应用构成，通过北向接口与 SDN 控制器进行交互。本章提出的软件定义的 VANET 体系架构如图 4.1 所示，本章在体系设计上遵循 SDN 三层平面划分的规定，同时考虑了 VANET 架构的特点。

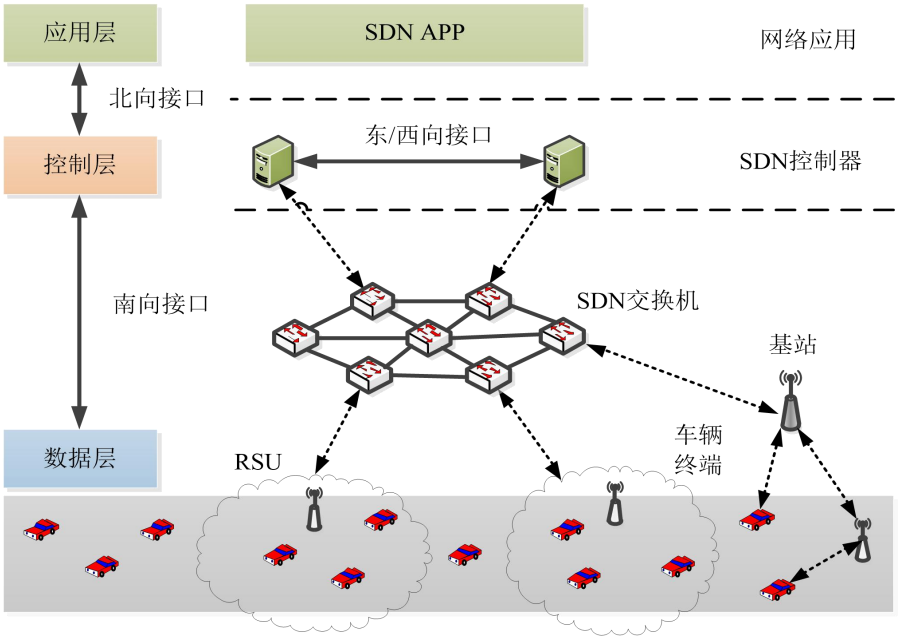


图 4.1 软件定义的 VANET 架构

软件定义的 VANET 主要元素包括：

(1) SDN APP。基于控制平面的北向接口，提供软件支持、网络监控、规则定义等服务，并为 VANET 应用服务提供网络支持。

(2) SDN 控制器。是网络的逻辑控制中心，管理并控制区域内的网络行为。此外，控制器之间通过东/西向接口协同合作管理网络。

(3) SDN 交换机。数据转发设备，接受 SDN 控制器的控制，在数据平面上负责数据的转发和处理。

(4) 异构接入网络。在数据平面上为车辆终端提供异构多样的接入方式。例如，通过 RSU、4G/LTE 等接入。

(5) 终端。移动车辆作为数据收发的终端。

在上述体系结构中，车辆与控制器通信的方式有三种：车辆终端通过蜂窝网络接入 SDN 核心交换层与控制器通信；车辆终端通过 RSU 接入 SDN 核心交换层与控制器通信；车辆终端通过 RSU 接入蜂窝网络再接入 SDN 核心交换层与控制器通信。

软件定义的 VANET 可以提供的服务类型多样，在实际网络部署中，数据层面的 SDN 交换机担负起数据转发的功能，SDN 交换机之间链路的传输带宽、负载等情况等直接影响数据传输的性能。本章对上述架构中 SDN 核心交换层的 QoS 路由问题展开研究，主要考虑以下两点：

(1) 由于车辆的移动性，RSU 的任务负载量是动态变化的。因此，需要根据 RSU 当前的任务负载量来考虑动态 QoS 路由，以达到保障接入 RSU 的车辆业务的 QoS。

(2) 由于 VANET 服务的实时性，因此需要考虑多业务并发的路由机制，以

提高动态 QoS 路由的效率，及时地保障接入 RSU 的车辆业务的 QoS。

4.3 面向多业务多约束的动态 QoS 路由机制

第 3 章提出了一种流表用量感知的 QoS 路由机制，而本节也将继续研究流表用量感知路由，但不同的是本节提出的 QoS 路由机制能够满足多业务并发的 QoS 需求，并能动态地加入和删除业务。本节首先介绍流表用量感知的动态 QoS 保障框架，然后介绍 QoS 路由模型。

4.3.1 流表用量感知的动态 QoS 保障框架

随着移动设备和移动流量的增长，车辆之间的通信（V2V）以及车辆与基础设施的通信（V2I）将会有更多的需求并不断增长。VANET 提供了广泛的服务，为了满足具有不同特点的应用服务的 QoS，本节提出了软件定义的 VANET 下流表用量感知的动态 QoS 保障框架，如图 4.2 所示：

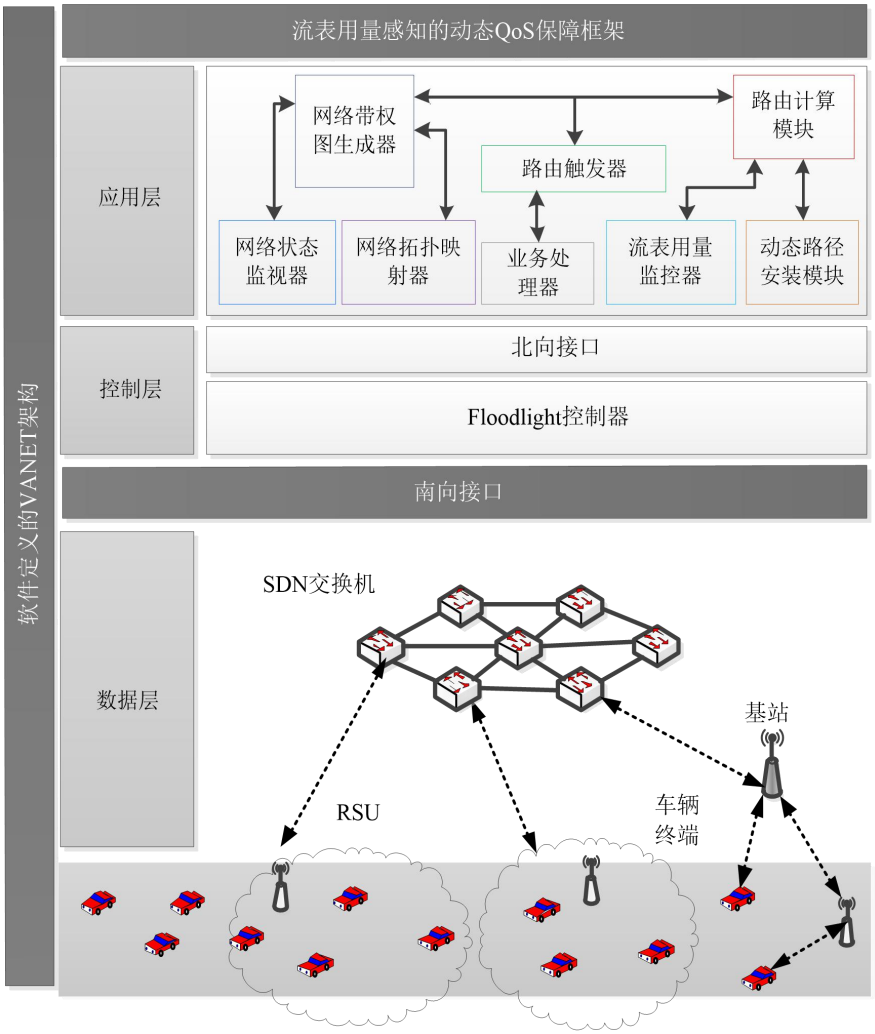


图 4.2 流表用量感知的动态 QoS 保障框架

从图 4.2 可以看到，流表用量感知的动态 QoS 保障框架是通过 Floodlight 控制器的北向接口扩展而来的，整个框架主要由 8 个模块构成：

(1) 网络状态监视器。实时地搜集链路的状态信息，比如每一条链路当前的吞吐量，并将采集到的数据发送给网络带权图生成器。例如在一个 5 台交换机组成的网络中，用一个 5*5 大小的矩阵来保存链路当前的可用带宽，吞吐量的计算方式参照 3.4.1 节公式(3.10)。如图 4.3 所示，每条链路的初始带宽为 BMbps，交换机 Si 向交换机 Sj 发送数据包的速率为 b_{ij}Mbps，b_{ij} 与 b_{ji} 不一定相等，因此链路(i,j)可用带宽的计算方式为$(B\text{-}b_{ij})$Mbps。

$$\begin{pmatrix} 0 & B & B & B & B \\ B & 0 & B & B & B \\ B & B & 0 & B & B \\ B & B & B & 0 & B \\ B & B & B & B & 0 \end{pmatrix} - \begin{pmatrix} 0 & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & 0 & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & 0 & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & 0 & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & 0 \end{pmatrix} = \begin{pmatrix} 0 & B-b_{12} & B-b_{13} & B-b_{14} & B-b_{15} \\ B-b_{21} & 0 & B-b_{23} & B-b_{24} & B-b_{25} \\ B-b_{31} & B-b_{32} & 0 & B-b_{34} & B-b_{35} \\ B-b_{41} & B-b_{42} & B-b_{43} & 0 & B-b_{45} \\ B-b_{51} & B-b_{52} & B-b_{53} & B-b_{54} & 0 \end{pmatrix}$$

图 4.3 链路可用宽带矩阵

(2) 网络拓扑映射器。实时地监测网络拓扑结构的变化，比如，是否有交换机增减，是否有终端机增减，并将数据发送给网络带权图生成器。网络的拓扑结构可以用一个矩阵保存，例如在一个 5 台交换机组成的网络中，用一个 5*5 大小的矩阵来保存网络的拓扑结构，此矩阵实质上就是邻接矩阵。如图 4.4 所示：

$$\begin{pmatrix} \infty & 1 & 1 & 0 & 0 \\ 1 & \infty & 1 & 1 & 0 \\ 1 & 1 & \infty & 1 & 0 \\ 0 & 1 & 1 & \infty & 1 \\ 0 & 0 & 0 & 1 & \infty \end{pmatrix}$$

图 4.4 5*5 邻接矩阵

邻接矩阵能够反映交换机之间的连接情况，但是无法反映交换机之间是通过哪些端口连接的，同时也无法获取主机与交换机的连接信息。因此，除了邻接矩阵，还需要保存主机与交换机以及交换机之间的端口连接关系。表 4.1 所示的是交换机端口连接关系表，其中包括 48 位交换机 mac 地址、交换机端口号、相连的交换机 mac 地址以及相连的主机 mac 地址。

表 4.1 交换机端口连接关系表

属性	属性说明
交换机标识	48 位 mac 地址
交换机端口号	4 位二进制
相连交换机标识	48 位 mac 地址
相连主机标识	48 位 mac 地址

(3) 网络带权图生成器。接收网络状态监视器和网络拓扑映射器发送过来的链路与网络拓扑数据，并按照一定的权重计算并生成网络权重视图。网络带权图

可以用一个矩阵保存，例如在一个 5 台交换机组成的网络中，用一个 5*5 大小的矩阵来保存网络的带权图，如图 4.5 所示，在无向图中 $c_{ij}=c_{ji}$，c_{ij} 表示链路 (i,j)的上的代价。

$$\begin{pmatrix} \infty & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & \infty & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & \infty & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & \infty & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & \infty \end{pmatrix}$$

图 4.5 5*5 带权矩阵

(4) 流表用量监控器。负责实时地监控所有交换机流表的使用情况，并将监控的结果发送给路由计算模块以便路由决策的进行。交换机的流表使用情况可以用一张表格保存起来，如表 4.2 所示，每一台交换机都会统计它的最大流表容量以及当前的流表使用量。

表 4.2 交换机流表使用情况表

属性	属性说明
交换机标识	48 位 mac 地址
流表总量	最大流表项容量
流表用量	已使用的流表项数量

(5) 路由计算模块。等待路由触发器的路由指令，根据网络权重视图、多业务的 QoS 需求以及流表使用情况来为多业务并发选路。路由计算的流程如图 4.6 所示，在接收到网络权重视图、多业务的 QoS 需求以及流表使用情况之后，生成 QoS 路由模型，然后求解 QoS 路由模型，最后触发动态路径安装模块将路由结果转化成流表信息下发给交换机，关于路由模型的内容将在下一节详细介绍。

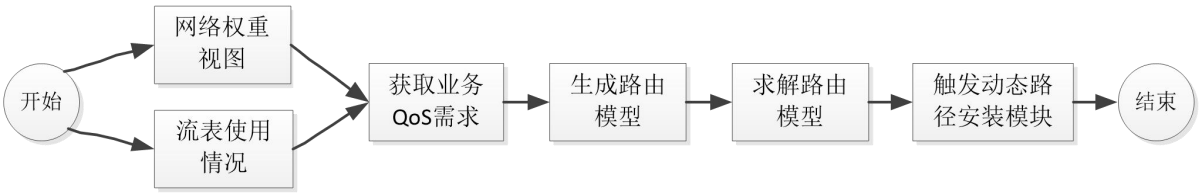


图 4.6 路由计算流程图

(6) 业务处理器。业务处理器能够支持业务的动态接入和退出，当有业务接入时，通知路由触发器，若当前的网络资源不足以满足业务需求则拒绝业务接入；当有业务退出时，通知路由触发器以便删除业务的路由信息，同时删除业务的信息。表 4.3 所示的是业务处理器管理的业务信息，业务标识是唯一标记一条业务的字符串，业务可以有在线和退出两种状态，业务带宽的单位是 Mbps，业务流表标识是该业务对应的所有流表项的标识，在安装的时候需要指定，并通过此标识删除。

表 4.3 业务信息表

属性	属性说明
业务标识	全局唯一的字符串
业务状态	在线/退出
业务带宽	Mbps
业务流表标识	与业务标识保持一致

业务管理的总体流程如图 4.7 所示，当有新业务接入的时候会先通知路由触发器，路由触发器会尝试给业务分配路径，如果分配失败会反馈给业务管理器拒绝业务接入，如果分配成功业务管理器会在业务信息表里添加此业务的信息，会选取一个全局唯一的标识来标记此业务，之后便进入了监管阶段。当业务退出时，业务管理器需要删除此业务在业务信息表里的记录，并归还业务标识。

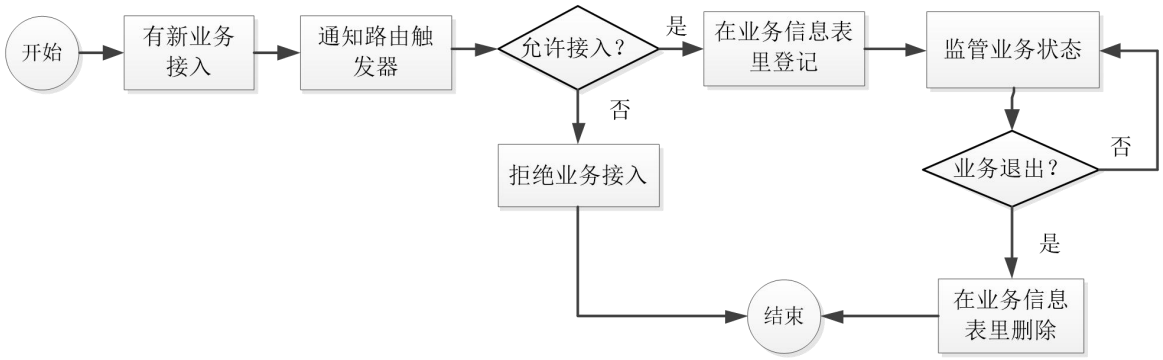


图 4.7 业务管理流程图

(7) 路由触发器。触发路由计算模块进行路由计算，同时监控成功接入业务的 QoS，当其 QoS 受到影响时触发路由计算，当业务退出时需要通知业务管理器来删除业务有关信息，但是该业务已经下发的流表信息由路由触发器删除，删除的依据是业务信息表。

(8) 动态路径安装模块。根据路由计算模块选路的结果下发业务的流表，需要将路由计算的结果转化为流表项。

流表用量感知的动态 QoS 保障框架的工作流程如图 4.8 所示，首先，网络拓扑映射器检测网络的拓扑结构，同时网络状态监视器收集链路的状态信息，两者都将数据发送给网络带权图生成器生成网络权重图；然后，业务处理器检测新接入的业务，如果有业务接入，就通知路由触发器，路由触发器根据业务的需求触发路由计算模块进行路由计算；接着，路由计算模块根据网络权重图与流表使用情况执行路由算法，如果路由失败，即网络当前可用资源不能满足新接入业务的 QoS 需求，业务处理器就会拒绝新业务的接入，否则就通过动态路径安装模块解析路由结果并下发流表；最后，路由触发器监控成功接入业务的 QoS，如果发现业务的 QoS 不满足其需求则重新触发路由计算模块选路，若业务退出，则通知业务管理器删除此业务信息，同时删除此业务的路由信息。

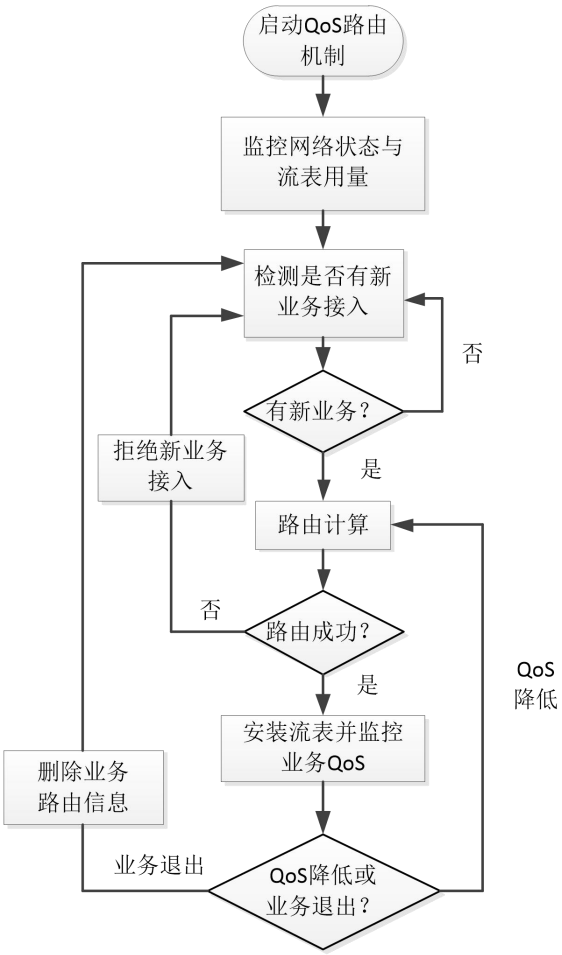


图 4.8 流表用量感知的动态 QoS 保障框架的工作流程

4.3.2 流表用量感知的多业务多约束 QoS 路由模型

VANET 提供了广泛的服务，这些服务又具有不同的特点，因此构建 QoS 路由模型需要考虑不同的 VANET 业务之间具有不同的 QoS 需求这一特点。

文献[27]提出的路由模型结合了多商品流问题和受限最短路径问题。多商品流问题是指在多源多目的的情况下，如何为所有商品分配满足其约束的路径并且总代价最小的问题，而结合了受限最短路径问题的多商品流问题是指在多源多目的的情况下，如何为所有商品分配满足其约束条件并且代价最小的最短路径的问题。因此，文献[27]提出的路由模型适用于本章提出的不同类型 VANET 业务的 QoS 路由问题。但从第 3 章的实验中可以看出，流表溢出会降低文献[27]提出的 QoS 路由机制的性能，因此本章在文献[27]提出的路由模型的基础上进行了改进，增加了流表用量的限制条件，将流表用量也作为选路的依据，从而进一步提升网络 QoS 保障的性能。

根据链路状态参数(丢包率、时延、吞吐量)、业务需求以及流表使用情况来选择合适的路径。假设网络以有向图 $G=(N, E)$表示，N 是节点集合，E 是节点间链路集合，节点 i 和节点 j 之间链路带宽表示为 b_{ij}，时延为 d_{ij}，丢包率为

p_{ij}，每条流的开销为 c_{ij}，网络中需要路由的流的数量为 k，节点 i 的流表使用量为 u_i。构建出的线性规划模型如下所示，模型的参数如表 4.4 所示。优化的目标是要找到满足所有业务约束条件的最小代价和的路径。

表 4.4 线性规划模型参数

参数	定义
$b_{ij} \geq 0$	链路(i, j)的可用带宽
$c_{ij} \geq 0$	链路(i, j)的代价
$\alpha \geq 0$	时延权重
$\beta \geq 0$	丢包率权重
$s_k \in N$	流 k 的源节点
$t_k \in N$	流 k 的目的节点
$f_k \geq 0$	流 k 的数量
$P_{\max}^k \geq 0$	可接受的最大丢包率
$p_{ij} \geq 0$	链路(i, j)上的丢包率
$D_{\max}^k \geq 0$	可接受的最大时延
$d_{ij} \geq 0$	链路(i, j)上的时延
$U_{\max}^i \geq 0$	节点 i 最大流表容量
$u_i \geq 0$	节点 i 的流表用量
$B^k \geq 0$	流 k 所需带宽

变量：$x_{ij}^k \geq 0$，业务 k 在链路（i,j）上流的总和；

目标函数：

$$\min: \sum_{(i,j)\in A}\sum_{k\in K} c_{ij}x_{ij}^k; \tag{4.1}$$

约束条件：

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = \begin{cases} f_k, \text{if } i = s_k, \\ -f_k, \text{if } i = t_k, \\ 0, \text{if } i \neq s_k, t_k, \end{cases} \forall i \in N, \forall k \in K, \tag{4.2}$$

$$\sum_{(i,j)\in A} p_{ij}x_{ij}^k \leq p_{\max}^k, \forall k \in K, \tag{4.3}$$

$$\sum_{(i,j)\in A} d_{ij}x_{ij}^k \leq D_{\max}^k, \forall k \in K, \tag{4.4}$$

$$\sum_{k\in K} B^k x_{ij}^k \leq b_{ij}, \forall (i,j) \in A, \tag{4.5}$$

$$x_{ij}^k \in \{0,1\}, \text{if } u_i < U_{\max}^i, \forall (i,j) \in A, \forall k \in K, \tag{4.6}$$

$$x_{ij}^k = x_{ji}^k = 0, \text{if } u_i = U_{\max}^i, (i,j) \in A, (j,i) \in A, \forall k \in K. \tag{4.7}$$

式(4.1)是目标函数，需要找到满足所有业务约束条件的最小代价和的路径。对于路径的代价的计算，主要考虑了丢包率和时延 2 个因素，并分别赋予它们对路径代价影响的权重：

$$c_{ij} = \alpha d_{ij} + \beta p_{ij}, \forall (i,j) \in A, \tag{4.8}$$

其中，α 和 β 分别为时延和丢包率对链路代价影响的权重，可以通过管理这些参数来满足不同类型业务的需求。时延包括 4 个组成部分，定义如下：

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}, \tag{4.9}$$

总时延由 d_{nodal} 表示，d_{proc}，d_{queue}，d_{trans}，d_{prop} 分别表示处理时延、排队时延、发送时延和传播时延：

(1) d_{proc}:处理时延是交换设备转发处理所花费的时间，通常是微秒级，但是在 SDN 网络中还需要考虑外部控制器的处理时延，例如当一条新流的第一个分组到达交换设备的时候需要向控制器发送信息。

(2) d_{queue}:排队时延是数据包在输入队列中排队等待处理，在输出队列中等待转发所花费的时延。通常排队时延取决于队列当中数据包的数量，因此排队时延的波动会比较大，一般可以从微秒级变动到毫秒级。

(3) d_{trans}:发送时延是从发送数据帧的第一个比特算起，到该帧的最后一个比特发送完毕所需的时延。通常是毫秒级。

(4) d_{prop}:传播时延是数据包在链路传输所花费的时延，传播的速度相当于光速，在广域网中一般为毫秒级。

因此，对总时延 d_{nodal} 的估计大概为毫秒级，仿真实验中在 10ms 至 100ms 之间选取链路时延。同时仿真实验中对丢包率的选取大约为 1%。

式(4.2)为流量守恒定律，保证流的输入和输出守恒；式(4.3)(4.4)(4.5)分别为最大可接受丢包率、最大可接受时延和链路的带宽容量限制；式(4.6)(4.7)是对变量取值范围的限制，对于一条链路而言，有选和不选 2 种选择，分别用 1 和 0 表示。此外式(4.6)(4.7)还考虑了对节点 i 流表溢出时的约束，即当节点 i 的流表溢出时，不再选择经过节点 i 的链路。

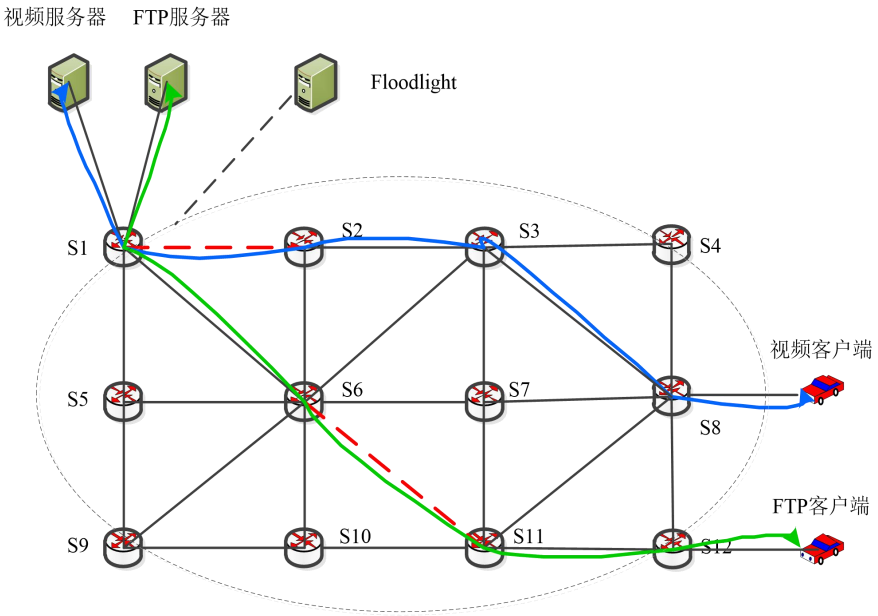
业务吞吐量是通过交换机端口的数据发送速率来计算的，参照 3.4.1 节公式(3.10)。以上内容是通过经典的多商品流问题与最短路径问题构建了 QoS 路由模型，对于上述 QoS 路由模型的求解方法与 3.4.1 节所述一致，在此不再赘述。对于求解上述线性规划模型的算法时间复杂度的评估，变量数量与边的数量和业务数量有关，等于 $|A||K|$，而约束条件的数量也与边数和业务数量相关，为 $|N||K|+|A|+|K|$。由于在大规模软件定义 VANET 中，通常会由多个控制器实现分布式控制，因此，单个控制器所控制的网络规模一般不会太大，因此采用了求解器求解的方式以便能够快速找到最优解，同时满足 VANET 环境的时延敏感的特性，而计算开销相对较小的智能优化算法一般只能找到近似解。

4.4 性能评估

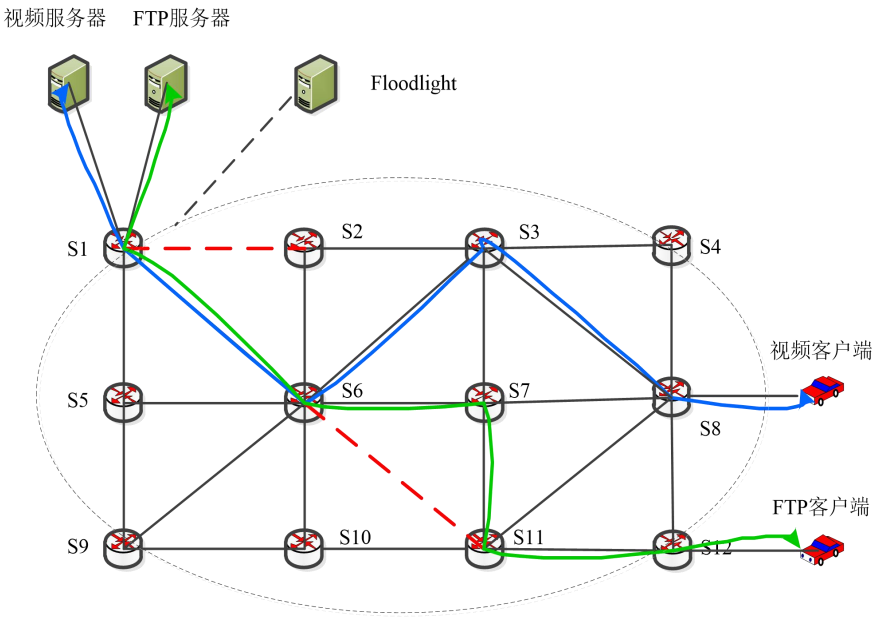
实验采用 Mininet 作为仿真平台，Floodlight 作为 SDN 控制器，OpenFlow1.3 作为控制器与数据平面交互的南向接口协议，并基于控制器的北向接口实现了 4.3 节所提出的流表用量感知的动态 QoS 保障框架。网络中共有 12 台 SDN 交换机，设定链路的带宽为 40Mbps，丢包率为 1%，时延为 10ms。视频服务器采用 VLC Media Player[49]，通过 cvlc 命令推送高清视频流到视频客户端，FTP 服务器采用 Apache FtpServer[50]，在 FTP 客户端通过 wget 命令下载文件，同时实验使用 Iperf 工具实现链路拥塞，服务器通过交换机 *S*1 接入网络，并通过服务器配置设定了文件传输的平均带宽为 5Mbps，视频传输的平均带宽为 6Mbps。实验一共分为 2 部分，第 1 部分是在流表未溢出的场景下对客户端选路与吞吐量性能的分析；第 2 部分是在流表溢出的场景下对业务动态接入下的性能分析。实验数据均通过 10 次实验求均值得到。

(1) 客户端选路与吞吐量性能分析

在流表未溢出的场景下，首先评估了本章所提的 QoS 路由机制对业务 QoS 的保障性能。视频客户端和 FTP 客户端分别通过 *S*8 与 *S*12 接入网络。实验通过启用和不启用本章所提出的 QoS 路由机制 2 种情况来观察客户端选路和吞吐量的变化情况，实验结果如图 4.9 和图 4.10 所示。在该实验场景中，通过在一定时间时分别向链路 *S*6—*S*11 与 *S*1—*S*2 增加负载 FTP-Loader 与 Video-Loader 来改变该链路的拥塞情况，通过使 *S*6—*S*11 和 *S*1—*S*2 链路拥塞而让 FTP 客户端和视频客户端的 QoS 受到影响。



（a）关闭 QoS 路由机制的选路情况

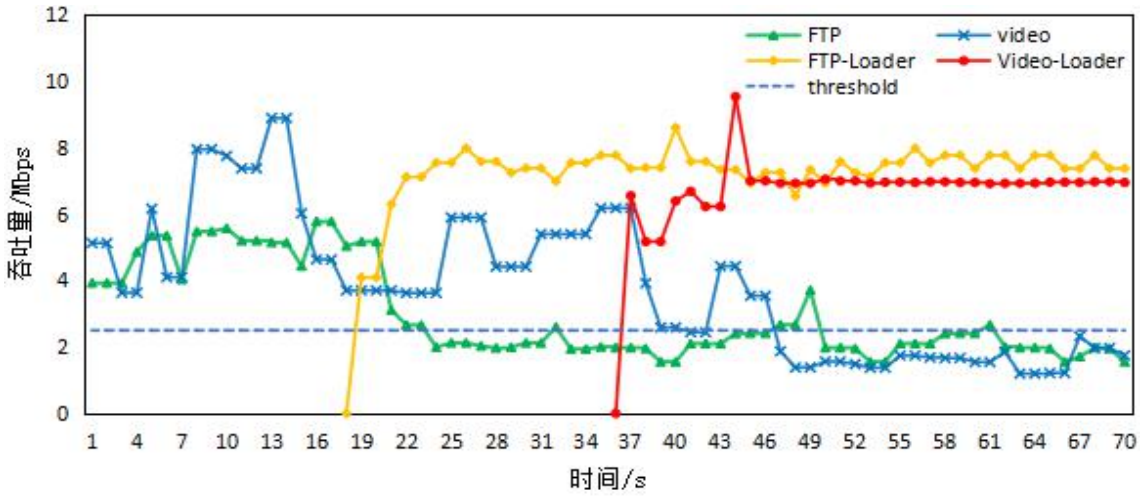


（b）启用 QoS 路由机制的选路情况

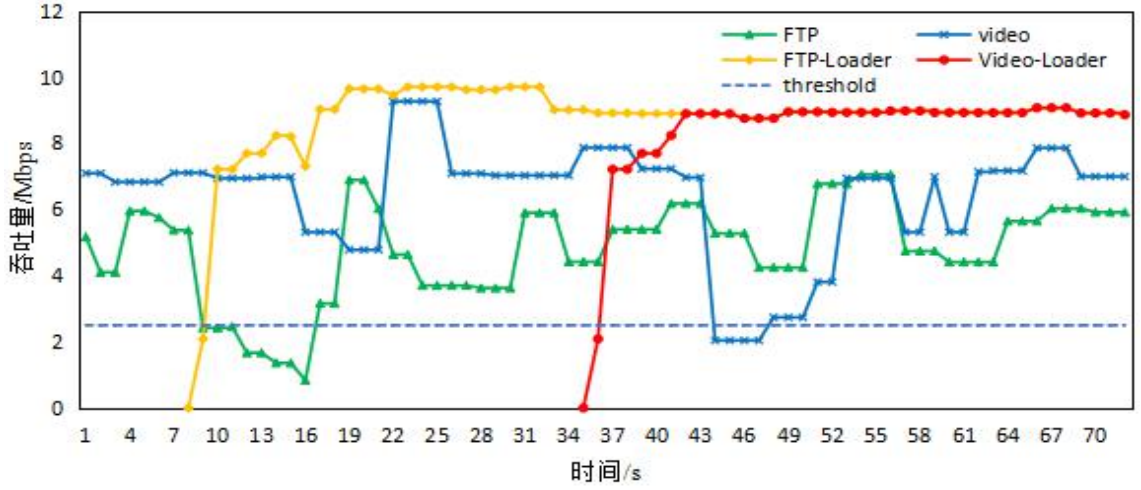
图 4.9 链路拥塞时启用和关闭 QoS 路由机制的选路情况

图 4.9(a)所示的是未启用 QoS 路由机制，而是采用默认最短路径路由机制场景下的选路结果。可以看出，即使链路 *S*6—*S*11 和 *S*1—*S*2 发生了拥塞，控制器也不会为客户端调整路径，仍然按照原始路径转发数据；图 4.9(b)所示的是启用 QoS 路由机制的选路情况，由于 QoS 路由机制检测到了客户端 QoS 受到链路拥塞的影响，所以及时地为客户端调整了路径，视频服务选择了非拥塞路径 *S*1—*S*6—*S*3—*S*8 来传输视频数据，而 FTP 服务选择非拥塞路径 *S*1—*S*6—*S*7—*S*11—*S*12 传输数据。

图 4.10 所示的是客户端吞吐量的变化情况，通过实验统计设定了触发重新选路的最低吞吐量阈值为 2.5Mbps，即当 QoS 路由机制检测到客户端吞吐量小于 2.5Mbps 时，会为客户端重新选路。图 4.10(a)所示的是未启用 QoS 路由机制时客户端吞吐量的变化情况，FTP-Loader 和 Video-Loader 分别在 18s 和 36s 时加入到网络中来，造成了 FTP 客户端和视频客户端的平均吞吐量分别下降到 2.13Mbps 和 2.06Mbps，均低于最低吞吐量阈值；图 4.10(b)所示的是启用 QoS 路由机制后客户端吞吐量的变化情况，当 FTP-Loader 和 Video-Loader 分别在 18s 和 36s 时加入到网络中来时，FTP 客户端和视频客户端的平均吞吐量首先出现了下降，但是当下降到阈值 2.5Mbps 之后，吞吐量又迅速开始回升。FTP 客户端吞吐量在 16s 时开始回升，视频客户端吞吐量在 44s 时开始回升，并都回升到满足其各自 QoS 的需求。因此，从实验结果看到，本章所提 QoS 路由机制能够检测到客户端吞吐量低于阈值并为其重新分配了满足其 QoS 需求的路径，使得客户端的吞吐量得到了回升。



（a）关闭 QoS 路由机制的吞吐量变化情况



（b）启动 QoS 路由机制的吞吐量变化情况

图 4.10 启动和关闭 QoS 路由机制时客户端吞吐量的变化情况

图 4.11 所示的是启用和关闭本章所提出的 QoS 路由机制时视频业务所受 Video-Loader 影响的情况，能够更加直观地反映图 4.10 中视频 QoS 保障的情况。图 4.11(a)所示的是关闭 QoS 路由机制时视频业务受 Video-Loader 影响的情况，由图 4.10(a)可知 Video-Loader 在视频开始传输 36s 时加入到网络中来，造成了视频客户端的平均吞吐量下降到 2.06Mbps，低于最低吞吐量阈值，因此在图 4.11(a)中视频从播放质量正常转变到画质受损，根本无法保障视频传输的 QoS；图 4.11(b)所示的是启用 QoS 路由机制后视频业务受 Video-Loader 影响的情况，由图 4.10(b)可知当 Video-Loader 在 36s 加入到网络中来时，视频客户端的平均吞吐量首先出现了下降，但是当下降到阈值 2.5Mbps 之后，吞吐量又迅速开始回升，视频客户端吞吐量在 44s 时开始回升，并都回升到满足其 QoS 的需求，因此在图 4.11(b)中视频从播放质量正常转变到画质轻微受损，但是马上能够恢复到原有的质量水平，与图 4.11(a)的视频播放质量有显著的差别。

（a）关闭 QoS 路由机制时视频受 Video-Loader 影响情况

（b）启动 QoS 路由机制时视频受 Video-Loader 影响情况

图 4.11 启动和关闭 QoS 路由机制时视频受影响情况

(2) 业务动态接入下的性能分析

在流表溢出的场景下，对本章提出的 QoS 路由机制与文献[27]提出的 QoS 路由机制在业务动态接入的情况下进行了性能的比较，观察业务动态接入的并发度与系统吞吐量之间的关系。图 4.12 所示的是 FTP 业务动态接入的场景：

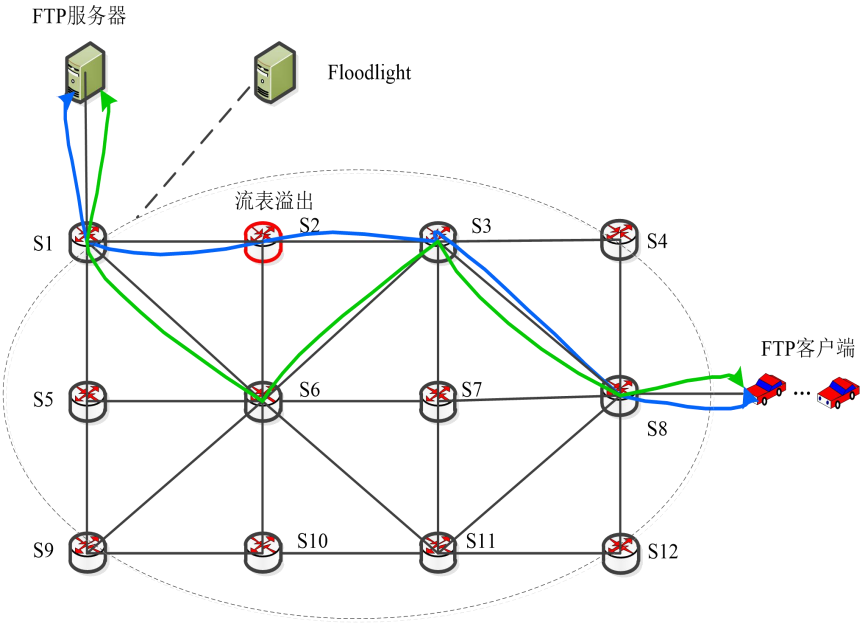


图 4.12 FTP 业务动态接入的场景

本次实验设置了交换机 *S2* 的流表容量为 5，因此在第 6 条 FTP 业务加入进

来时会发生流表溢出。FTP 服务器通过交换机 *S*1 接入网络，通过服务器配置设定了 FTP 传输的平均带宽为 5Mbps，并设置了每隔 10s 向交换机 *S*8 动态接入 2 个 FTP 业务，当 QoS 路由机制启动之后会不停地检测业务的动态接入，QoS 路由机制为每批接入的业务分配合适的路径并监控接入业务的 QoS。由于本章提出的动态 QoS 路由机制能够检测到流表状态，因此选路结果为 *S*1—*S*6—*S*3—*S*8，相反文献[27]提出的 QoS 路由机制选路结果为 *S*1—*S*2—*S*3—*S*8。

图 4.13 所示的是随着业务接入数量的增加，业务并发度与系统吞吐量变化之间的关系。从图 4.13 中可以看出，随着业务数量并发地增加，本章提出的 QoS 路由机制能够保障系统吞吐量并呈增长的趋势，而文献[27]提出的 QoS 路由机制无法保障并发业务动态接入下的系统吞吐量，原因在于 20s 时发生了流表溢出，文献[27]无法及时地为动态接入的业务更换路径而导致系统吞吐量得不到提升。该实验说明了本章提出的动态 QoS 路由机制能够支持业务的动态加入，在保障多个业务 QoS 的同时能够避免流表溢出带来的不良影响。

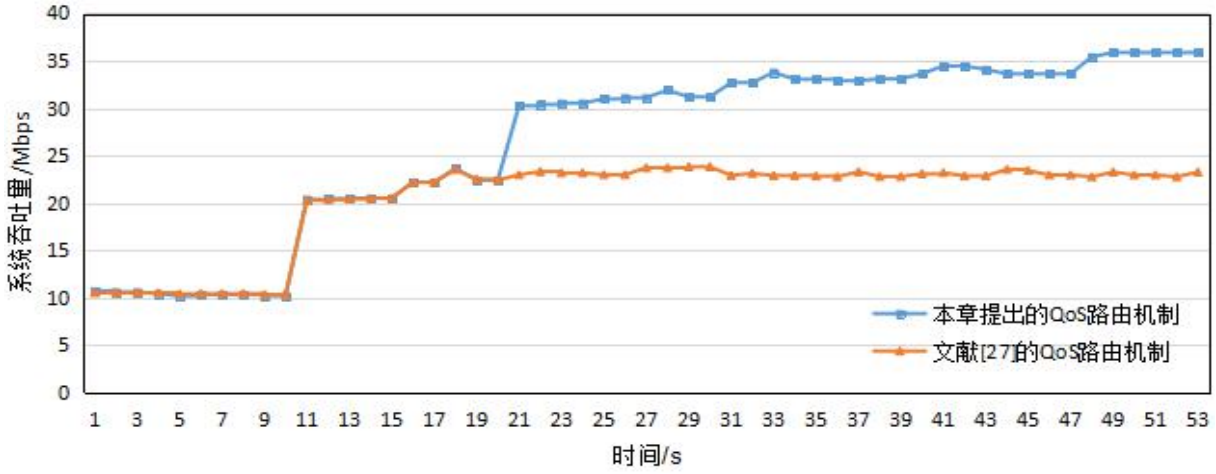


图 4.13 业务并发度与系统吞吐量变化的关系

综上所述，本章提出的动态 QoS 路由机制能够为多业务提供并发的 QoS 路由，支持业务的动态加入和退出，在能够在保障业务 QoS 需求的同时避免了流表溢出带来的影响。

4.5 小结

软件定义的 VANET 架构可以为 VANET 中各类服务提供更有效的支持。本节首先设计了一种面向异构多网接入的软件定义 VANET 架构，完善了现有工作中数据平面的设计，利用数据平面的 SDN 交换机实现异构多网接入的数据转发，接着以此架构为基础提出了一种流表用量感知的动态 QoS 保障框架，并提出了多业务流多约束条件下流表用量感知的 QoS 路由模型，为 VANET 应用服务提供并发的动态 QoS 路由，能够满足多业务对各自丢包、时延和吞吐量的要求，还能够感知交换机流表的使用情况，避免流表溢出对 QoS 路由的影响，进

一步提高了网络 QoS 保障的性能。

在保障应用 QoS 的同时，动态 QoS 路由机制也带来了网络开销。网络开销主要包括 QoS 路由机制启动时采集网络拓扑和链路信息所带来的开销以及启动后定时采集业务 QoS 指标和流表用量的开销。QoS 路由机制启动时，需要采集交换机的基本信息、链路的基本信息和网络拓扑结构，这部分开销的大小会随着交换机和链路数量的增加而增加；当 QoS 路由机制启动后，会定时（秒级）并发地采集业务的 QoS 指标和流表使用情况，这部分的开销不仅与交换机、链路的数量有关，还与系统的并发负载能力有关。

结 论

当前 VANET 中的众多应用主要依赖于传统的 IP 分组网络，该网络的尽力而为服务模式不能为数据传输的带宽、端到端延迟和包到达率等性能提供服务质量保证。SDN 的出现为 VANET 提供了一种新型的网络架构，许多研究者都将 SDN 的架构应用于 VANET 形成软件定义的 VANET 架构，而在软件定义 VANET 架构下，可以考虑更多的网络接入方式，也能够更有效地管理网络的资源，从而保障 VANET 应用服务的 QoS。传统的 QoS 模型不具备对网络信息的采集与分析功能，也无法支持动态实时更新配置信息，也就无法基于网络当前的情况动态地选择合适的路径，因此在软件定义的 VANET 下，基于传统的 QoS 模型无法保障应用服务的动态 QoS。SDN 的出现有效地提高了网络的可编程性和灵活性，能够动态地收集网络当前的参数，实时响应业务需求，从而为应用服务提供动态的 QoS 保障。在 SDN 架构下，为了给应用服务提供 QoS 保障，许多研究人员进行了相关的理论和实践研究，但这些研究存在以下几点不足：其一，面向单业务，并发性差；其二，缺乏对业务动态接入和退出的处理；其三，不具有流表用量的感知能力。针对以上不足，本文的主要贡献如下：

(1) 由于 SDN 路由过程的实现不得不依赖于交换机的流表管理，因此流表策略对 QoS 路由机制性能的影响至关重要，但当前 QoS 路由机制的研究没有考虑流表用量感知的能力，因此本文先通过实验分析了流表溢出对 SDN 控制器和 QoS 路由机制的影响，然后分析了 OpenFlow 网络中流表溢出的现象及原因，最后为了避免流表溢出对 QoS 路由机制产生的影响，本文提出了流表用量感知的 QoS 路由机制 FUQR，FUQR 在路由的过程中能够感知流表状态。实验结果表明，FUQR 能够感知流表溢出交换机，极大地减轻了控制器的负载，同时有效地保障了应用服务的 QoS。

(2) 设计了一种面向异构多网接入的软件定义 VANET 架构，为了给 VANET 应用服务提供并发的动态 QoS 路由，本文设计并实现了流表用量感知的动态 QoS 路由机制，包括 QoS 保障框架与路由模型。流表用量感知的动态 QoS 保障框架允许使用模块化的方式管理网络，并支持业务流的动态加入和退出，同时在此框架的基础上建立了多业务流多约束条件下流表用量感知的 QoS 路由模型，该模型不仅考虑了丢包、时延和吞吐量等链路状态参数，还考虑了业务需求以及交换机的流表使用情况。实验结果表明，本文提出的动态 QoS 路由机制不仅能够满足多业务对各自丢包、时延和吞吐量的要求，还能够感知流表用量，从而避免流表溢出对 QoS 路由机制的影响，进一步提高了网络 QoS 保障的性能。

综上所述，本文以软件定义的 VANET 为应用背景，针对现有 QoS 路由机制的不足，从 SDN/OpenFlow 的架构出发，设计并实现了一种具有流表感知能力的多业务并发的动态 QoS 路由保障机制，从而保障软件定义的 VANET 下多业务并发的动态 QoS。出于时间和能力的不足，本文并未搭建真实的实验平台，此外还有许多后续工作可以开展：

(1) 本文设计的软件定义 VANET 架构结合了异构多网接入方式，同时本文设计的动态 QoS 路由机制是通过软件定义 VANET 架构的核心交换层提供从服务端到 RSU 的动态 QoS 路由，主要考虑了边缘交换机动态变化的任务负载以及动态接入和退出的业务，同时分析了流表策略对 QoS 路由机制性能的影响，以便进一步地保障 QoS 路由机制的性能。但本文没有考虑车辆与车辆之间的动态 QoS 路由，从本文设计的架构中可以看到，服务端可以通过核心交换层传输数据到 RSU，但车辆与车辆之间也可以通过数据共享的方式来减少向服务端的请求，而这种数据的共享也需要动态 QoS 路由机制的保障，但是无线传输场景下的 QoS 路由机制面临着更大的挑战，既需要克服无线传输本身的问题，也需要考虑车辆的移动状态。因此，研究软件定义 VANET 架构下车与车之间的动态 QoS 路由是一个值得研究的问题。

(2) VANET 提供了广泛的服务，这些服务又具有不同的特点，有的实时性较高，但带宽占用量较小，而有的属于带宽密集型服务，因此本文构建的 QoS 路由模型考虑了不同的 VANET 业务之间具有不同的 QoS 需求这一特点。但是在现实中的应用服务具有优先级的区别，比如交通事故播报信息、实时路况信息这一类的优先级就高于天气、新闻一类的信息，如果能够在保障多业务 QoS 的同时还能够保证业务优先级的顺序，那么就更加符合现实生活场景中的业务需求。因此本文需要在 QoS 路由模型的基础上考虑多业务的优先级这一特点，研究面向优先级的多业务多约束 QoS 路由模型及其保障机制。

(3) 在本文提出的软件定义的 VANET 架构的接入层中需要考虑车辆切换接入点的问题，比如车辆在一个 RSU 接入并成功路由，一段时间后车辆移动到另一个接入点，因此为了减少重新路由所带来的开销，可以考虑从前一个 RSU 的边缘交换机直接路由到第二个边缘交换机以保证接入点切换的效率。

(4) 在大规模软件定义 VANET 中，通常会由多个控制器实现分布式控制，因此，本文可以考虑采用多个控制器来扩展控制平面，进而搭建一个更加完善的测试系统，对本文提出的动态 QoS 路由机制进行性能测试。

(5) 本文的实验均为仿真实验，因此本文还需要进一步考虑搭建真实的平台，进一步验证本文提出的动态 QoS 路由机制的有效性。此外，由于 VANET 中更多的是无线接入的方式，因此本文实验也需要进一步测试在无线网络环境下动态 QoS 路由机制的性能。

参考文献

[1] Cunha F D D, Boukerche A, Villas L, et al. Data communication in VANETs: A survey, challenges and applications. Journal of Biological Chemistry, 2014, 274(39): 27605-27609

[2] Kirkpatrick K. Software-defined networking. Communications of the Acm, 2013, 56(9): 16-19

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

[4] Wahab O A, Otrok H, Mourad A. VANET QoS-OLSR: QoS-based clustering protocol for vehicular Ad hoc Networks. Computer Communications, 2013, 36(13):1422-1435

[5] Bradai A, Ahmed T, Benslimane A. ViCoV: Efficient video streaming for cognitive radio VANET. Vehicular Communications, 2014, 1(3):105-122

[6] Oche M, Noor R M, Aghinya J I. Network centric QoS performance evaluation of IPTV transmission quality over VANETs. Computer Communications, 2015, 61(C):34-47

[7] Barba C T, Aguiar L U, Igartua M, et al. A collaborative protocol for anonymous reporting in vehicular ad hoc networks. Computer Standards & Interfaces, 2013, 36(1):188-197

[8] Huang C J, Wang Y W, Chen H M, et al. An adaptive multimedia streaming dissemination system for vehicular networks. Applied Soft Computing, 2013, 13(12):4508-4518

[9] Luan T H, Ling X, Shen X. Provisioning QoS controlled media access in vehicular to infrastructure communications. Ad Hoc Networks, 2012, 10(2):231-242

[10] 陈林星, 曾曦, 曹毅. 移动 Ad Hoc 网络:自组织分组无线网络技术. 电子工业出版社, 2006:511-516

[11] Ku I, Lu Y, Gerla M, et al. Towards software-defined VANET: Architecture and services. In Proceedings of Ad hoc Networking Workshop. Piscataway, NJ: IEEE, 2014: 103-110

[12] Li H, Dong M, Ota K. Control plane optimization in software-defined vehicular Ad hoc networks. IEEE Transactions on Vehicular Technology, 2016, 65(10): 1-1

[13] He Z, Cao J, Liu X. SDVN: enabling rapid network innovation for heterogeneous vehicular communication. IEEE Network, 2016, 30(4): 10-15

[14] Truong N B, Lee G M, Ghamri-Doudane Y. Software defined networking-based vehicular Ad hoc network with fog computing. In Proceedings of International Symposium on Integrated Network Management. Piscataway, NJ: IEEE, 2015: 1202-1207

[15] Richard Yang, 毕军, Guofei Gu. 软件定义网络专题(英). 中国通信, 2014(02)

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

[17] Foundation O N. Software-defined networking: The new norm for networks. ONF White Paper, 2012:1-12

[18] Medved J, Varga R, Tkacik A, et al. OpenDaylight: Towards a model-driven SDN controller architecture. In Proceedings of World of Wireless, Mobile and Multimedia Networks. Piscataway, NJ: IEEE, 2014:1-6

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

[20] Feamster N, Rexford J, Zegura E. The road to SDN: an intellectual history of programmable networks. Acm Sigcomm Computer Communication Review, 2014, 44(2):87-98

[21] Celenlioglu M R, Mantar H A. An SDN-based intra-domain routing and resource management model. In Proceedings of 2015 IEEE International Conference on Cloud Engineering (IC2E). Piscataway, NJ:IEEE Computer Society, 2015:347-352

[22] Huang N F, Liao I J, Liu H W, et al. A dynamic QoS management system with flow classification platform for software-defined networks. In Proceedings of 8th International Conference on Ubi-Media Computing (UMEDIA). Piscataway, NJ: IEEE, 2015

[23] Egilmez H E, Civanlar S, Tekalp A M. An optimization framework for QoS-enabled adaptive video streaming over openflow networks. IEEE Trans on Multimedia, 2013, 15(3): 710-715

[24] Civanlar S, Parlakisik M, Tekalp A M, et al. A QoS-enabled openflow environment for scalable video streaming. In Proceedings of GlobeCom Workshops. Piscataway, NJ: IEEE, 2010: 351-356

[25] Adami D. A network control application enabling software-defined quality of service. In Proceedings of IEEE Int Conf on Communication. Piscataway, NJ: IEEE, 2015: 6074-6079

[26] Caraguay V, Leonardo N, Fern P, et al. Framework for optimized multimedia routing over software defined networks. Computer Networks, 2015, 92(P2): 369-379

[27] Ongaro F, Cerqueira E, Foschini L, et al. Enhancing the quality level support for real-time multimedia applications in software-defined networks. In Proceedings of Int Conf on Computing, Networking and Communications. Piscataway, NJ: IEEE, 2015: 505-509

[28] 孙杰, 李莉, 沈苏彬.一种基于 QoS 和动态负载均衡的路由策略.计算机技术与发展, 2016, 26(11): 188-194

[29] Egilmez H E, Dane S T, Bagci K T, et al. OpenQoS: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In Proceedings of Signal & Information Processing Association Summit and Conf. Piscataway, NJ: IEEE, 2012: 1-8

[30] Adrichem N L M V, Doerr C, Kuipers F A. OpenNetMon: Network monitoring in openflow software-defined networks. In Proceedings of Network Operations and Management Symposium. Piscataway, NJ: IEEE, 2014:1-8

[31] Bueno I, Aznar J I, Escalona E, et al. An OpenNaaS based SDN framework for dynamic QoS control. In Proceedings of Future Networks and Services. Piscataway, NJ: IEEE, 2013: 1-7

[32] Wang Wendong, Qi Qinglei, Gong Xiaoyang, et al. Autonomic QoS management mechanism in software defined network. China Communications, 2014, 11(7): 13-23

[33] Georgopoulos P, Elkhatib Y, Broadbent M, et al. Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In

Proceedings of ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking. New York: ACM, 2013: 15-20

[34] Li L, Wu C. Design and describe REST API without violating REST: A petri net based approach. In Proceedings of IEEE International Conference on Web Services. Piscataway, NJ: IEEE, 2011:508-515

[35] Open Networking Foundation. Openflow switch specification 1.0.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf, 2009-07-19

[36] Open Networking Foundation. Openflow switch specification 1.3.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf, 2012-07-25

[37] Gude N, Koponen T, Pettit J, et al. NOX: towards an operating system for networks. Acm Sigcomm Computer Communication Review, 2008, 38(3):105-110

[38] POX. The POX controller. https://github.com/noxrepo/pox, 2013-11-14

[39] David Erickson. Beacon. https://openflow.stanford.edu/display/Beacon/Home, 2013-02-04

[40] Big Switch Networks. The floodlight documentation. https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Getting+Started, 2016-02-07

[41] Maestro. A system for scalable openflow control. http://www.cs.rice.edu/~eugeneng/papers/TR10‐11.pdf, 2014-02-21

[42] NEC Corporation. Helios: Fully distrabuted openflow controller platform. http://groups.geni.net/geni/attachment/wiki/GEC9Demo Summary/Helios_Open Flow_Controller_GEC9_Demo.pdf, 2014-03-13

[43] Ryu. Welcome to RYU the network operating system(NOS). https://ryu.readthedocs.io/en/latest/, 2014-01-01

[44] Mininet. The mininet documentation. https://github.com/mininet/mininet/wiki/Documentation, 2016-08-20

[45] Curtis A R, Mogul J C, Tourrilhes J, et al. DevoFlow: scaling flow management for high-performance networks. In Proceedings of ACM SIGCOMM 2011 Conference on Applications, Technologies,

Architectures, and Protocols for Computer Communications, Toronto, On, Canada, August. DBLP, 2011:254-265

[46] Zarek A, Ganjali Y, and Lie D. Openflow timeouts demystified. University of Toronto, Toronto, Ontario, Canada, 2012

[47] Wang Zheng, Crowcroft J. Quality-of-service routing for supporting multimedia applications. IEEE Journal on Selected Areas in Communications , 1996, 14(7): 1228-1234

[48] Sourceforge. Ipsolve: Mixed integer linear programming (MILP) solver. https://sourceforge.net/projects/lpsolve/, 2016-09-14

[49] VideoLAN. VLC media player. http://www.videolan.org/vlc/, 2017-05-06

[50] Apache. Apache FtpServer. http://mina.apache.org/ftpserver-project/, 2017-05-06

附录 A 攻读硕士学位期间发表的学术论文

论文发表：

[1] 付彬, 查理佳, 李仁发, 肖雄仁. 软件定义的 VANET 下流表用量感知的动态 QoS 路由机制. 计算机研究与发展（已录用）

附录 B 攻读硕士学位期间所参与的项目

[1] 国家自然科学基金项目[61502162]：软件定义的VANET动态QoS路由及流表更新机制研究

[2] 中央高校基本科研业务费[531107040289]：软件定义的VANET跨层服务保障机制研究

致　谢

时光匆匆如流水，转眼便是大学毕业时节，春梦秋云，聚散真容易。我在湖南大学这个充满底蕴深厚的千年学府度过了三年的研究生生活，最大的收获不只是那一本学历证书，而是老师们的谆谆教诲和同学朋友们的无私帮助。三年的研究生涯，让我结识了诸多良师益友，一起探寻真理，这是我一生难以忘记的美好时光。值此论文完成之际，谨向所有帮助关心我的老师、同学、朋友和家人表示我由衷的感谢！

首先我要感谢我的导师付彬老师。在三年的硕士研究生生活中，付老师作为我们的导师，其认真负责，精益求精的工作作风对我产生了深远的影响。付老师会定期与我们分享学术前沿，监督我们的学习进展，在我们研究工作出现瓶颈的时候给予我们正确的指导与积极的鼓励，尤其对我论文的撰写给予了巨大帮助，从论文的构思、实验到写作都不遗余力地给予我指导。除此之外，付老师在生活上也给予我们贴心的照顾，凡事都为学生着想，让我们在轻松的学习气氛下完成了研究生期间的学习与工作。

其次我要特别感谢李仁发教授。在硕士研究生这三年的学习生活中，我体会到了李老师那宽广的胸襟、渊博的知识、严谨的治学态度和敢为人先的精神。李老师会定期与我们分享学术前沿，监督我们的学习进展，除此之外李老师还特别注重我们实践能力的提升，为我们提供了许多到企业参观实习的机会，让我们将所学与实际相结合，提升了我们求职的竞争力。谨向李老师致以最诚挚的敬意和深深的谢意。

感谢项目组吴迪老师，李蕊老师，肖玲老师对我的指导！感谢实验室的师兄师姐们，特别是黄晶师兄、吴武飞师兄、黄一智师兄、白洋师姐，在平常的学习生活中为我解答疑惑、讲解问题。感谢这三年来一起在实验室奋斗的同窗好友，秦凯强、朱立民、刘四平、杨竞、卢兴沄、李鲜、吕超、马孔强、李万里。在硕士研究生期间，我们互相学习、取长补短、团结互勉、共同进步，一起度过了这难忘而短暂的研究生涯。

特别感谢我的父母，谢谢你们对我无私的爱，是你们的关心、鼓励和支持让我取得今天的成绩。我会带着你们的期许，带着你们的爱，继续奋斗。

最后，谨向审阅本论文及答辩组的老师们致以诚挚的谢意。由于本人水平有限，文中难免有不足之处，敬请各位老师批评、指正。

查理佳

2017 年 4 月 25 日于长沙