doi:10.11959/j.issn.1000-436x.2017034

PriQoS：基于 SDN 的区分优先权流控机制

孙志[1,2]，陈鸣[1]

（1. 解放军理工大学指挥信息系统学院，江苏 南京 210007；2. 海军航空工程学院青岛校区信息管理中心，山东 青岛 266041）

摘 要：尽力而为的 IP 网络难以满足日益提高的服务质量需求，亟需研发一种兼容现有 IP 软硬件资源的 QoS 控制机制。为此，提出了一种基于软件定义网络的区分服务优先权的流控机制 PriQoS。当网络资源紧缺时，使用基于业务流综合优先权的多队列带宽分配算法来满足高优先权流的带宽需求；PriQoS 部署在网络边缘，无需修改端系统应用程序和核心网路由器，具有易于部署、与 IP 应用和设备兼容的优点。实验结果表明 PriQoS 机制能够优先保证高优先权业务的服务质量。

关键词：软件定义网络；流控；综合优先权；服务质量；层次分析法

中图分类号：TP393 **文献标识码**：A

PriQoS: priority-differentiated flow control mechanism based on SDN

SUN Zhi[1,2], CHEN Ming[1]

(1. College of Command Information Systems, PLA University of Science & Technology, Nanjing 210007, China;
2. Information Management Center, Naval Aeronautical Engineering Institute Qingdao Branch, Qingdao 266041, China)

Abstract: It was difficult to meet the increasing demand of quality of service (QoS) for the best-effort IP network, so it was urgent to develop a kind of QoS control mechanism that was compatible with the existing IP software and hardware resources. Therefore, a priority-differentiated control mechanism based on the software defined networking called PriQoS was proposed. When the network resource was scarce, a multi-queue bandwidth allocation algorithm based on the integrated priority of service flow was used to meet the bandwidth requirement with high priority flows. PriQoS was deployed on the network edge, which was no need to modify applications in the end systems and routers in the core network. The mechanism has advantages such as easy to deploy and being compatible with the IP applications and devices. Experimental results show that the PriQoS mechanism can guarantee the QoS of higher priority services.

Key words: SDN, flow control, integrated priority, QoS, AHP

1 引言

网络应用的快速发展对网络通信服务质量(QoS, quality of service)提出了越来越高的要求。根据应用的需求、目的和场景不同，各种网络系统所追求的性能指标也不尽相同，在某些场景中可能会要求依据业务流的多维属性（如用户级别、应用重要性或业务流的实时性要求等），对不同的业务流实现不同的 QoS 保证。如对指挥网来说，当发生战争或突发灾害等紧急情况时，网络基础设施可能部分受损而导致可用通信资源减少，然而对这些热点区域的网络通信量却往往急剧增加，如果不进行合理管控，容易导致一些有关指挥或救援等关键业务的服务质量无法得到保证。在这种情况下，根据业务流的多维属性，实现基于优先权的区分 QoS 保证就显得尤为重要。

收稿日期：2016-06-17；**修回日期**：2016-11-23

基金项目：国家重点基础研究发展计划（“973”计划）基金资助项目（No.2012CB315806）；国家自然科学基金资助项目（No.61379149, No. 61402521）；江苏省自然科学基金资助项目（No.BK20140070, No.BK20140068）；江苏省未来网络科技计划基金资助项目（No.BY2013095-1-06）

Foundation Items: The National Basic Research Program of China (973 Program) (No.2012CB315806), The National Natural Science Foundation of China (No.61379149, No.61402521), The Natural Science Foundation of Jiangsu Province (No.BK20140070, No.BK20140068), Jiangsu Future Network Innovation Institute Research Project on Future Networks (No.BY2013095-1-06)

虽然 QoS 问题一直是网络通信领域研究的热点问题之一，但因为传统的 IP 协议使用“尽力而为”的网络传输模式，该模式更注重网络的传输效率和可达性，却难以实现通信业务的 QoS 保证[1]。为克服 IP 协议体系的不足，近年来人们陆续提出了一些新的网络体系和技术，其中，软件定义网络（SDN，software defined networking）技术[2]吸引了越来越多的关注，其具有控制平面与数据平面分离、控制平面掌握全网拓扑信息、基于流转发等特点，为克服传统 IP 网络在 QoS 方面的不足提供了新的思路。然而在相当长的一段时间内，SDN 技术并不能完全取代传统的基于 IP 路由的现有网络。一方面，SDN 由于采用集中式的控制平面，扩展性受到限制；另一方面，因特网现有的巨大软硬件资源和投资需要得到保护，这就要求 SDN 技术必须是可演化的，而且这种演化不仅是在一个局域网中不同设备和协议间的兼容，也可以是基于 OpenFlow 技术的多个子网间能够通过已有 IP 骨干网进行通信[3]，所以研究利用 SDN 技术解决现有 IP 网络中的资源管控机制具有非常重要的现实意义。

2 相关研究

网络资源管控是为了对有限的网络资源进行有效的管理分配，实现网络高效通信和 QoS 保证，从而使各种业务应用能够满足其特定的通信需求。传统 IP 网络无连接和尽力而为的特性导致其较难对用户提供有效的 QoS 保证能力。为了在现有 IP 网络体系结构上实现一定的 QoS 功能，IETF 制定了多种的 QoS 解决方案，主要有综合服务(IntServ[3], integrated service)、区分服务(DiffServ[4], differentiated service)和 QoS 路由[5] (QoS-based routing)等服务模型和机制。这些机制均是在传统 IP 协议的基础上进行修补式实现，通常需要对传统 IP 核心网设备进行修改，在大规模网络内应用时部署较为困难，且业务流的控制粒度与协议开销和扩展性方面通常存在较为突出的矛盾，所以目前这些机制并没有得到大规模推广应用。

近年来兴起的 SDN 技术有很多的潜在优势，使其能够非常方便地进行网络资源管控，实现 QoS 功能。1）SDN 采用集中式管控，控制器可以通过南向接口对网络资源信息进行查询，掌握全局拓扑信息和资源使用情况。2）SDN 的控制平面与数据平面分离，控制平面可编程，所以可按实际需求开发相应的资源管控应用，并结合当前网络态势，对网内资源和业务流量实现动态管控。3）SDN 的数据平面基于流的匹配转发模式尤其适合进行高效的资源管控，无需以分组为单位进行处理，从而大大降低了资源管控的开销。4）OpenFlow 各版本的规范提供了越来越完善的 QoS 支持，在 1.0.0 规范中提供了对端口队列的支持；在 1.3.0 中增加了计量表（meter table），可以对每流进行计数，从而实现多种简单的 QoS 业务（如限速），也可以结合端口队列来实现复杂的资源管控机制。5）在流表匹配字段中允许匹配 IP 分组中 ToS 字段服务类型（type of service），并可对该字段进行修改，从而可以方便地与 DiffServ 兼容。

基于 SDN 的上述优势，很多学者对 SDN 中的资源管控机制进行了大量研究，如 Bueno 等[6] 提出了一种在 SDN 中基于网络即服务（NaaS, network as a service）的软件框架方案 NCL，可对资源管控策略进行灵活配置，实现了对交互式应用的 QoS 保障。Egilmez 等[7] 提出了一种 SDN 中的对多媒体应用的 QoS 保障机制 OpenQoS，将业务流区分为多媒体流和数据流，其中，对多媒体流使用动态 QoS 保障路由算法。这些方案在选择需要保障的业务流时，只是针对应用场景的特定需要，对某一类业务进行保障，没有考虑业务流的多维属性。Akella[8] 提出了一种 SDN 中基于云端用户优先级和应用类型的带宽保证资源分配机制，将业务流分为 QoS 流和尽力而为的流，分别选择不同的路由方案，其中 QoS 流的路由选择过程考虑了带宽、时延和路径长度。类似地，Tomovic 等[9] 提出了 QoS-aware 算法，通过对 QoS 流与非 QoS 流进行不同的路由，实现 QoS 流量控制的目的，其中，QoS 路由将带宽作为权重代入 Dijkstra 算法中求得路径。这些方案通常以 SDN 网络中的 QoS 路由为研究对象，通过测量网络内的多个性能参数，能够较好地实现服务质量保证，但当网络规模较大时，为实现精确资源管控，必然会带来较大的性能测量开销，所以这些机制的扩展性受到限制。

如上所述，尽管目前已在 SDN 中的资源管控和 QoS 保障方面取得了很多研究成果，但在平衡资源管控机制的扩展性与实现开销、尽可能减少对核心网和终端设备的改动、如何更为科学地识别确定需保障业务流等方面都存在进一步研究的必要性和紧迫性。

3　PriQoS 机制的功能架构

如图 1 所示，当使用 OpenFlow 的企业网之间通过广域网进行互联时，企业网中的带宽通常较为充足，而广域网的带宽则相对较低，这种带宽的不匹配导致在边界交换机处容易形成性能瓶颈。

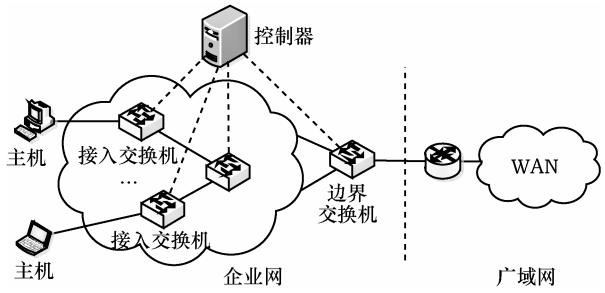

图 1　OpenFlow 企业网与广域网互联

本文提出了一种在这种应用场景中的基于业务流综合优先权的 QoS 保证机制 PriQoS（priority based QoS）。PriQoS 利用 SDN 数控分离的特性，通过控制器对业务流的多维属性进行识别和映射，计算出该业务流的综合优先权（integrated priority），并根据该优先权，在边界交换机上对带宽资源进行动态管控，优先适配高优先权业务流的带宽需求，从而保证重要业务在资源紧缺时也能够正常通信。

图 2 描述了 PriQoS 闭环自适应动态控制模型，在子网内的 OpenFlow 接入交换机上实现流优先权的标记；在边界交换机上实现流的优先权队列；在 SDN 控制器中，实现队列状态采集、需求感知、带宽适配和应用规则 4 个功能模块。通过各交换机和功能模块间的协作，形成一个闭环自适应的动态控制过程。

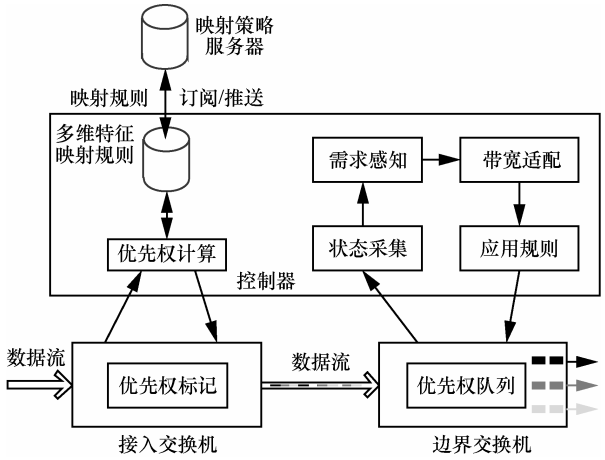

图 2　PriQoS 闭环自适应动态控制模型

4　基于业务流多维属性的综合优先权

要实现基于优先权的业务流管控，首先需要能够识别和区分业务流属性。当某业务流的第一个分组由于无法匹配接入交换机流表时，该分组将被发送至控制器。控制器通过对分组的解析，可感知该分组的某些属性，如通过 IP 地址识别用户级别，通过应用端口和高层协议识别应用类别。控制器从映射策略服务器下载业务流特征映射规则库，并据此对业务流属性进行量化。PriQoS 根据用户级别、应用重要性和流量类型 3 个属性对流进行区分，并分别赋予不同的权重，据此计算出该流的综合优先权。

4.1　业务流属性的识别和量化

PriQoS 将业务流按识别属性分别进行量化赋值。其中，用户级别分为高、中、低这 3 个类别；应用重要性分为紧急、重要、一般、次要、不重要这 5 个类别；流量类型分为网控流量（如 Telnet、DNS）、严格实时流量（如 VoIP）、自适应延迟实时流量（如 IPTV、VOD 等）、弹性业务（如 HTTP、FTP 等）这 4 个类别[10]。

需要说明的是，因为要对业务流的各属性值分别进行归一化处理，所以量化数值只需能够区分类别高低即可。用户级别、应用重要性和流量类型均属于效益型属性，即量化数值越高，该流对应的优先权也趋向于越高。

4.2　业务流综合优先权的计算

控制器在收到 packet-in 分组后，对其进行解析，识别出该业务流各属性所对应的量化数值，并进行归一化处理。例如将用户级别属性的量化数值 u 使用式(1)进行归一化处理。

$$V_{\text{user}} = \frac{u - u_{\min}}{u_{\max} - u_{\min}} \tag{1}$$

其中，u 为业务流特征映射规则库中记录的当前业务流用户级别属性所对应的量化数值，$u_{\min}$ 为用户级别中最小的量化数值，$u_{\max}$ 为最大量化数值，归一化后的 V_{user} 在(0,1]内。类似地，对业务流的应用重要性和流量类型属性也进行归一化处理，得到 V_{app} 和 V_{traffic}。这 3 个值越大，说明该流的优先权越高，该业务流的服务质量越应该优先得到保证。但是，具体的综合优先权数值还需要进行加权运算得出，即根据归一化的 V_{user}、V_{app} 和 V_{traffic}，使用式(2)计算该业务流的综合优先权 P。

$$P=\omega_1 V_{\text{user}}+\omega_2 V_{\text{app}}+\omega_3 V_{\text{traffic}} \tag{2}$$

其中，$\omega_i(i=1,2,3)$为各属性在优先权计算中的权重，体现了业务流中各属性对优先权计算的贡献度。

4.3 基于 AHP 的业务流属性权重因子的确定

在式(2)中，业务流各属性已经进行了归一化处理，关键在于如何确定各属性的权值ω_i，然后计算出该流的优先权数值，进而生成相应的流表，对该流的分组进行优先权标记。权重因子ω_i可通过多种方式确定，最简单的方法可根据实际网络通信需求通过直觉确定，如根据 3 个属性的重要程度，分别确定为 0.3、0.5、0.2，表示应用重要性属性对优先权贡献度最大，但该种方法较为粗糙，受决策者主观影响较大。为了较为精确地确定各属性在优先权计算中的权重ω_i，本文使用了模糊综合评价中的层次分析法（AHP，analytic hierarchy process）[11] 进行确定，其过程如下。

1) 建立确定优先权的层次结构模型，如图 3 所示。该模型所要解决的主要问题是确定业务流的多维属性 V 对于业务流优先权 P 的贡献度权重。

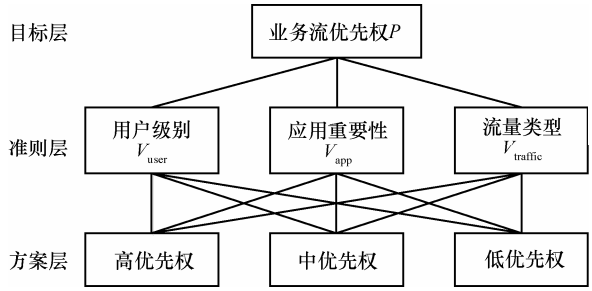

图 3 业务流优先权层次结构模型

2) 建立判断矩阵 $\boldsymbol{A}=(r_{ij})_{n\times m}$，其中，各元素 r_{ij} 表示第 i 个属性与第 j 个属性相比，对于业务流优先权的重要性之比。表 1 中列出了 Saaty 给出的 9 个重要性等级及其赋值，即 1~9 标度法。将各属性两两比较的结果构成的矩阵即为判断矩阵。显然，$r_{ij}>0$，$r_{ii}=1$ 且 $r_{ij}=\dfrac{1}{r_{ji}}$

表 1 Saaty 标度

r_i 比 r_j	量化值
极端重要	9
强烈重要	7
明显重要	5
稍微重要	3
同样重要	1
插值	2, 4, 6, 8

根据网络应用场景的实际需求，可由网络决策者和专家对各属性的重要性进行预估，并进行两两比较，建立模糊互补判断矩阵 $\boldsymbol{A}$。如根据该网络的实际应用特点，确定用户级别权重略低于应用重要性（$r_{12}=\dfrac{1}{2}$），却稍高于流量类型（$r_{13}=3$），而应用重要性明显高于流量类型（$r_{23}=5$），据此建立模糊互补判断矩阵 $\boldsymbol{A}$。

$$\boldsymbol{A}=\begin{bmatrix} 1 & \dfrac{1}{2} & 3 \\ 2 & 1 & 5 \\ \dfrac{1}{3} & \dfrac{1}{5} & 1 \end{bmatrix}$$

3) 根据判断矩阵的最大特征值，得到对应的特征向量，并进行归一化处理。上例中判断矩阵 $\boldsymbol{A}$ 的最大特征值$\lambda_{\max}=3.004$，对应的特征向量为 $[0.463\quad 0.871\quad 0.164]^{\mathrm{T}}$，归一化后得到权值向量$\boldsymbol{\omega}=[0.309\quad 0.581\quad 0.110]^{\mathrm{T}}$。

4) 对构建的判断矩阵 $\boldsymbol{A}$ 进行一致性检验。本例中，$CI=0.002$，当 $n=3$ 时，随机一致性指标 $RI=0.58$，$CR=0.003\,4<0.1$。说明判断矩阵 $\boldsymbol{A}$ 的不一致程度在容许范围内有满意的一致性，通过一致性检验，所以可用其归一化特征向量$\boldsymbol{\omega}$作为权重向量。

使用上述方法，可确定优先权计算式(2)中各权重因子的数值。在本例及实验评估中，用户级别、应用重要性、流量类型 3 个属性的权值分别取为$\omega_1=0.31$、$\omega_2=0.58$、$\omega_3=0.11$。

显然，使用 AHP 方法确定权重因子比通过直觉方式确定更为精确，尤其当需要考虑的业务流属性维度较多时更能体现其优势。为减小单一决策者构造判断矩阵时的主观误差，还可结合基于 OWGA 算子的多属性群决策[12]等方法加以完善。

需要说明的是，通过 AHP 方法确定权重的算法并不需要频繁运行，只需要在网络初始化时运行一次即可，而且也无需在控制器上运行，而是在映射策略服务器上运行，然后将计算出的权重结果推送至各控制器，所以其复杂度并不会对控制器性能造成影响。

5 PriQoS 机制的实现

PriQoS 的实现利用了 SDN 能够由控制平面对数据平面进行灵活定制，并可以根据资源状态变化情况进行动态调整的优势。为了实现业务流按优先

权区分转发，由控制器分别对接入交换机和边界交换机下发不同的流表，实现不同的功能。在接入终端交换机中下发实现优先权标记功能的流表，对业务流标记不同的优先权并通过指定端口发送出去；在边界交换机下发建立端口队列和根据业务流优先权 P 进入对应队列的流表，并启动一个定时器，用于定期采集端口各队列的状态，实现多队列带宽动态分配。

5.1　映射策略的维护与推送

当多个 OpenFlow 子网间进行通信时，需要在不同域控制器之间进行业务流特征映射规则的同步，以保证优先权的全局一致性。在 PriQoS 机制中，在控制器之上设置了映射策略服务器来维护业务流特征信息库 traffic_MIB。控制器与映射策略服务器之间采用订阅/推送机制，当网络中有新的控制器加入或 traffic_MIB 发生变化时，映射策略服务器主动向各控制器推送最新的规则集合，从而保证全网映射规则的一致。

5.2　优先权标记

当子网内的终端将业务流首个分组发送到接入交换机时，由于交换机内没有与该分组相匹配的流表项，所以被发送至控制器，在控制器上触发 packet-in 事件。控制器通过查询业务流特征映射规则库，获得该流的各属性量化值，计算出该流的业务流的优先权 P。根据划分的优先权队列数目，将该数值映射成相应的 ToS 数值。由控制器给接入 OpenFlow 交换机下发“标记优先权并转发至目的端口”的流表项（使用 OpenFlow 的 ofp_action_nw_tos 动作），然后该流所有后续到达分组的 ToS 字段均被标记优先权级别，并转发至相应的出端口。

5.3　业务流按优先权进入边界交换机队列

PriQoS 对流量的动态管控基于交换机端口队列技术实现。根据所设置的优先权数量，在边界交换机的网关端口上建立相同数量的队列。当某个需要发送到广域网的业务流到达边界交换机时，该交换机上没有相匹配的流表，其首个分组被发送至控制器。由于该分组首部的 ToS 字段已经被标记优先权，控制器根据该数值，为业务流生成进入相应端口队列的流表项（ofp_action_enqueue），并下发至边界交换机，随后到达的分组即按照该流表项，进入出端口对应的优先权队列。

5.4　队列状态探测和带宽需求感知

为保证高优先权队列的传输带宽，控制器需按周期 T 向边界交换机发送高优先权队列的状态请求分组 ofp_queue_stats_request。交换机收到该请求后，会立即向控制器回复 ofp_queue_stats 分组，报告相应的端口队列信息，如已传输分组数 tx_packets、已传输字节数 tx_bytes 和因缓存溢出而丢弃的分组数 tx_errors 等。

控制器接收到该状态信息后，触发带宽需求感知模块。通过相邻周期内的 tx_errors 字段和 tx_packets 字段分别计算出该周期内因缓存溢出而丢弃分组数量 N_{error} 和实际传输的分组数量 N_{tx}，两者之和即为本轮询周期内实际到达该队列的分组数量；通过 tx_bytes 字段得到本周期内传输的字节数 $Bytes_{\text{tx}}$。按式(3)计算出该队列实际需要的带宽 BW_{demand}，单位为 Mbit/s。

$$BW_{\text{demand}} = \frac{(N_{\text{error}} + N_{\text{tx}})}{N_{\text{tx}}} \frac{8Bytes_{\text{tx}}}{10^6 T} \tag{3}$$

再通过式(4)计算实际分配带宽 $BW_{\text{high\_priority}}$ 与所需带宽 BW_{demand} 之间的差值 Δ_{BW}。

$$\Delta_{BW} = BW_{\text{demand}} - BW_{\text{high\_priority}} \tag{4}$$

Δ_{BW} 的正负反映了高优先权队列所分配的带宽是否能够满足业务流的带宽需求。$\Delta_{BW} \geqslant 0$ 说明为该队列分配的带宽低于当前所需带宽，需要增加带宽分配；相反，$\Delta_{BW} < 0$ 则说明为该队列分配的带宽超出了当前所需带宽，可以降低带宽分配。

5.5　动态带宽分配算法

PriQoS 的应用场景为：当网络资源不足时，优先保证最高优先权队列的传输带宽，即 PriQoS 承诺保证最高优先权队列的服务质量，较低优先权队列的服务质量只能尽力保证。所以，为保证高优先权业务流的需求能够迅速得到满足，同时减小因带宽频繁调整导致业务流传输速率的波动，PriQoS 机制在增大带宽和减小带宽的计算中采用了不同的触发条件：当高优先权队列带宽不足时，优先抢占最低优先权队列的带宽；反之，当高优先权队列带宽有冗余时，优先将冗余带宽分给其他的次高优先权队列，能够满足其带宽需求后，再分给低优先权的队列。

1) 只要 $\Delta_{BW} \geqslant 0$，说明高优先权带宽无法满足传输速率要求，就立即从目前能够满足带宽差值的最低优先权队列中，抢占大于$|\Delta_{BW}|$的最小带宽（按单位步长取整，单位步长为端口物理带宽的 5%），将其增加到高优先权队列，如当端口的物理带宽为 100 Mbit/s 时，设定带宽调整的单位步长 B_{step} = 5 Mbit/s，

当 Δ_{BW}=12 Mbit/s 时，即高优先权队列的带宽缺口有 12 Mbit/s，则取带宽增加值为 15 Mbit/s。

2) 只有当 $\Delta_{BW}<0$ 且$|\Delta_{BW}|>B_{\text{step}}$时，才减小队列的分配带宽，减小的幅度为小于 Δ_{BW} 的最大带宽（按单位步长取整）。如当 $\Delta_{BW}=-12$ Mbit/s 时，即高优先权队列有冗余带宽 12 Mbit/s，取带宽减小值为 10 Mbit/s。按优先权高低，将削减的带宽按优先权增加到无法满足带宽需求的其他队列中。

为查询端口队列状态，控制器中的状态采集模块将轮询边界交换机的端口队列状态，以便判断为高优先权队列分配的带宽能否满足传输要求。轮询周期越小，队列状态采集的准确度越高，但开销也会相应增高。考虑到信息准确度与开销之间的折中，PriQoS 采用了动态的状态轮询周期，即如果在上个周期内高优先权队列的分配带宽发生了变化，则将轮询周期由 T 缩短为 T_1，即 T_1 后再发送一个队列状态请求，以便及时了解上次带宽分配是否能够满足需求；相反，如果本周期高优先权队列分配带宽未发生变化，则维持轮询周期 T 不变。轮询周期 T 和 T_1 的具体取值，需根据网络的流量特征进行分析和优化。

上述队列状态探测和带宽分配的逻辑过程可用图 4 所示的流程描述。

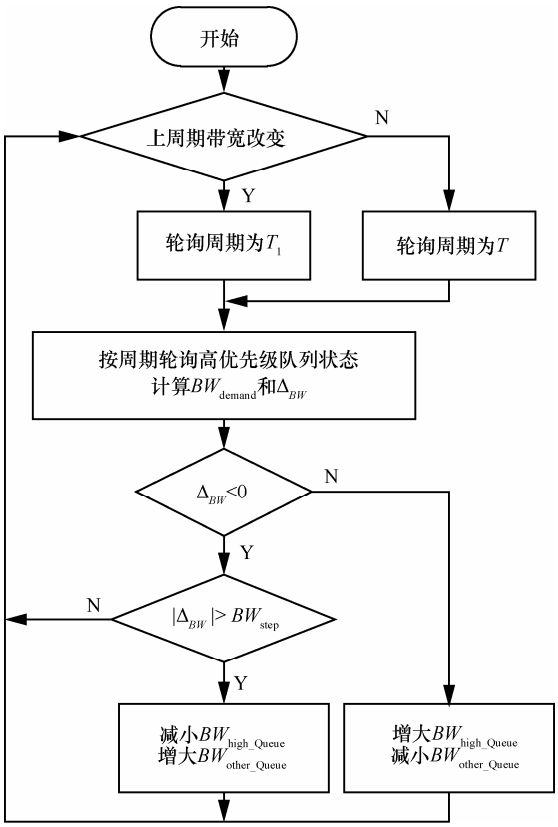

图 4 队列状态探测和带宽分配流程

5.6 队列带宽分配算法的实施

由于实验室中的硬件 OpenFlow 交换机无法动态从控制器接收调整队列限速的 ovs-vsctl 命令，从而无法实现动态改变队列带宽的功能，所以 PriQoS 采用基于 Linux 内核模块的 Open vSwitch[13]作为边界交换机。

Open vSwitch 中的 QoS rate-limiting 功能是采用令牌桶（Token-Bucket）机制进行的。该机制不仅能够满足动态限速的功能需求，还可以处理队列缓存的资源回收和重复使用的问题，故 PriQoS 在实现时，同样采用了基于 Linux-HTB 的带宽管控机制。在 Linux-HTB 队列调度机制下，令牌桶 [14]（HTB, hierarchical token bucket）中的令牌以一定速率（即为队列设定的传输速率）填充桶。每一个令牌表示可以发送一定数量的数据，没有获得令牌就不能发送数据。如果桶内没有足够的令牌数据就必须等待，待有足够的令牌时再发送，或者数据被丢弃，从而实现对流的限速和过滤整形。

在 PriQoS 中，控制器在边界交换机上按优先权数量建立同样数量（3 个）的优先权 HTB 队列，如果新的带宽分配方案与上轮方案不同，则将新的带宽分配方案通过 ovs-vsctl 命令动态下发给边界 OpenFlow 交换机，命令格式为

ovs-vsctl [--OPTION] -- set port <port> qos= @newqos

-- --id=@newqos create QoS type= linux-htb queues:<queueid>=@newqueue [queues:<queueid>= @newqueue1]

-- --id=@newqueue create queue other-config: min-rate=<minrate> other-config: max-rate= <maxrate>

[-- --id=@newqueue1 create queue other-config: min-rate=<minrate> other-config: max-rate= <maxrate>]

其中，QoS type 表示使用的队列类型，指定为 linux-htb，各队列的 min-rate 和 max-rate 相等，均设定为 PriQoS 计算出的各队列带宽速率，从而使各队列的实际传输带宽严格匹配 PriQoS 动态计算出的带宽分配方案。

6 实验与评估

为验证 PriQoS 机制的效用，在 Mininet 2.2.1[15]中创建了如图 5 所示的实验系统，并对 PriQoS 进行了实验和评估分析。实验主要通过各优先权队列的传输带宽和分组溢出丢弃率等指标，对不同优先

权业务流的带宽感知和分配机制进行了分析；另外还对计算业务流综合优先权时的控制器开销进行了测试分析。

6.1　实验环境

图 5 中 S_1、S_2、S_3、S_4 为 4 台 OpenFlow 交换机，其中，S_1 和 S_4 作为接入交换机，实现为各接入终端发出的流标记 ToS 优先权功能；S_2、S_3 作为进行带宽管控的边界交换机，实现交换机端口队列划分、流的入队与调度等功能。该服务器安装 Ubuntu 14.04 LTS，CPU 为 Intel Core i7-4790，主频为 3.60 GHz，内存为 8 GB。通过 ovs-vsctl 命令将端口 S_2-eth_2 的物理带宽设置瓶颈链路，带宽为 100 Mbit/s，其他链路带宽均为 1 000 Mbit/s。选用 POX[16]控制器（carp 版本），控制器与交换机之间使用 OpenFlow1.0.0[17] 通信。

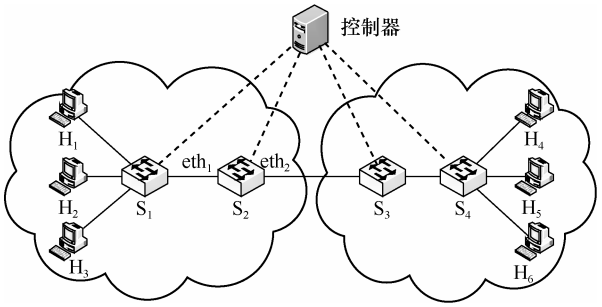

图 5　PriQoS 系统测试环境

为降低实现的复杂性，且不失一般性，在测试 PriQoS 对高优先权业务流的 QoS 保证能力时，对实验过程进行如下简化。

1) 省略优先权映射的过程，直接规定 H_1 和 H_4 产生的流量属于高优先权，H_2 和 H_5 为中优先权，H_3 和 H_6 为低优先权。

2) 带宽调整步长 B_{step}=边界交换机的端口链路带宽×5%。

在 S_2 的端口 eth_1 和 eth_2 上实现优先权队列，使用 iperf 产生 4 条不同速率的 UDP 测试长流，分别为流 1（H_1→H_4:30 Mbit/s）、流 2（H_1→H_4:40 Mbit/s）、流 3（H_2→H_5:30 Mbit/s）和流 4（H_3→H_6:30 Mbit/s），iperf 流量发生器的各项设置均采用默认值，各条流进入网络的时间如图 6 所示。由于 S_2-eth_1 和 S_2-eth_2 的物理带宽为 100 Mbit/s，所以在 200~400 s 内，4 条流对带宽的总需求接近或超出链路能力，重点观察该段时间内 PriQoS 对高优先权业务流的带宽保证能力。实验中将 PriQoS 机制与 L_2 learning 机制（不具备 QoS 功能的二层自学习交换机）进行了对比。

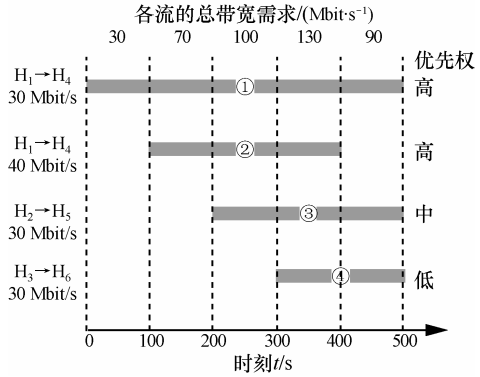

图 6　测试流的带宽及进入时间

6.2　高优先权流的传输速率

4 条测试流中的流 1 和流 2 均为高优先权流，首先对流 1 的传输速率进行了分析，实验结果如图 7 所示。

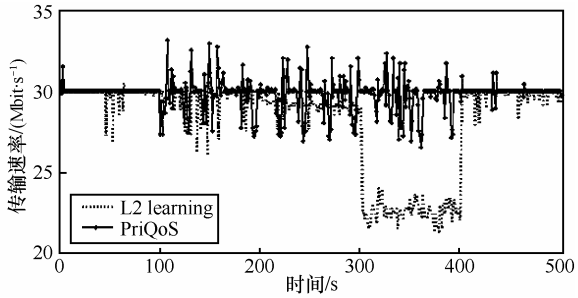

图 7　高优先权流 1 的传输带宽

0~100 s：网络中只有高优先权流 1，所需带宽为 30 Mbit/s，PriQoS 的高优先权队列和 L_2 learning 交换机的链路能力均能满足要求，所以能较好地保持传输速率。

100~200 s：由于加入了高优先权流 2，传输速率为 40 Mbit/s，此时高优先权流的带宽需求为 70 Mbit/s，而高优先权队列的实际带宽为 40 Mbit/s，所以需要对增加该队列带宽，扩容后流 1 仍然保持了 30 Mbit/s 的传输速率。在使用 L_2 learning 交换机的情况下，由于流 1 和流 2 共享 100 Mbit/s 带宽，所以也能够保证传输的吞吐量。

200~300 s：加入了中优先权流 3，速率为 30 Mbit/s，此时 3 条流的总带宽需求为 100 Mbit/s，到达链路的性能上限。此时 PriQoS 由于首先保证高优先权流的服务质量，流 1 的传输速率没有受到影响，仍然为 30 Mbit/s；而在 L_2 learning 交换机中，3 条流共享 100 Mbit/s 链路带宽，所以流 1 的传输速率有所下降（链路利用率不可能达到 100%），大致为 28 Mbit/s。

300~400 s：加入了低优先权流4，速率为30 Mbit/s，此时 3 条流的总带宽需求为 130 Mbit/s，超出链路的性能上限。此时在 PriQoS 中，高优先权流 1 的传输速率仍然没有受到影响，为 30 Mbit/s；而在 L_2 learning 交换机中，4 条流竞争使用瓶颈带宽，所以流 1 的传输速率下降较多，大致为 22 Mbit/s，比实际需求低 8 Mbit/s。

400~500 s：流 2 传输结束，此时总带宽需求为 90 Mbit/s，高优先权流需求为 30 Mbit/s，链路性能能够满足要求，2 种机制表现基本相同。

实验结果分析：1）在所有业务流总带宽需求大于物理带宽时，如果不加以控制（使用 L_2 learning 交换机制），高优先权的通信需求将无法得到满足；2）PriQoS 能够感知高优先权流的带宽需求，并进行高优先权队列带宽的动态增减适配，当发现高优先权队列由于带宽不足出现缓存溢出分组丢失时，迅速从低优先权队列中抢占一部分带宽，只要端口的物理带宽大于高优先权流的总需求，PriQoS 就能够满足高优先权业务流的通信需求。

6.3 高优先权流的分组丢弃率

当队列的带宽不足时，将会导致缓存溢出而丢弃部分分组，所以通过队列的分组丢弃率也能够体现该队列带宽的变化情况。图 8 对比了在 2 种模式下高优先权流 1 在传输过程中的因缓存溢出而导致的分组丢弃率，可以看出，在使用 PriQoS 机制时，分组丢弃率始终保持在较低水平。如在 200~400 s，即使所有流的带宽总需求超出物理带宽时，分组丢弃率也能稳定保持在小于 1%的较低水平，而此时如果使用 L_2 learning 机制，高优先权队列的分组丢弃率将达到 25%，严重影响了该流的服务质量。

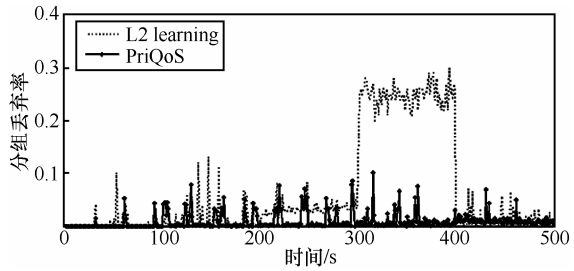

图 8 高优先权流 1 的分组丢弃率

实验结果分析：PriQoS 能够及时对高优先权流的带宽需求做出响应，尽可能使分组丢弃率保持在较低水平，从而有效保证其服务质量。

6.4 在 PriQoS 模式下 4 条流的传输吞吐量

从图 9 可以看出，高优先权的流 1 和流 2 在传输过程中能够始终保持其要求的传输速率，但中优先权流 3 和低优先权流 4 则由于会被高优先权队列抢占带宽，所以其传输速率会经常受到影响，波动较大，有时甚至被完全抢占导致传输速率为 0。而在 400~500 s，由于只有流 1、流 3、流 4 这 3 条流，速率均为 30 Mbit/s，总带宽需求为 90 Mbit/s，小于链路性能 100 Mbit/s，所以能够自适应地恢复各自的传输速率。

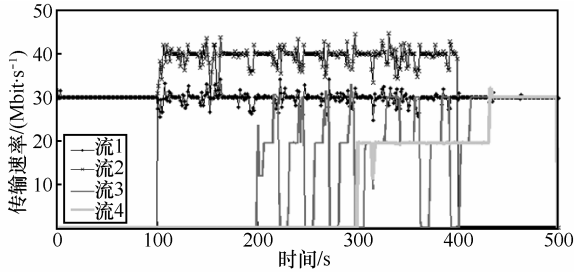

图 9 使用 PriQoS 时 4 条流的传输速率

图 10 为使用 L_2 learning 时，4 条流的传输情况。显然，在 300~400 s 由于 4 条流总带宽需求超出链路能力 30%，此时流 1、流 3、流 4 下降到 23 Mbit/s，而流 2 需求为 40 Mbit/s，但实际速率下降到 30 Mbit/s，4 条流均比各自的需求带宽减少 23%，即高优先权的流 1、流 2 的传输带宽无法得到保证。

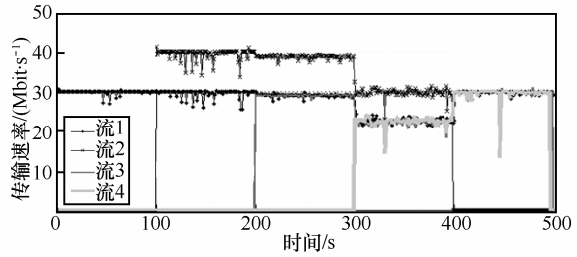

图 10 使用 L_2 learning 机制时 4 条流的传输速率

实验结果分析：当网络拥塞时，PriQoS 牺牲其他较低优先权流的传输性能，来尽可能保证高优先权的传输带宽和服务质量。

6.5 控制器开销

为了评估优先权计算过程对控制器带来的开销，还对控制器响应流请求的能力进行了测试。由于 PriQoS 控制器在生成流表时需要识别业务流的用户等级、应用重要性和流量类型等属性，并检索和归一化对应的量化数值，然后计算综合优先权，这些过程不可避免会对控制器带来一定的计算和

时延开销。通过单位时间内控制器响应的流表请求数量，能够反映出这种开销的大小。

实验中使用 iperf 分别由 H_1、H_2、H_3 向 H_4、H_5、H_6 产生多条不同目的端口的流，通过流的源 IP 地址作为识别用户级别的特征，通过运输层端口号作为识别应用重要性和流量类型的特征。先将业务流特征映射规则库读入内存，使用 oflops 下的 cbench[18]工具来测量控制器的实际性能参数，并通过 oftrace[19]工具进行分析，获得控制器对 packet-in 分组的响应性能，从而对 PriQoS 的检索和计算开销进行评估。Cbench 工具主要用来测量控制器的实际吞吐量，它通过仿真一定数量的 OpenFlow 交换机向控制器发送 packet-in 分组，并根据 packet-in 分组的响应分组来记录控制器的实际性能。

图 11 显示了在使用 PriQoS 和 L_2 learning 这 2 种机制时，在单位时间内（1 s）能够响应的流请求数量。可以看出，当控制器接收到的流请求速率小于 1 200 个/秒时（以下称为临界请求速率），PriQoS 和 L_2 learning 模式均能对所有的流请求做出响应。但当流请求速率超过该临界请求速率时，会有部分流请求无法得到响应。并且随着请求流速率的进一步增大，2 种模式下的控制器的响应能力均会降低，随后基本稳定在 900~1 000 个/秒。当流请求速率低于 5 000 个/秒时，PriQoS 和 L_2 learning 2 种机制性能基本相同；但超过该值时，PriQoS 会对控制器的流响应能力造成一定程度的影响，均值大约为 900 个/秒，而 L_2 learning 模式大约为 1 000 个/秒，影响幅度大约为 10%。

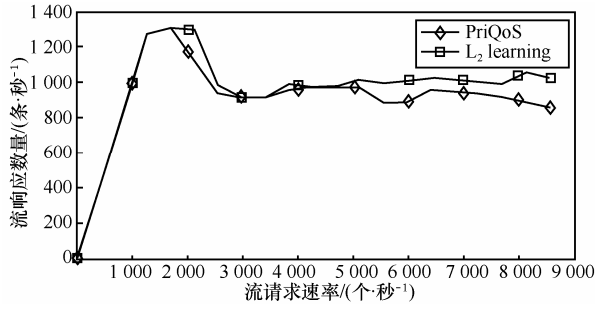

图 11　单位时间内（1 s）流建立请求的响应数量

实验结果分析：在 PriQoS 机制下，随着流请求数量的猛增，大量请求流的量化数据检索和加权求和过程对控制器的 CPU 和内存均会增加一定的开销，从而对控制器的响应能力造成轻微影响。但该问题在实际应用中并不会造成太大的影响，可以采用商用高性能服务器运行控制器程序，以提高控制器的响应能力。

6.6　端口队列状态轮询周期 T 的取值

PriQoS 通过轮询方式来感知交换机的端口队列状态，以便判断高优先权队列分配的带宽能否满足传输要求。轮询周期 T 越小，队列状态采集的准确度越高，控制器对队列带宽不足的感知越灵敏，但控制器的开销也会相应增高；反之亦然。为了验证该结论，并确定实验中轮询周期 T 的取值，通过实验分析了在取不同的轮询周期时，控制器对流请求的响应能力和高优先权队列的分组丢弃率，如图 12 所示。

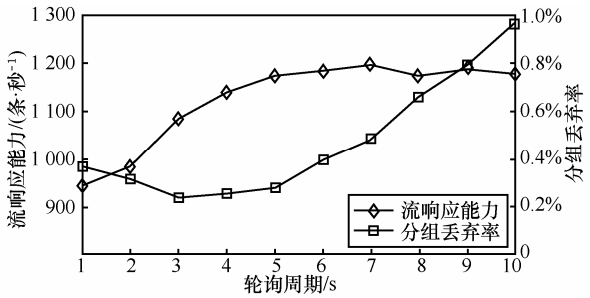

图 12　PriQoS 中轮询周期取值的影响

流请求的响应能力反映了不同的轮询周期对控制器开销的影响。实验中发送到控制器的流请求数量约为 2 000 条/秒，从图 12 中可以看出，该控制器对流请求的响应能力约为 950~1 200 条/秒。当轮询周期 T 较小时，由于控制器频繁根据收到的队列状态进行计算，开销较大，所以其流请求响应能力降低；随着 T 取值的增大，流响应能力逐渐上升，当 T 大于 4 s 时，这种影响基本可以忽略。

高优先权队列的分组丢弃率反映了不同的轮询周期对带宽不足状况的感知能力。如果能够及时感知到带宽不足导致分组丢弃的状况，并及时进行带宽重分配，可有效降低分组的丢弃率。在图 12 中，随着轮询周期 T 的增大，分组丢弃率逐渐增加，当 T 大于 5 s 时，分组丢弃率上升较快（但始终低于 1%）。

实验结果分析：在 PriQoS 机制下，轮询周期 T 的取值对控制器开销和高优先权队列的通信质量会产生一定影响。基于上述分析，实验中将 T 取为 5 s，此时具有较小的控制器开销，且高优先权队列的分组丢弃率也较低；将每次带宽重分配发生后的动态探测周期 T_1 设为 1 s，在以长流为主的实际网

络中，带宽重分配通常不会频繁发生，所以较小的动态探测周期 T_1 并不会对控制器造成过大开销，并有利于快速感知本轮带宽重分配能否满足传输需求。

7 结束语

SDN 的控制平面和数据平面分离的特性，为网络创新提供了极大的便利。利用这种优势来解决传统的 IP 网络存在的 QoS 问题是一个值得研究的课题。本文提出了一种在 IP 网络边缘部署 PriQoS 机制的方法，PriQoS 基于 SDN 技术，通过控制器在边界交换机端口上实现优先权队列，并动态轮询队列的状态，为高优先权队列自适应分配带宽，以经济高效的方式确保了高优先权业务流的通信质量。实验结果表明，PriQoS 能够有效保障高优先权业务流的服务质量，但当流请求数量较高时，业务流多维属性识别、优先权计算等过程会对控制器带来开销。在后续工作中，将在设计高效的综合优先权映射算法，减小控制器开销和该技术的实用化等方面进行研究。

参考文献：

[1] BELL G. Failure to thrive: QoS and the culture of operational networking[C]// ACM SIGCOMM Workshop on Revisiting Ip Qos: What Have We Learned, Why Do We Care. ACM, 2003: 115-120.

[2] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. OpenFlow: enabling innovation in campus networks[J]. ACM Sigcomm Computer Communication Review, 2008, 38(2): 69-74.

[3] CLARK D, BRADEN B, SHENKER S, et al. Integrated service in the Internet architecture: an overview[S]. IETF RFC, 1994.

[4] BLAKE S, BLACK D, CARLSON M, et al. An architecture for differentiated services, Internet RFC 2475[S]. RFC, 1998.

[5] CRAWLEY E. A Framework for QoS-based routing in the Internet, Internet RFC 2386[S]. RFC, 1998.

[6] BUENO I, AZNAR J I, ESCALONA E, et al. An OpenNaaS based SDN framework for dynamic QoS control[C]// 2013 IEEE SDN for Future Networks and Services (SDN4FNS). 2013: 1-7.

[7] EGILMEZ H E, CIVANLAR S. An optimization framework for QoS-enabled adaptive video streaming over openflow networks[J]. IEEE Trans on Multimedia, 2013, 15(3): 710-715.

[8] AKELLA A V, XIONG K. Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN)[C]// 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC). 2014: 7-13.

[9] TOMOVIC S, PRASAD N, RADUSINOVIC I. SDN control framework for QoS provisioning[C]//Telecommunications Forum Telfor. 2014.

[10] AKYILDIZ I F, LEE A, WANG P, et al. A roadmap for traffic engineering in SDN-Openflow networks[J]. Computer Networks the International Journal of Computer & Telecommunications Networking, 2014, 71(3):1-30.

[11] SAATY T L. How to make a decision: the analytic hierarchy process[J]. European Journal of Operational Research, 1990, 48(1):9-26.

[12] 徐泽水. 不确定多属性决策方法及应用[M]. 北京: 清华大学出版社, 2004.
XU Z S. Uncertain multiple attribute decision making: methods and applications[M]. Beijing: Tsinghua University Press, 2004.

[13] Linux Foundation Collaborative Project Open vSwitch Documentation[EB/OL]. http://vswitch.org/.

[14] VALENZUELA J L, MONLEON A, SAN E I, et al. A hierarchical token bucket algorithm to enhance QoS in IEEE 802.11: Proposal, Implementation and Evaluation[C]//Vehicular Technology Conference, 2004: 2659-2662.

[15] HANDIGOL N, HELLER B, JEYAKUMAR V, et al. Reproducible network experiments using container-based emulation[C]//Proc of the 2012 ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT). New York: ACM Press, 2012: 253-264.

[16] Open Networking Lab. POX Wiki [EB/OL]. https://openflow. stanford.edu/ display/ONL/POX+Wiki/, 2016.

[17] Open Network Foundation(ONF). OpenFlow switch specification v1.1.0［EB/OL］. https://www.opennetworking.org/images /stories/downloads/specification/openflow-spec-v1.0.0.pdf, 2016.

[18] SHERWOOD R, YAP K K. Cbench: an openFlow controller benchmarker[EB/OL]. http://www. openflow.org/wk/index.php/Oflops.

[19] SHERWOOD R, YAP K. An openflow dump analyzer tracing library[EB/OL]. http://www.openflow.org/wk/index. php/Liboftrace.

作者简介：

孙志（1973-），男，山东潍坊人，海军航空工程学院青岛校区工程师，解放军理工大学博士生，主要研究方向为软件定义网络、无线自组网络、网络安全、网络测量和性能评价。

陈鸣（1956-），男，江苏无锡人，博士，解放军理工大学教授、博士生导师，主要研究方向为未来网络、网络体系结构和网络测量等。