doi:10.11959/j.issn.1000-436x.2017190

消除 OpenFlow 网络中 UDP 冗余控制分组的机制

胡慧[1]，陈鸣[1,2]，刘波[1]，许博[1]，邢长友[1,3]，胡超[1,3]
（1. 陆军工程大学指挥信息系统学院，江苏 南京 210007；
2. 南京航空航天大学计算机科学与技术学院，江苏 南京 211106；
3. 东南大学计算机网络和信息集成教育部重点实验室，江苏 南京 211189）

摘 要：无连接的 UDP 数据流产生的大量冗余控制分组会对 SDN 控制器和网络产生严重的性能影响。首先，测试并建模分析了冗余控制分组对控制器性能的危害，进而形成了解决该问题的基本思路。由此提出一种基于预装流表并滤除冗余分组（PFFR, preinstalling flow-tables & filtering redundant-packets）机制。PFFR 通过预装转移流表项限制 UDP 控制分组的初始速率，并通过按路径安装流表和冗余分组滤除算法快速消除冗余控制分组。实现了基于 PFFR 的原型系统并进行了测试，结果表明，PFFR 方法能够有效提升控制器的性能。
关键词：OpenFlow；控制器；UDP 流；冗余分组；预安装流表
中图分类号：TP393 **文献标识码**：A

Mechanism of eliminating UDP redundancy control packets in OpenFlow network

HU Hui[1], CHEN Ming[1,2], LIU Bo[1], XU Bo[1], XING Chang-you[1,3], HU Chao[1,3]
(1. College of Command Information Systems, PLA Army Engineering University, Nanjing 210007, China;
2. College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China;
3. Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211189, China)

Abstract: Large numbers of redundant control packets produced by connectionless UDP flows may engender serious influence over the performance of the SDN controllers and networks. The endangerment of the redundant control packets for the performance of SDN controllers by testing and modeling was firstly analyzed, and then a basic solution to solve the problem was formed. Therefore, a preinstalling flow-tables & filtering redundant packets (PFFR) mechanism was proposed. By preinstalling flow tables, PFFR limited the initial rate of control packets in UDP flows, and through installing flow tables according to paths and utilizing the redundant packets filtering algorithm, PFFR eliminated redundant packets rapidly. A prototype system based on PFFR was implemented and tested. The experimental results prove that the PFFR mechanism can effectively improve the performance of the controller.
Key words: OpenFlow, controller, UDP flow, redundant packet, preinstall flow table

1 引言

软件定义网络[1]（SDN, software-defined network）是一种新型的网络体系结构，它允许网络管理员通过在控制器上编制软件灵活地管理网络和部署网络新功能。SDN 采用控制平面与数据平面相

收稿日期：2016-12-23；修回日期：2017-03-29
通信作者：陈鸣，mingchennj@163.com
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），江苏省科技计划基金资助项目（No.BY2013095-1-06）；江苏省自然科学基金资助项目（No.BK20140070）；复杂电子系统仿真重点实验室基础研究基金资助项目（No.DXZT-JC-ZZ-2015-018）
Foundation Items: The National Basic Research Program of China (973 Program) (No.2012CB315806), The National Natural Science Foundation of China (No.61379149), The Science & Technology Plan of Jiangsu Province (No.BY2013095-1-06), The Natural Science Foundation of Jiangsu Province (No.BK20140070), The Basic Research Foundation of Science and Technology on Complex Electronic System Simulation Laboratory (No.DXZT-JC-ZZ-2015-018)

分离的架构，打破了 IP 层僵化的局面，开辟了网络创新的新途径。OpenFlow 作为 SDN 的一种南向接口标准[2]，已经得到了学术界和工业界的高度认可。SDN 可采用主动和被动这 2 种工作模式，其中，主动模式是指控制器预先安装流所需要的转发流表，而被动模式则是指控制器根据 OpenFlow 交换机发送的流请求分组为该流计算和安装传输路径流表。由于主动模式无法完全预测网络中所有流的需求，而被动模式则能够根据应用需求更为精细地管控流量，因此 SDN 被动工作模式得到了普遍应用。

但在另一方面，被动工作模式也会对控制器的性能产生不利影响。具体来说，当某条流的分组到达 OpenFlow 交换机，且该 OpenFlow 交换机没有与该流相匹配的流表项时，交换机会将该分组封装成 Packet_In 分组转发至 SDN 控制器。对于面向连接的数据流，如 TCP 流，它们建立连接的机制如三次握手机制会抑制后继分组的到达，使控制器不会收到同一条流中的其他冗余 Packet_In 分组。但对于无连接的数据流如 UDP 流而言，在控制器为该流完成传输路径的流表安装之前，该流可能会向网络中发送大量的分组，这些分组都将被 OpenFlow 交换机以 Packet_In 分组的形式转发给控制器，因此控制器会收到大量无连接流的 Packet_In 分组。由于对于每一条流，控制器仅需要一个 Packet_In 分组就可以计算路径并安装流表，因此大量属于同一条流的控制分组对于控制器而言都是冗余的，当控制器为这些大量的冗余控制分组执行相同的复杂处理时，就会严重浪费控制器的资源，降低控制器的性能，甚至导致控制器和整个 SDN 的崩溃。UDP 协议是一种重要的传输协议，在互联网中存在着大量的基于 UDP 协议的网络应用，如 P2P 下载应用、视频应用等，这些应用都导致了大量的 UDP 流，据统计，目前，国内 UDP 流量占到网络总流量的 10%~35%[3]，随着 SDN 在互联网中广泛部署，这些 UDP 流量必将会对 OpenFlow 网络性能产生严重影响。因此，降低冗余控制分组对 SDN 控制器性能的不利影响是一个亟需解决的问题。然而，不仅当前的 OpenFlow 协议（v1.5.1）[4]缺少相关处理机制，而且目前的研究仍未见对此问题进行深入分析及定量评价。

针对无连接数据流中的冗余控制分组降低控制器性能的问题，本文提出了一种 PFFR 的机制。实现了 PFFR 原型系统并进行了实验测试。

2 相关工作

在 OpenFlow 网络中，控制器执行集中式控制决策，复杂性大部分集中于控制器中[5]。在被动工作模式下，控制器要为每一条新流都安装路径流表，当流数很大时控制器将可能无法及时处理新流的路由请求而成为性能瓶颈。文献[6]指出，现有的 SDN 控制器均不能对 10 Gbit/s 高速链路网络中的大量新流进行及时处理。文献[7]指出，缺乏可扩展性将会引发耗尽控制器资源的各种攻击行为，SDN 可能会遭受比传统网络更为严重的危害。文献[8]分析了无连接数据流会向控制器发送大量的控制分组等待控制器处理，耗费控制器资源。

为了减少无连接数据流对控制器性能的影响，文献[9]分别提出了在控制平面和在数据平面消除冗余控制分组的方法。在控制平面通过维护最新 Packet_In 分组视图，使冗余控制分组在控制器端直接被丢弃，从而消除冗余控制分组。该方法当流量以及网络规模较大时，不能避免大量冗余控制分组占用控制器资源，因此不能明显改善控制器和交换机的性能。在数据平面，通过对 OpenFlow 交换机进行修改，将同一条流的冗余控制分组暂时缓存至 SDN 交换机，使每一条新流仅向控制器发送一个 Packet_In 分组，但该方法要求修改 OpenFlow 交换机，要在交换机上增加复杂的流识别功能，将部分控制功能下放至交换机中，同时需要交换机缓存大量的冗余分组，通过牺牲交换机资源缓解控制器压力，该方法的可部署性与可扩展性不强。

除了无连接数据流冗余分组外，人为地产生大量 Packet_In 控制分组的恶意攻击行为可能会对控制器性能产生严重影响[10]。文献[11]指出数据平面流数目的急剧增加将会导致交换机向控制器发送大量的流请求分组，并最终可能导致控制器崩溃。文献[12]为应对 SDN 控制器的单点失效问题，提出了使用备用控制器以提高控制器的恢复能力，但备用控制器仍会遭受大量 Packet_In 控制分组攻击，因此，该方法无法根本解决大量冗余 Packet_In 控制分组问题。文献[13]描述了多控制器级联可能存在的故障，说明了使用多个控制器无法完全解决单控制器失效的问题。因此，为保障 OpenFlow 网络的可靠运行，亟需研究解决因控制器应对大量控制分组而引发的系统性能问题。本文主要针对无连接 UDP 流产生大量冗余控制分组对控制器性能影响的问题进行研究。

3　无连接数据流冗余分组的危害

本节首先通过实验分析 UDP 冗余分组潜在的危害，然后建立控制器处理冗余分组的排队模型，并分析了解决该问题的思路。

3.1　无连接数据流冗余分组分析

在 OpenFlow 网络中，通常采用被动模式细粒度管理网络流量。当一条流进入 OpenFlow 交换机后，交换机会根据流首个分组的信息匹配已有的流表，并选择优先级最高的流表进行匹配，根据流表中的 Action 字段对该流进行相应的操作。如果没有匹配成功，交换机就会将流的首个分组以 Packet_In 形式转发给控制器。控制器处理后若同意转发，就会在相关交换机中插入流表，使该流的后继分组沿着交换机中的该流表进行转发。控制器对到达的 Packet_In 分组进行处理时，路由计算、安全性分析、安装流表等操作会消耗控制器资源，因此收到的控制分组越多，控制器资源消耗就越大。

实际上，对于每一条 UDP 流，控制器只需要处理首个 Packet_In 分组就能建立传输路径，然而它却可能要处理大量无用的 Packet_In 分组。而且，UDP 流速率越高，冗余分组越多，控制器浪费的资源就越多，导致其他流接受控制器服务的概率下降。严重时，可能导致其他到达的正常流因等待时间过长而超时，使用户服务的体验严重变差。例如，如果在某 TCP 流等待建立连接的时间内，控制器无法及时响应该 TCP 流，就会导致该 TCP 请求失败。当冗余 Packet_In 分组很多时，还会占用控制器和交换机的通信信道，影响控制器正常获取交换机的状态信息，对 OpenFlow 网络运行产生不良后果。

为了定量评价 UDP 流产生的冗余分组对控制器性能的影响，本文在 Mininet [14]中构建如图 1 所示的控制器性能测量环境，其中 OpenFlow 交换机采用 Open vSwitch [15]，控制器采用的是 RYU[16]。

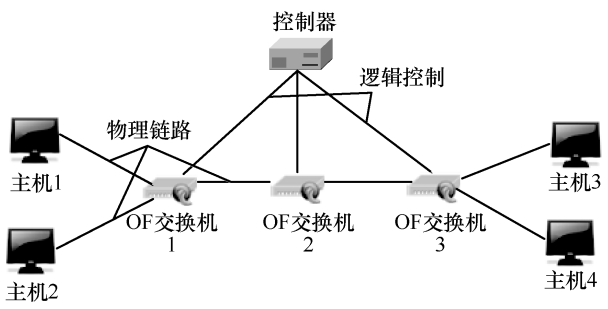


图 1　控制器性能测量环境

实验条件如下。

1) 在 4 台主机中的 H_1~H_3 和 H_2~H_4 之间分别发送 UDP 流，流数分别为 12、24、36 条；当不断地增加每条流的速率时，测量相应的系统性能指标。

2) 为考察这些 UDP 流对 TCP 流的影响，同时在 H_1~H_3 和 H_2~H_4 之间分别发送 12 条 TCP 流作为评价性能的参考。

3) 控制器根据流的源和目的 IP 地址和端口号四元组，计算到达分组的数量。

4) 测量的系统性能指标包括：到达控制器的 UDP 控制分组数量、控制器 CPU 利用率和 TCP 样本流连接建立时延。

5) 每组实验共进行 10 次，取其平均值作为最后结果。相关性能测量的结果如图 2 所示。

通过实验，可知：1) 随着 UDP 流发送速率的增长，到达控制器的 UDP 分组数量几乎呈线性增长（如图 2(a)所示），当 12 条 UDP 流以 1 Mbit/s 速率发送时，控制分组数为 40，有 28 个冗余分组；而当 12 条 UDP 流以 100 Mbit/s 速率发送时，冗余分组则达 611 个；2) 随着 UDP 流发送速率的增长，控制器 CPU 利用率也快速增加（如图 2(b)所示），测量表明，控制器处理一个分组的时延约为 0.8 ms，那么第 611 个控制分组产生了约 488.8 ms 的排队时延，同时也引起了控制器的 CPU 利用率快速增长；3) 大量冗余分组占用了控制器服务等待队列，导致后继 TCP 流的首个分组无法及时进入队列得到服务，使 TCP 建立连接的时延急剧增加（如图 2(c)所示），甚至会因队列满而被丢弃，从而导致 TCP 流连接请求因超时而重发甚至建立连接失败。如图 2(c)所示，当 36 条 UDP 流的发送带宽大于 50 Mbit/s 时，TCP 连接建立失败。

综上，无连接数据流的大量冗余控制分组对控制器乃至整个 OpenFlow 网络性能产生的负面影响很大。

3.2　控制器处理分组的排队模型

本节采用排队论对 3.1 节的问题进行进一步分析。在如图 1 所示的简单拓扑环境下，若 K 台 OpenFlow 交换机都与控制器直接相连，对于连接交换机的每个控制器端口，采用 M/G/1-FCFS（first come first serve）排队模型来描述到达分组的队列排队情况。控制器对于所有交换机的连接端口采用循环（round robin）调度算法。控制器对每个端口的平均处理速率为

$$c=\frac{C}{K}$$

其中，c 表示控制器对单个队列中分组的平均处理速率，C 表示控制器主机的处理速率。

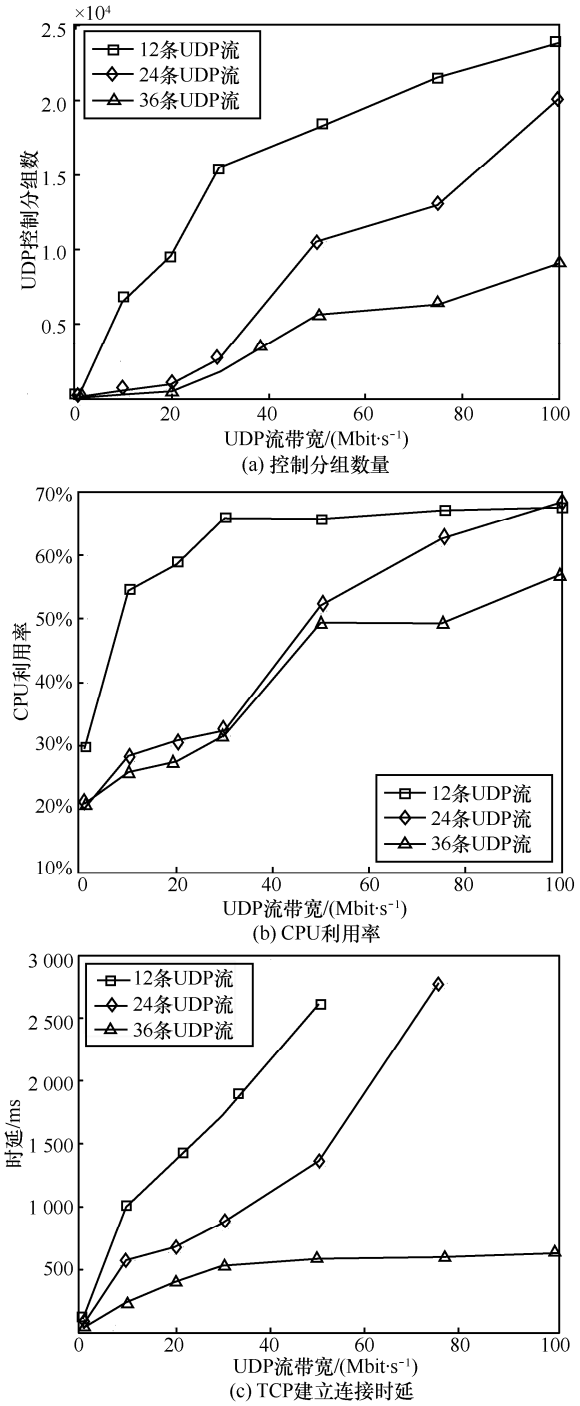


图 2 UDP 流对系统性能的影响

对于一个控制器端口，在负载为 ρ 时，根据 Pollacrek-Khintchin 公式[17]，控制器端口中平均排队的分组数目为

$$\mathrm{E}[N]=\frac{\lambda\rho}{2(1-\rho)}\frac{\mathrm{E}[x^2]}{\mathrm{E}[x]}$$

其中，x 表示该条流突发的大小，$\mathrm{E}[x]$ 表示突发分组数量 x 的期望值，$\mathrm{E}[x^2]$ 表示突发分组数量 x 的二阶矩，ρ 为交换机和控制器连接链路的负载，即到达流量和链路容量的比值。可以看出，控制器端口队列长度只与当前的链路负载和流的突发速率大小有关。

由于控制器要处理多个交换机发送的 Packet_In 分组，假设 Packet_In 的到达速率为 λ，可得

$$\rho=\frac{\lambda}{c}=\frac{\lambda K}{C}$$

对于发送速率为 V 的数据流，其传输速率与其突发速率有式(1)的关系

$$\begin{aligned}\mathrm{E}[V]&=\lambda\mathrm{E}[x]\\ \mathrm{E}[V^2]&=\lambda^2\,\mathrm{E}[x^2]+\rho\end{aligned}\tag{1}$$

由于 UDP 流是无连接的，将持续向控制器发送分组，那么存在 $\lambda=V$。因此，该数据流进入网络后，控制器端口的平均队列长度为

$$\mathrm{E}[N]=\frac{KV(CV-K)}{2C(C-KV)}\approx\frac{KV^2}{2(C-KV)}\tag{2}$$

对于一个最新转发给控制器的分组，得到控制器处理的平均等待时间为

$$\mathrm{E}[w]=\frac{\mathrm{E}[N]}{\lambda}\approx\frac{KV}{2(C-KV)}\tag{3}$$

当 $\mathrm{E}[N]$ 大于控制器端口缓存时，后续到达该控制器的分组将会被丢弃，导致新流无法得到控制器的响应，如果到达的分组是交换机请求控制器下发流表项或控制器响应交换机的分组，将会导致 2 个平面的通信低效或故障，甚至导致整个 OpenFlow 网络失常；同理，对于面向连接的 TCP 流，当 $\mathrm{E}[w]$ 大于 TCP 流建立连接的时间，TCP 流建立连接就会因超时而失败。这些问题可能导致 OpenFlow 网络的性能下降和网络服务的失效。

同时，通过分析式(2)和式(3)可得以下结论。

1) 从式(2)可得，$\mathrm{E}[N]$ 对于 V 呈单调递增关系，那么无连接数据流导致的冗余分组排队长度与流的发送速率 V 成正比。因此，为了减小无连接流产生的冗余控制分组排队数目，在交换机建立起数据流路径之前，应尽可能降低无连接数据流的控制分

组到达控制器的速率。对于 E[w]，从式(3)也可以得到类似的结论。

2) 从式(2)可得，E[N]对于 K 呈单调递增关系，那么无连接数据流导致的冗余分组排队长度与传输路径上交换机数目成正比。因此，为了减小无连接流产生的冗余控制分组排队数目，需要采用按路径安装流表的方式，即控制器收到流的控制分组后，为该流安装端到端的路径，从而使该流仅通过一台交换机将控制分组转发给控制器。对于分组的排队时延进行分析，也可以得到类似的结论。

根据以上 2 个结论，可以得到解决无连接流冗余分组的基本思路：1) 在控制器为无连接流建立路径完成之前，降低该流向控制器发送分组的速率；2) 对于无连接流的流表采用端到端的安装方式，而不是选择传统的逐跳安装的方式。

根据 3.1 节的实验结果可知，即使降低无连接流转发给控制器的分组数目，也不能保证控制器只接收到一个控制分组。因此，为了降低控制器处理冗余控制分组的资源消耗，进一步在控制器中部署冗余分组的快速检测方案，以滤除多余的控制分组，降低控制器的资源消耗。

根据上面的分析，本文在第 4 节中提出一种新型 PFFR 方法。

4　PFFR 方法设计

基于上述分析，本文提出一种 PFFR 方法。图 3 给出该方法工作的基本过程。PFFR 方法包括以下 3 个主要步骤。

步骤 1　安装转移流表，即预先在 OpenFlow 交换机上安装默认流表，并限制 UDP 流的最大转发速率。

步骤 2　按路径为交换机安装流表，即控制器计算出数据流的传输路径，为沿途交换机安装流表。

步骤 3　控制器尽快滤除队列冗余分组，即控制器使用某种算法快速定位冗余分组并丢弃之。其中步骤 2 由控制器路由模块实现，步骤 3 则由控制器滤除模块实现。

4.1　转移流表

为了降低 UDP 流在路径流表安装完成之前产生大量的控制分组对控制器性能的影响，必须限制 UDP 流控制分组向控制器的转发速率。由于 OpenFlow 交换机与控制器相连接的端口为逻辑端口，因而不能在该端口对 UDP 控制分组进行限速。本文借助于 OpenFlow 网络中包含多个交换机，预先在各交换机中安装了转发流表，这些流表将 UDP 流的分组转发到相邻交换机，并在第一跳交换机的出端口设置限速队列限速 UDP 流，从而减小流向控制器的冗余控制分组。由于限速队列会导致该队列大量的分组丢失，如果将所有 UDP 流都对应到一个限速队列，这必将导致后续其他 UDP 流无法向控制器发送控制分组，因此要使用多个队列来转发 UDP 流。

具体来说，对于客户到服务器的 UDP 流，控制器预先在 OpenFlow 交换机 OVS_1 的出端口（OVS_1

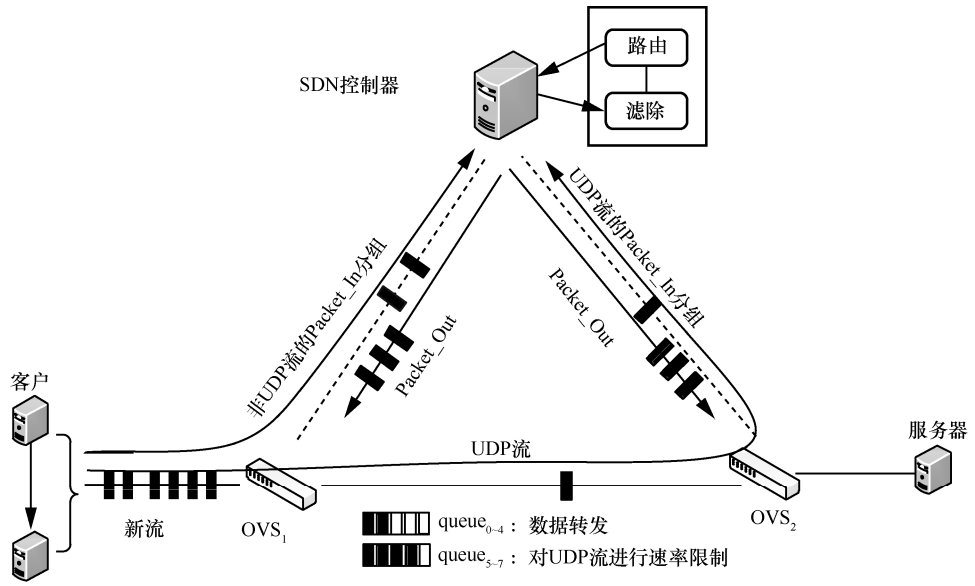


图 3　PFFR 方法的工作过程

和 OVS_2 连接的端口）上设定多个队列，然后通过预装优先级较低的流表将所有新到达的 UDP 流经过限速队列转移至相邻交换机 OVS_2 上，并由 OVS_2 将限速后的 UDP 流转发至控制器。由于 OpenFlow 交换机端口可支持 8 个队列（$queue_0$~$queue_7$），将 OpenFlow 交换机中 8 个队列均设置为相同的优先级，队列间采用循环调度的方式。为了避免新到达的 UDP 流请求分组拥挤在一个队列中，使用其中 3 个队列（如 $queue_5$、$queue_6$、$queue_7$）用于转移新到达的 UDP 流，而其他队列用于传输非 UDP 流和已安装流表的 UDP 流，实现不同种类流（无路径流表 UDP 流，已安装流表的 UDP 流和非 UDP 流）的传输隔离。同时借助于 OpenFlow 技术可以将不同种类流放置到对应的队列中。

下一个问题是限速队列的带宽应当设置为多大为宜？显然，式(2)为限速队列的定量计算提供了依据。在式(2)中，首先需要获取控制器的分组处理速率。为此，首先通过测量获取在一台典型机器上运行的控制器能够处理分组的速率。图 4 给出了在一台主频为 3.2 GHz、CPU 为 i5-3470、内存 4 GB、以太网卡为 1 Gbit/s 的主机上运行 RYU 控制器（版本为 3.16），它处理分组并下发流表的速率。

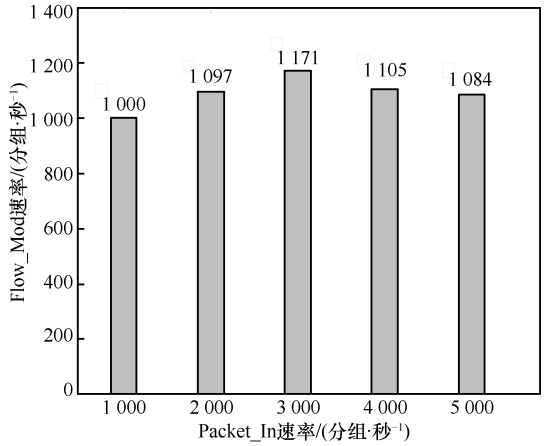


图 4 Flow_Mod 速率

图 4 中最大 Flow_Mod 速率为 1 171 分组/秒。当取分组长度为 1 500 B，交换机的平均排队长度为 1 个分组时，交换机向控制器转发的 UDP 流速率应小于 14.052 Mbit/s，约为 14 Mbit/s。由 3.2 节分析可知，冗余分组数与控制器相连的交换机数目 K 有关，因此，每台交换机中 $queue_{5\sim7}$ 的传输速率 V 最大值为 14 Mbit·s^{-1}/K。进行转移 UDP 流的流程如算法 1 所示。其中，第 4)行~第 6)行表示当控制器获取到 ICMP 分组时，安装转移流表，而对于其他流的控制分组，控制器根据控制分组信息为流安装流表。

算法 1 转移 UDP 流

输入 网络拓扑 tp, 流 F 的 Packet_In 分组 p, 所限速率 V

输出 Packet_In 分组 p 的流表项

1) for 每一条新到的分组 p do；
2) if p 是一条 Packet_In 分组 then
3) 通过分组 p 提取流 F 的特征信息；
4) if p 是一条 ICMP 分组 then
5) 为分组 p 被动安装流表项；
6) 为 UDP 流主动安装转移至相邻交换机的流表项；
7) else
8) 为分组 p 被动安装流表项；
9) end if
10) end if
11) end for

算法 1 的时间复杂性为 $O(|F|+N)$，其中 $|F|$ 为网络中流的数目，N 为连接主机的交换机数目。算法 1 较控制器正常的处理逻辑，仅增加了 $O(N)$ 的时间复杂性，同时 N 较于 $|F|$ 是一个很小的值，因此算法 1 并不会严重增加控制器的开销。采用转移流表方法，可以有效降低 UDP 流到达控制器的 Packet_In 分组速率，并且要求 K 台交换机都将 UDP 流速率限制在 14 Mbit·s^{-1}/K 之下。

4.2 按路径安装流表

PFFR 要求控制器根据网络的拓扑和流量等信息，一次计算出从数据流源主机到目的主机之间的端到端路由，从而同时向沿途的交换机安装流表。如果控制器为每一条 UDP 流采用逐跳的方式安装流表，那么每一跳交换机都将会向控制器发送大量的控制分组，因而会大大增加控制器的处理负担和网络流量。

例如，在图 3 中控制器收到客户发往服务器的流，根据网络视图可以计算得到的传输路径为 OVS_1→OVS_2，那么控制器依次给 OVS_1、OVS_2 安装流表。这样，客户到服务器的流将不会通过 OVS_2 再将控制分组转发到控制器。

4.3 滤除冗余 Packet_In 分组

一旦控制器为 UDP 流安装了端到端流表，UDP 流的后继分组就会沿路径传输并且不会转向控制

器。不过，在控制器队列中可能还有大量冗余分组等待控制器来处理，这些分组只会增加控制器负担，而没有任何作用，需要尽快采用算法 2 来滤除。

为此，PFFR 在控制器中维护一张最新的 Packet_In 分组视图 Packet_View，所有发送至控制器的分组信息的散列值存储于该视图中。每当 Packet_In 分组到达控制器后，通过查询该视图是否包括该散列值以检测该控制分组是否在近期已被操作。若 Packet_View 中没有该散列值，说明这是该流的首个分组，则将该值保存在视图中，并立即计算路径并安装流表；若 Packet_View 中已有该散列值，则检测到冗余分组，直接将其丢弃。注意到该流的信息可能会重复使用，PFFR 在 Packet_View 中保存每个分组的散列值时，也要保存当前的时刻，并为每个散列值设置一个 2 s 的生存时间，用软状态方式来避免影响新流。

算法 2　滤除冗余 Packet_In

输入　流 F 下的 Packet_In 分组 p，时间 t

输出　*HashTable*, 对分组 p 的处理

1) *HashTable*←Ø
2) for 每一条新到的分组 p do；
3)　if p 是一条 Packet_In 分组 then
4)　　通过分组 p 提取 F 的特征信息；
5)　　h←*hash*(F)；
6)　　if h∈*HashTable* then
7)　　丢弃分组 p；
8)　　continue；
9)　　else
10)　　t←*time*();
11)　　*HashTable*←*HashTable*∪<h,t>；
12)　　根据分组 p 计算路由；
13)　　end if
14)　end if
15) end for

算法 2 的时间复杂性为 $O(|F|)$。因此，算法 2 并没有增加控制器处理分组的时间复杂性。本文所提的 PFFR 方法总的时间复杂性为 $O(|F|+N)$，对控制器负载的影响极小，可忽略不计。

5　性能评价

本节实现了 PFFR 原型系统，并在实验环境下测试了系统的性能。

5.1　实验设计

实验选用 Mininet 作为实验平台。Mininet 仿真平台已被验证具有和真实网络环境相同的实验仿真真实度，其中运行真实网络流量，而网络中的节点皆通过网络功能虚拟化技术实现，尤其是在实验环境中，使用 Open vSwitch 软件交换机作为 OpenFlow 交换机。Mininet 具有很高的精确度，可以高质量再现文献[18]等的实验结果，已被广泛应用于 SDN 方案性能测试平台。实验拓扑如图 1 所示，其中的 3 台 OpenFlow 交换机是用 Open vSwitch v2.4.0 充当，预设置 8 个队列中的 3 个队列用于初始 UDP 流，PFFR 方法转移流表初始速率为 4.67 Mbit/s；交换机端口缓存为 150 个分组，端口采用 FCFS 调度模型和弃尾排队模型。控制器采用 RYU 3.16 版本。交换机和控制器运行于配置为主频 3.2 GHz、CPU i5-3470、内存 4 GB 和 1 Gbit/s 以太网卡的主机上，网络中所有链路带宽为 1 000 Mbit/s。实验中使用 Iperf 作为流量发生器，共发送 36 条 UDP 流和 12 条 TCP 流，其中，UDP 流的带宽从 1 Mbit/s 逐渐增加到 100 Mbit/s。每次实验的运行时间为 60 s。实验共重复 10 次，取其平均值作为实验结果。

5.2　实验结果及分析

图 5 给出了具有 PFFR 和没有 PFFR 方法的原型系统的性能评估曲线。性能评价指标包括：1) 控制器收到的 UDP 控制分组数目；2) 用于比较的 TCP 流建立连接时延；3) 控制器 CPU 的利用率；4) Flow_Mod 的速率。

在图 5(a)中，当控制器未采用 PFFR 方法时，UDP 控制分组的数量随着 UDP 流带宽的增加而呈显著上升，可多达 23 801 个，此时存在着 23 765 个冗余分组。相比之下，采用 PFFR 方法能够将分组数量维持在[36,73]的范围，最多仅产生 37 个冗余分组，这极大地减少了到达控制器的冗余分组。冗余分组的减少既能降低控制器的负担，也降低了分组从交换机转发至控制器所占用的带宽。

图 5(b)给出了 2 种情况下 TCP 连接建立时延。当 TCP 与 UDP 一起竞争控制器排队资源与处理能力时，一个 TCP 流仅会向控制器发送一个请求分组，该请求分组会与 UDP 流的大量冗余分组一道排队等候。TCP 分组得到服务的概率很低，有时甚至会因进入不了队列而造成 TCP 流连接超时。实验结果表明，PFFR 方法能够将 TCP 连接时延降低至 40 ms 内，而不采用 PFFR 方法则可能使 TCP 连接时延达

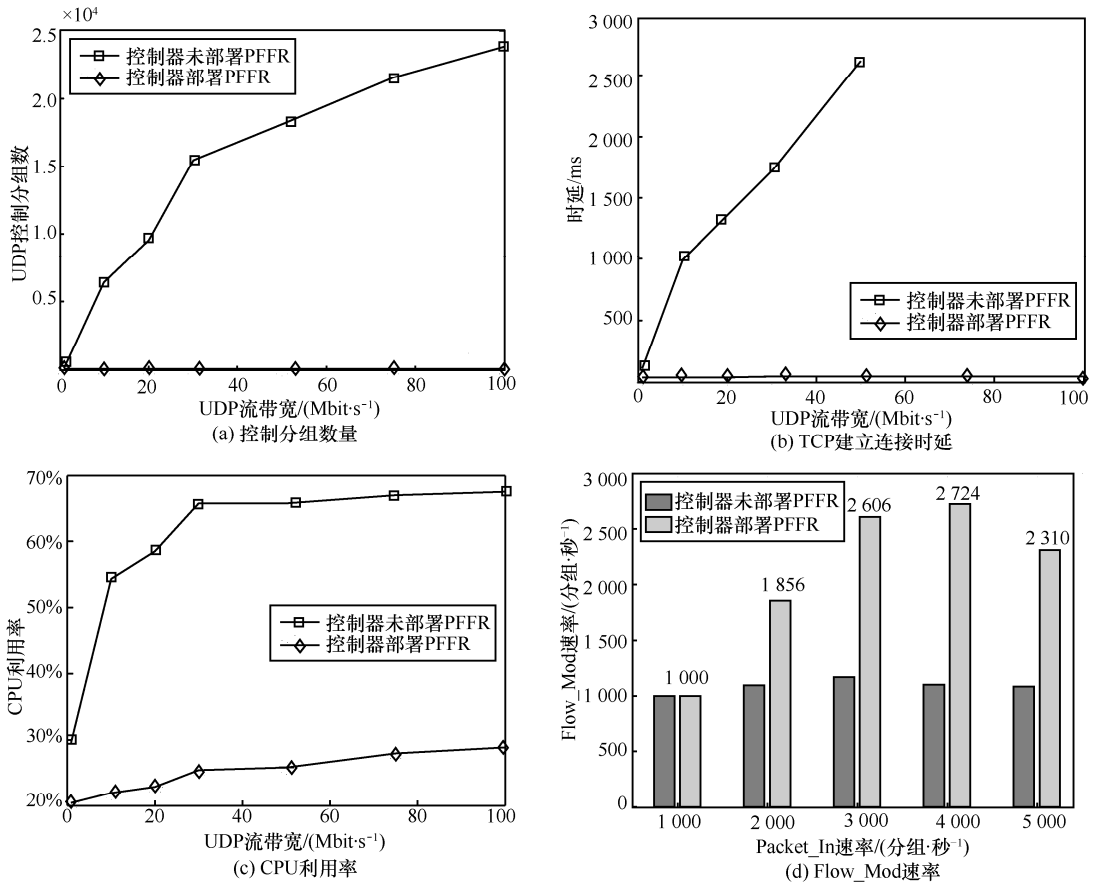


图 5　原型系统测试的性能

到 2 700 ms，甚至导致连接建立失败。

图 5(c)给出了 2 种情况下控制器的 CPU 利用率。可见，PFFR 方法使控制器 CPU 利用率维持在较低水平，始终低于 30%。而在不采用 PFFR 方法时，CPU 利用率随分组的到达速率而增加，通常高于 50%并趋向 70%，有时会使控制器运行不稳定。

图 5(d)给出了 2 种情况下控制器产生的 Flow_Mod 速率。可见，PFFR 方法使控制器产生 Flow_Mod 速率高达 2 724 分组/秒的速率，较不采用 PFFR 方法提升了 133%。采用 PFFR 方法时的测试表明，当 Packet_In 速率低于 1 000 分组/秒时，控制器能够及时处理到达分组；当 Packet_In 速率增加到 2 000 分组/秒时，控制器处理分组的速率提升了 69.1%；而在 Packet_In 速率达到 4 000 分组/秒时，Flow_Mod 速率达到 2 724 分组/秒，提升了 122.5%。这归功于 PFFR 方法滤除冗余分组的功能。测量结果表明，为新流计算路径并下发流表的时延为 1.516 6 ms，而分组散列处理的时延仅为 0.449 ms，两者相差 70.4%。

综上，本文提出的 PFFR 的确能够抑制 UDP 冗余分组的数目，提升控制器的性能，该机制能够提高 OpenFlow 网络的健壮性和可扩展性。

6　结束语

在利用 SDN 作为一种新型网络体系结构带来的便利时，需要重视可能存在的一些问题。无连接数据流对于控制器性能甚至网络的稳定性和扩展性的确存在着威胁，需要认真研究处理。本文通过实际测试并建模分析了 UDP 冗余控制分组对 SDN 控制器性能产生的影响，然后提出一种预装流表项并滤除冗余分组方法。原型系统实验表明了 PFFR 机制的有效性。下一步工作中，将在实际 OpenFlow 网络环境下进行 PFFR 机制的性能验证和实际部署。

参考文献：

[1] KREUTZ D, RAMOS F M V, VERISSIMO P E, et al. Software-defined networking: a comprehensive survey[J]. Proceedings of the IEEE, 2014, 103(1): 10-13.

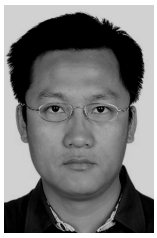
[2] LARA A, KOLASANI A, RAMAMURTHY B. Network innovation using OpenFlow: a survey[J]. IEEE Communications Surveys & Tutorials, 2014, 16(1): 493-512.
[3] LEE D, CARPENTER B E, BROWNLEE N. Observations of UDP to TCP ratio and port numbers[C]//Proc Int Conf on Internet Monitoring and Protection (ICIMP). 2010.
[4] SPECIFICATION O F S. Version 1.5. 1[J]. Open Networking Foundation, 2015.
[5] YAN Q, YU F R, GONG Q, et al. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges[J]. IEEE Communications Surveys & Tutorials, 2016, 18(1): 602-622.
[6] TOOTOONCHIAN A, GORBUNOV S, GANJALI Y, et al. On Controller Performance in Software-Defined Networks[J]. Hot-ICE, 2012, 12: 1-6.
[7] JARSCHEL M, OECHSNER S, SCHLOSSER D, et al. Modeling and performance evaluation of an OpenFlow architecture[C]//23rd International Tourism Consulting Partnership. 2011: 1-7.
[8] SHIN S, YEGNESWARAN V, PORRAS P, et al. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks[C]//Custom Computer Services. 2013: 413-424.
[9] 左青云, 陈鸣, 丁科, 等. OpenFlow 网络冗余控制分组消除机制研究[J]. 计算机研究与发展, 2014, 51(11): 2448-2457.
ZUO Q Y, CHEN M, DING K, et al. Eliminating redundant control messages in OpenFlow network[J]. Journal of Computer Research and Development, 2014, 51(11): 2448-2457.
[10] SHIN S, GU G. Attacking software-defined networks: a first feasibility study[C]//The 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. 2013: 165-166.
[11] GUDE N, KOPONEN T, PETTIT J. NOX: towards an operating system for networks[C]//ACM SIGCOMM, 2008:105-110.
[12] FONSECA P, BENNESBY R, MOTA E, et al. A replication component for resilient OpenFlow-based networking[C]//IEEE Network Operations and Management Symposium. 2012.
[13] PORRAS P, SHIM S, YEGNESWARAN V, et al. A security enforcement kernel for OpenFlow networks[C]//1st Workshop Hot Topics in Software. Defined Networks. 2012: 121-126.
[14] HANDIGOL N, HELLER B, JEYAKUMAR V, et al. Reproducible Network Experiments Using Container-Based Emulation[C]//Conference on emerging Networking Experiments and Technologies. 2012.
[15] PFAFF B, PETTIT J, KOPONEN T. The design and implementation of Open vSwitch[C]//USENIX Networked System Design and Implementation, 2015.
[16] RYU P T. RYU SDN framework using openflow 1.3. Website[J]. 2014.
[17] GROSS D, SHORTLE J F, THOMPSON J M, et al. Fundamentals of Queueing Theory[C]//Wiley-Interscience, 2008.
[18] ALIZADEH M, GREENBERG A, MALTZ D A, et al. Data center TCP (DCTCP)[J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 63-74.

作者简介：

胡慧（1993-），女，山西吕梁人，陆军工程大学硕士生，主要研究方向为软件定义网络、网络安全。

陈鸣（1956-），男，江苏无锡人，南京航空航天大学、陆军工程大学教授、博士生导师，主要研究方向为网络体系结构、网络测量、网络管理等。

刘波（1988-），男，陕西渭南人，陆军工程大学博士生，主要研究方向为未来网络、数据中心网络、流量工程。



许博（1980-），男，甘肃兰州人，博士，陆军工程大学副教授，主要研究方向为分布式计算、网络测量等。

邢长友（1982-），男，河南杞县人，博士，陆军工程大学副教授，主要研究方向为未来网络、P2P 多媒体通信等。

胡超（1984-），男，江西吉安人，博士，陆军工程大学讲师，主要研究方向为分布式计算、网络体系结构等。