OpenFlow 交换机流表溢出问题的缓解机制

乔思祎[1),2)] 胡成臣[1)] 李 昊[1)] 管晓宏[1)] 邹建华[1)]

[1)](西安交通大学智能网络与网络安全教育部重点实验室 西安 710049)

[2)](通信网络信息传输与分发技术重点实验室 石家庄 050081)

摘 要 在新兴的软件定义网络(Software Defined Networking,SDN)、OpenFlow 交换机中,为满足 OpenFlow 协议宽匹配域的需求,SDN 交换设备需要更大的查找表存储容量.当流表溢出时,将导致控制报文数目爆炸性增长、数据包传输时延增大等危害网络正常运行的后果.然而考虑成本因素,高速查找表容量不可能无限增加.即使单纯地增加流表容量,并不能使溢出的概率降低为零,且极不经济.本文分析了网络流量的特征,提出了一种流表共享方法(Flow Table Sharing,FTS),针对流表溢出现象带来的危害,完善了 Table-Miss 处理机制,有效遏制了由于流表溢出而引发的危害网络正常运行的情况.相比目前的 Table-Miss 处理方式,FTS 对流表溢出情况下控制消息数量和 RTT 时间的优化都达到两个数量级.此外,该文针对流表扩散方法设计了简单高效的基于 OpenFlow 组表的随机路由选择算法,系统结构实施简单,可以方便地降级为现行的通用 Table-Miss 处理模式.

关键词 OpenFlow;流表;溢出;Table-Miss;组表

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2018.02003

A Mechanism of Taming the Flow Table Overflow in OpenFlow Switch

QIAO Si-Yi[1),2)] HU Cheng-Chen[1)] LI Hao[1)] GUAN Xiao-Hong[1)] ZOU Jian-Hua[1)]

[1)](*Ministry of Education Key Lab for Intelligent Network and Network Security, Xi'an Jiaotong University, Xi'an* 710049)

[2)](*Science and Technology on Information Transmission and Dissemination in Communication Network Laboratory, Shijiazhuang* 050081)

Abstract Software Defined Networking is an emerging network architecture, which decouples the control plane from the data plane and operates the global network with elaborate abstraction. The flow table plays an important role in an OpenFlow Switch(OFS) and is the key resource to support the SDN/OpenFlow abstraction. To provide wire-speed processing, fast memory(e.g., TCAM, QDR, SRAM) is utilized to form the flow table. Unfortunately, the development of such kind of fast memories is far behind the hungry requirement on its usage, especially for the TCAM. As a result, the flow table installed in OFS has tremendous risk to be overflow, possibly leading to large quantity of Packet-In/Packet-Out messages between OFS and controller. Generally, an incoming packet from a flow is processed according to the action specified in the according flow entry in the flow table(s). If no entry is matched in the flow table, a packet-in message querying how to process the packet will be sent to the controller from the switch. If the number of active flows always touches the maximum number of entries in the flow table, the table-miss events are not avoidable. So that at first we investigate how to mitigate the overhead when occurring table-miss events based on the phenomenon of uneven flow table distribution. The basic idea is to distribute

收稿日期:2016-09-28;在线出版日期:2017-09-24.本课题得到国家自然科学基金(61272459)、国家"八六三"高技术研究发展计划项目(2013AA013501)、教育部新世纪人才计划(NCET-13-0450)、通信网络信息传输与分发技术重点实验室开放课题(ITD-U15004/KX152600013)资助.**乔思祎**,男,1990 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为软件定义网络、物联网.E-mail:qsy2011815@163.com.**胡成臣**(通信作者),男,1981 年生,博士,教授,主要研究领域为计算机网络、网络测量和数据中心网络.E-mail:huc@ieee.org.**李 昊**,男,1987 年生,博士,讲师,主要研究方向为网络测量、软件定义网络.**管晓宏**,男,1955 年生,博士,教授,主要研究领域为复杂网络优化、计算机信息安全.**邹建华**,男,1964 年生,博士,教授,主要研究领域为计算机视觉与模式识别、复杂系统分析.

the packets facing table-miss event in heavily loaded switch to other lightly loaded switches instead of triggering packet-in messages always in hot switches. The conceptual simplicity of FTS idea hides two significant challenges. (1) How to select a right port randomly by SDN switch. (2) How to make this progress "pipeline-able" in a general SDN switch without changing its Hardware. The new mechanism proposed in this paper to handle the Table-Miss event is named Flow Table Sharing (FTS). The evaluations have demonstrated that FTS reduces both control messages quantity and RTT time by two orders of magnitude compared to current state-of-the-art OpenFlow Table-Miss handler. We first build a switch in MININET (test environment), and measure the number of control messages generated by setting up a new flow transfer (TCP, UDP) when the flow table of the switch is overflow, as well as the packet loss rate and the average delay. Then, we evaluate the flow table demand in the optimal way that all switches have enough flow table resources and set this result as the control group. Then, on the one hand, we evaluate the additional flow table demand which is required by rebuilding the interrupted flow, when the FTS try to fix the problem caused by the overflow. On the other hand, we evaluate the total flow table consumption which is required by building the new transmission for the first time, after the overflow happened. Even during the flow table overflow period, denial of service for new flows does not happen. We designed an external user switch-computed Group Table select algorithm and show its validity and fastness. It is easy to implement, easy to control and the current state-of-the-art OpenFlow Table-Miss handler is a special case of FTS.

Keywords OpenFlow; flow-table; over flow; Table-Miss; group-table

1 引 言

1.1 背景介绍

软件定义网络(Software Defined Networking, SDN)将网络解耦为控制平面和数据平面,网络业务抽象为控制平面内网络操作系统中的应用程序[1-2]. 网络管理员以及开发人员利用软件定义网络架构编写网络应用程序,可快速重定义网络配置. 软件定义网络数据平面内的数据包由 SDN 交换机统一负责转发,SDN 交换机与控制器之间通过南向接口协议来传递配置信息. 为提高 SDN 网络的开放性,OpenFlow 南向接口协议[3]目前已经成为 SDN 的事实标准. OpenFlow 协议中描述了 SDN 交换机匹配数据包的流水线,以及数据平面与控制平面之间的消息类型. 当数据包成功匹配一条流表项,其表项相应的动作集(Actions)将被执行. 如果数据包没有被任意一条流表项匹配,这种情形将被定义为 Table-Miss(未匹配). 目前,处理 Table-Miss 的方式为丢弃数据包或者将数据包发送到控制器端,再由控制器端最终决定此数据包的处理方法.

为支持高速的数据包转发(40 Gbps、100 Gbps)[4],交换机内必须使用由快速缓存器(例如,TCAM,QDR、SRAM)构成的查找模块. 其中,SDN 交换机带掩码转发性能关键取决于基于三态内容寻址储存器(Ternary Content Addressable Memory, TCAM)构成的查找流表(Flow Table). TCAM 具有单周期查找以及掩码查找功能[5],然而由于其功耗、价格极高,所以基于 TCAM 的流表容量通常比较小,并且极有溢出的可能[6]. 流表溢出会引发数据包 Table-Miss、数据包丢包现象,使得交换机转发性能大幅降低,甚至会产生 SDN 控制报文数量爆炸,给 SDN 控制器的安全造成隐患. 为了适应真实流量的快速变化的特征,交换机需要支持快速的流表更新速率和大容量查找表. 目前商用 SDN 交换机最大流表更新速率一般不会超过 10000 条/s[7].

我们从新西兰 ISP 的一个出口带宽为 3.7 G 的路由器端口捕获的实时流量[8]中统计了流量的(15 min,约 2 千万个数据包)变化特征,发现增大流表容量可以减少流表对更新速率的需求. 并且做了定量分析,使用近期最少使用算法(Least Recently Used, LRU)为流表更新策略. 我们发现当流表容量为 10 k 时,流量中变化速率超过交换机更新性能的包的个数将达到 10%. 流表容量再扩大到 50 k 条,有 6%的

包无法及时更新. 如果再扩大到 250 k 条，仍有 5% 的包无法得到更新，其特征趋势由图 1 示出. 我们发现，当流表容量以指数增大时，设备对流表更新速度需求的缓解并不显著. 这种情况在嗜流表容量的 SDN 网络中会更加突出，根据 OpenFlow 协议，单条流表项将会达到 45 个域、最大 144 Bytes 的宽度. 这意味着若想维持同样深度的流表项，交换机需要更多的流表空间.

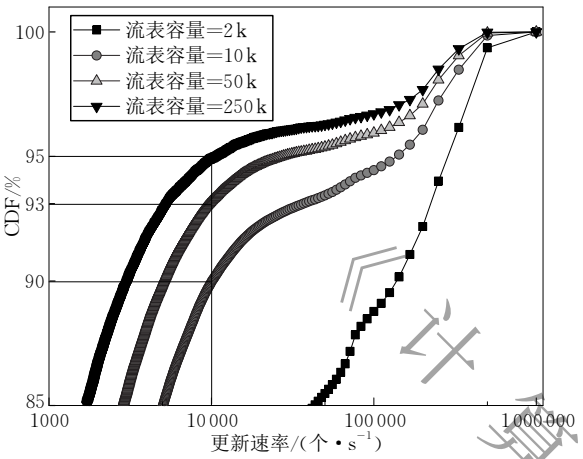


图 1　流量更新速率的概率累积分布

面对上述这些问题，为使 SDN 的技术优势在广域网间应用，目前业界不得不使用、购买性能更强、更昂贵的 SDN 交换机硬件. 在实际应用中，若 SDN 交换机的流表资源不足，对于最终来不及更新的那部分流，交换机会将它们以 Table-Miss 的形式处理. 从目前的控制策略来看，每一个 Table-Miss 事件，SDN 交换机都会给 SDN 控制器发送一个 Packet-In 消息. 当控制消息过多时，会引发控制器运算能力阻塞，阻碍控制平面对数据平面的即时控制能力，进而导致数据平面转发效率下降等问题.

1.2　相关工作

总结目前的相关工作，提升流表更新速率是解决 Table-Miss 问题的关键. 优化由 TCAM 组成的高速查找器的更新时间，更是解决提升速率问题的核心. 通常使用 TCAM 的查找速度快于由其它存储器组成的查找器，这点对高速交换机的实现是优势. 但是，基于 TCAM 的流表项删除、替换与更新的操作都比较复杂，并且 TCAM 有效容量低，能够存储的流表项条数又比较少，这又是设计人员不得不面对的问题.

在研究解决如何提高流表更新速率的问题时，研究人员通常从以下两方面入手：

(1) 提升流表项的数量. 显然，当 TCAM 内存容量固定时，降低单条流表项存储宽度可以增加总流表项的存储深度. 在 SDN 数据中心应用场景下，重新定义流的标识头，可压缩单条流表项存储宽度，例如将所有流分配到 16 bits 宽度的匹配域[9]，数据包离开设备之前再将原包头还原. 然而这种方式仅仅支持在数据中心等局域网使用 SDN 技术，无法向广域网推广、可扩展性比较差、适用范围窄.

(2) 提高数据平面流表的更新速度. SDN 交换机流表替换动作由控制器指挥，通常流程：SDN 控制器首先删除一条表项，之后在合适的位置上再添加一条新表项. 一般 TCAM 流表存储经过优化后，流表项之间有互相依赖[10]，若删掉一条与其它有互相依赖的流表项，会引发复杂的 TCAM 转存处理. 因而改写和删除 TCAM 表项相对于转发查找的速度是非常缓慢的. 若流表剩余空间足够大，控制器可以直接在空闲空间添加新表项，不去更改已经存在的流表项，这样可以节省更新流表的时间. Kannan 等人提出一种更新机制[11]，提前探测流表中的死流并将其删掉，为添加新流表项做空间预留工作，这样能够降低流表溢出的风险，提升更新速度.

这类方法的核心目标是增加交换机有效流表空间，尽量拖延流表溢出的发生，然而没有考虑流表溢出后导致网络性能下降的问题. 当流表溢出现象发生时，现有工作对缓解网络压力依然无效，也没有给出如何应对流量峰值所造成服务中断的解决办法. 本文正是针对这一研究上的不足，提出了一种流表共享机制(Flow Table Sharing，FTS). 其关键思路为借用邻居节点空闲流表资源缓解决本地流拥塞. FTS 部署简单，耗费硬件资源极小. 为避免流表查找策略的冲突，FTS 修改了 OpenFlow 协议中对 Table-Miss 的处理方式，当流表溢出后对未匹配的流使用本地路由策略转发，无需立即请求控制器，也不会在本机内安装新流表项.

我们会在第 2 节详细分析流表溢出造成的性能危害，以及分析从哪些角度可以最大程度避免性能的削弱；在第 3 节，对 FTS 的设计进行详细分析；第 4 节介绍交换机如何具体实现 FTS 以及关键算法；在第 5 节我们评估 FTS 的性能以及其它代价；第 6 节是本工作的总结和对以后发展方向的评估.

2　流表溢出问题分析

2.1　流量细粒度趋势

SDN 架构将网络抽象为数据平面和控制平面.

数据平面的转发设备简化为统一的 SDN 交换机，弱化了传统网络对不同设备的区分，仅将 SDN 交换机的核心功能抽象为安装流表和查找转发数据包. SDN 控制器负责管理整个网络以及计算 SDN 交换机内的流表项. 基于 TCAM 的 SDN 交换机内部转发表，可快速查找每一个数据包的执行动作. 然而基于 TCAM 的查找表价格高(每兆 bits 芯片，人民币 2500 元左右)、功耗大(每兆 bits 芯片，15 瓦). 鉴于目前有限的流表容量，SDN 交换机无法对所有流进行长时间的维持.

如今网络不应该仅局限于 best-effort 服务理念，服务供应商、内容提供商更希望拥有细粒度地区别各种使用场景的能力[12]，以赚取差异化服务后能获取到的最大利润. 此外，数据中心内的流量工程[13]和虚拟化技术[14]，对网络流量细粒度管控都提出了更多的需求. 以 OpenFlow 为代表的 SDN 协议标准，刚好为这个趋势提供了网络设备支持超多域、超细粒度的方案. 最新的 OpenFlow1.5 协议中提出，OpenFlow 已经支持多达 45 个域的包匹配分类.

转发设备对于细粒度流线速查找的支持，需要依靠基于 TCAM 的高速查找芯片. 从实际制造成本的角度考虑，基于 TCAM 的各类商用 SDN 交换机流表容量只有数千条[7]. 其容量需求的增加，将直接导致网络设备厂商生产成本大幅提升，网络运营商购买设备成本的大幅提升，以及租用链路的网络内容提供商的成本大幅增加. 互联网产业界只能以花费更大的成本为前提，来获得赚取更大利润的可能性. 然而这个概率在投资回报到达之前是无从知晓的，这会打击厂商投资新设备变革的信心. 这对矛盾在现实中显然阻碍了网络创新的推动.

此外，由 TCAM＋SRAM 双存储器组成的转发表，可大程度增加流表数目的容量[15]. 考虑到 SRAM 流表的查找速度较慢，无法满足线速要求，交换机只能将老鼠流(流量极小的网络流)以及死流(超时流)放入 SRAM 空间内，并且必须保证活动的大象流(流量极大的网络流)维持在基于 TCAM 的储存空间内. 实际上例如在真实部署域间骨干或大型数据中心网络中运行的庞大流量需求是远远无法得到满足的[16]. 但是，研究人员依然期待看到让用户享受 SDN 技术带来的有益之处. 越来越多的实例表明 SDN 技术正在向更大的地域范围，更多的节点互联方向发展[17]. 流表数目的匮乏，在日后大规模网络部署中会对网络应用造成怎样的影响？该如何优化地利用有限的资源？都是现在亟待探讨的问题.

2.2 问题分析

由于 SDN 架构控制平面与数据平面物理上的分离，SDN 控制器与 SDN 交换机之间通常通过专用网络建立 TCP 连接进行通信，这个专用网络称为安全通道(Secure Channel). OpenFlow 协议将各类消息内容封装成应用程序接口，并以数据包的形式在安全通道内传输. 控制消息的包头开销和远程传输，将导致数据平面与控制平面之间产生消息的通信量与消息分发的延迟远大于传统网络[18]. 因此研究人员必须面对这两个源于 SDN 架构的挑战：(1)控制平面——超多消息，导致的控制系统庞大复杂、性能需求极高；(2)数据平面——因流表空间不足，产生转发性能瓶颈.

对于第一类问题，研究人员提出了诸如分布式多控制器架构[19]，控制器中间代理层等[20]，提升控制器消息处理的能力.

本文主要针对第二点数据平面的挑战进行研究，分析其可能带来的严重后果. SDN 交换机在数据平面可分为 SDN 软件交换机和 SDN 硬件交换机. SDN 软件交换机通常架设在商用服务器内，可以保存足够多的规则内容. 但是其交换转发速度慢，且无法高效地处理通配规则[21]. 所以，我们可以利用软件 SDN 交换机来应对流量较小、时间离散度比较大的老鼠流. 为保证大流量流的服务质量，SDN 交换机的查找匹配功能必须由硬件流水线来实现.

假设流的到达是服从参数 λ 的泊松过程，根据排队论我们可以得到如下结论：在交换机中必然存在某时刻，使得容量为 C 的流表充满，发生流表溢出现象. 这种情况会根据流粒度的不同定义而有不同的真实值，但是在理论上他们各自的溢出趋势是相同的. 证明过程参见附录 1.

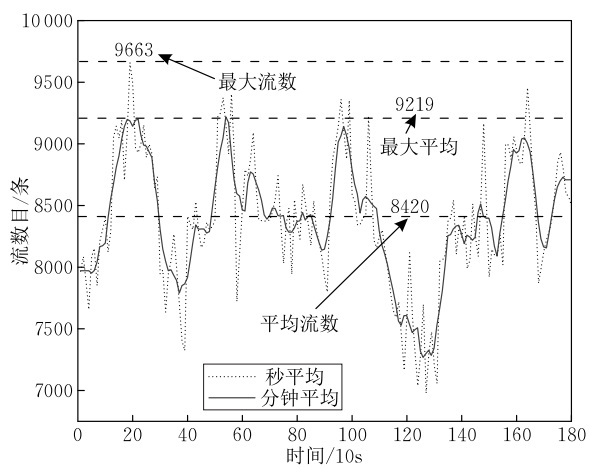


图 2 以 10 s 为窗口统计流数目

为得到对真实流量特征的统计，我们对 ISP 流量的流数目进行了分析，并随机截取了一部分，如图 2 所示为 ISP 路由器流数目的统计，在这 30 min 的时间段内平均流数目 $Avr=8420$. 假设控制器对全网流量调度(削峰填谷)以分钟为时间单位，那么在这种强度的网络流量需求下，最大分钟平均流表使用量为 $Max\_Avr=9120$. 然而，最多同时到达交换机流数目 $K=9663$. 根据图中数据计算：流数量超过 Max_Avr 的概率为 8.9%，而剩下 91.1% 的时间内流表平均有 7.6% 的空闲.

网络管理者应该根据 SDN 交换机流表容量来决定策略的最大细致程度. 根据 TCAM 使用比例动态调整流策略. 为使得经济效益最大化，我们希望减少流表空闲的概率.

约束条件：

$$C \leq Max\_Avr \leq K \tag{1}$$

$$Avr \leq C \tag{2}$$

(1) 容量 C 小于最大平均值 Max_Avr 有助于提高流表平均使用率；(2) 为满足正常转发的冗余性需求，平均使用数目又得小于容量 C.

当交换机以一定的概率发生流表溢出时，我们分析此时对转发设备带来的性能危害. OpenFlow 协议规定，当控制器准备对交换机下发流表项时，如果交换机流表已经被装满，交换机会向控制器报告消息〈ofp_error_msg，$OFPFMFC\_TABLE\_FULL$〉，并且交换机端拒绝安装这条流表项. 控制器得到错误报告消息后，可以选择忽视此流，也可以执行流表替换规则：(1) 忽视此流，造成拒绝服务现象；(2) 如果控制器采纳此流，则需先删除一条已有流表项，再重新添加此流. 但是如果被删除的已有流表项是一条活跃流，继而又造成这条活跃流暂时中断服务. 那么会有极大的可能导致数据包传输时延增大，传输带宽变小，甚至丢包. 我们将在第 5 节展示流表溢出后通信效果变差的实验结果.

接下来我们以一个实例来具体分析以上论述：如图 3 所示，四个节点组成的简单拓扑. 每台 SDN 交换机都与一个 SDN 控制器相连. 假设在某一时刻，S2 交换机内流表项被填充满. 之后如果另有新流到达，按照现有 OpenFlow 协议并针对 S2 交换机分析其操作流程：(1) S2 交换机查找表项无匹配结果，并触发 Packet-In 消息；(2) 控制器接收到 Table-Miss 消息后需要下发新表项，并且控制器计算判断出 S2 流表已满，选择一条流表项删除；(3) 控制器下发新流安装到 S2；(4) 新流转发建立成功.

值得注意的是控制器删除的那条原有流有可能是一条活动流(active flow)，还没有超时，此流后续的数据包会快速到来. 但是控制器无法轻易得知哪条流是最近最长时间没有被匹配的，控制器很难按照最优化的方式删除活动流. 当这条流表项被删除后，后续到达的数据包又会被交换机判别为新流，网络控制闭环会重复前面的过程. 新流频繁到达将会导致在安全通道内充斥大量重复控制消息数据包，从而耗费了控制器的计算处理资源，占用了安全通道内有限的通信带宽，也降低了交换机的转发效率.

另外我们也可以假设：控制器发现流表存储已满，不再向交换机内添加新流表项. 这种方式保证了已有流量的正常转发，但会导致系统必须放弃对新流的响应. 综上所述，本问题最根本的原因是由于流量的起伏波动，流数目暂时越过了交换机容量的上线，导致单个交换机性能下降，进而导致整体网络服务质量下降.

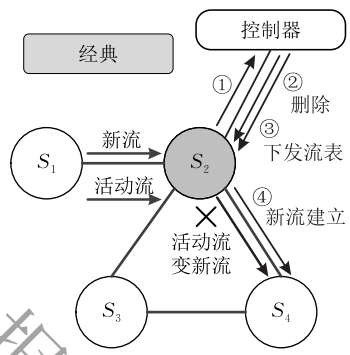


图 3　经典的新流处理过程

3　流表共享机制设计

针对之前两章节所遇到的问题，我们提出了一种流表共享机制，该机制的设计目标是缓解由于流表溢出而造成的转发能力下降或暂时无法提供服务的现象. 流表共享机制从两个关键点进行优化：(1) 允许流表溢出的交换机转发新流；(2) 减小重复控制消息的数量.

3.1　允许流表溢出的交换机转发新流

问题：没有建立流表项的流无法被交换机转发转发. 经典的〈匹配-执行〉操作方式，规定数据包在被匹配成功之后才能被处理. 当交换机内没有容纳新流的流表空间时，新流匹配表项无法被安装，交换机也无法对新流执行任何操作. 然而我们希望当交换机流表溢出时允许转发，这与现行的模式相矛盾.

解决思路：我们为交换机增加了随机转发模式.

这与〈匹配-执行〉不同，随机转发模式将流量随机地转发到邻居节点.这种方式没有宏观的策略指导，转发方向只要不等于入端口方向即可，从而避免路由环路的产生.因而这条本可能被拒绝服务的流，就出现在了邻居节点上.邻居节点与本节点同时发生流表溢出的概率很小，因此这条流会有更大的可能性成功地在邻居节点上建立正确的策略性转发路径.

3.2 减小重复控制消息的数量

问题：新流频繁到来，引发重复的 Packet-In 消息.在 OpenFlow 协议下，SDN 交换机收到无法匹配的流后，会触发 Packet-In 消息.Packet-In 消息携带了包头信息给控制器，作为控制器判断如何进行包处理的信息依据.若到达的流数量超过了交换机的匹配容量，那么对交换机而言，在当前时段内无法处理的包都可归为新流.这不得不导致携带相同包头信息的控制消息反复在安全通道内传递，降低有效控制信息通信效率.

解决思路：我们将通配转发引入到随机转发模式，通配任何数据包包头.因此对于任意一条额外的新流，都会被匹配，交换机总能保证此数据包被处理，从而不产生重复的 Packet-In 消息.下一章详细论述 FTS 结构如何设计与实施.

4 随机路由策略与实现

为抑制在流表溢出时因更新流表而产生重复控制消息，解决因新流没有相应的〈匹配-执行〉信息而无法获取转发服务的现象，我们提出了〈随机转发-流表扩散〉的方法.溢出的节点利用邻居节点内空闲的资源，重新建立一条从邻居到目的地的转发路线.

此方法需要在一定的前提下才可正常工作，显然，如果网络中的交换机都随机转发，必然导致数据风暴或者数据包无法抵达目的地.因此我们假设，网络中流表溢出只存在于随机的少量的转发节点中.利用反证法证明：如果网络内大部分结点都发生流表溢出，那么此情境下的网络必处于全面拥塞的状态.根据实际情况，多数时段内网络并不会总出现大规模的全面拥塞.因此网络内流表总是有足够的空余，能满足基本的数据包波动.另外，控制器会实时地根据拥塞情况调整网络流的控制粒度，因而流表项的使用只会趋于一种用尽的动态稳定，再次从宏观角度保障不会有大规模流表溢出的现象发生.

图 4 是对应于经典 SDN 网络，采用了缓解机制后的网络结构图.假设交换机 S2 流表溢出，若此时新流到达，匹配为 Table-Miss 且交换机不产生 Packet-In 消息.即在交换机流表溢出后，仅在本地执行离线策略.通过一定的本地计算，快速得到新流的转发动作，从而避免将新流的数据包丢弃.其中的快速算法：随机转发，应满足线速需求.本例中，新流被随机转发到 S4 中，且之前建立的活动流不会受到本地离线策略的影响，可以继续转发.新流到达 S4 后，便可在此邻居节点上重新实施路由.下面的两小节将详细说明交换机离线策略的设计以及实现方式.

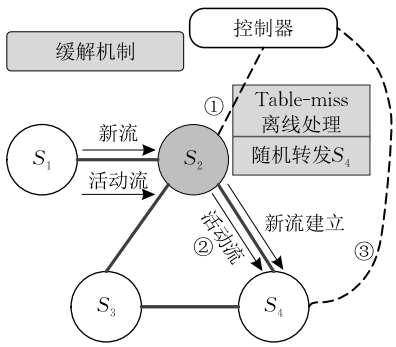


图 4 流表溢出后 FTS 对新流的处理过程

4.1 随机路由离线策略设计

我们将交换机中的离线随机策略抽象为如下图 5 形式.

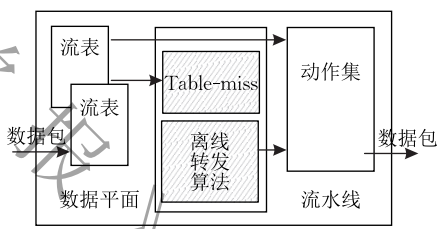


图 5 目标交换机流水线结构

数据包到达交换机首先查找流表，匹配成功后执行已经设定的处理操作例如，更新计数器，执行指令集等.之后判断流表是处于否溢出状态并且是否需要触发 Table-Miss 事件：如果没有溢出则直接执行最终的动作集，此处动作集包括但不限于触发 Packet-In 消息；如果流表溢出，则在交换机本地计算数据包的转发端口.

包含缓解机制的交换机与经典 SDN 交换机的区别只在发现 Table-Miss 事件时，判断交换机是否处于流表溢出状态；以及在交换机离线处理策略中的 out_port 端口计算算法的实施.为支持交换机线速转发，以上算法只能在交换机的硬件快速流水线中实施.修改交换机硬件流水线是一件比较困难的事情，然而我们发现流水线内的 flow-table，指令集，动作集都可以灵活配置.因此我们只能考虑将此

算法与 SDN 交换机流水线内拥有灵活性的组件耦合起来.

我们将设计的总体流程总结如下.

(1) 流表匹配. 经过流表匹配之后，可以得到 packet 是否为 Table-Miss.

(2) 根据 metadata 以及交换机自身判断提供的流表使用率，判断是否需要本地计算.

(3) 流水线内的 out_port 算法计算数据包的转发端口.

(4) 执行动作集.

其中必须满足的限制条件：

(1) 随机转发之后的出端口(out_port)不能等于入端口(in_port)，否则会产生环路.

(2) 为满足吞吐率，out_port 随机算法必须简洁高效.

限制条件带来了 2 个硬件实施上的挑战：

(1) 实现随机. 没有现成的指令集可以完成随机指定动作集.

(2) 目前 OpenFlow 协议规定的硬件流水线还没有明确判断流表是否溢出的流水线级. 贸然修改 SDN 交换机流水线则改变了标准的协议，这样会导致交换机可扩展性差的缺点.

为应对以上的挑战，我们选择以组表作为 SDN 交换机实施随机转发的关键组件.

4.2　随机路由策略组表方式实现

4.2.1　OpenFlow 组表

OpenFlow 协议规定 SDN 交换机需包括组表(group-table)，组表包含多个动作桶项目. 每一个动作桶都包含一个可执行动作集和相关参数. 组表类型有 ALL，SELECT，INDIRECT，FAST FAILOVER 共四种. 我们选择组表的 SELECT 操作类型.

SELECT：一个数据包只被组内的一个桶执行，这个动作的选取基于交换机的用户选择算法. 例如哈希某些用户配置的域，或者简单的循环选择. 选择算法的配置和状态不属于 OpenFlow 协议的范围内.

如图 6 所示，组表是交换机流水线动作集的一种形式. 组表将多个动作集内容通过一定算法联到一起，将给数据包带来更灵活的操作方式. 组表包括组表编号字段、组表类型字段、计数器和桶. 针对不同的组表类型，桶的执行方式也有所不同. OpenFlow 协议规定组表的 SELECT 算法可以由外部用户自己定义，此选择算法用来选择任意一个可执行的桶，如果当属于一个选定桶的动作集出端口丢失连接，逻辑会自动将桶范围缩小至剩余的可用桶并重新选择.

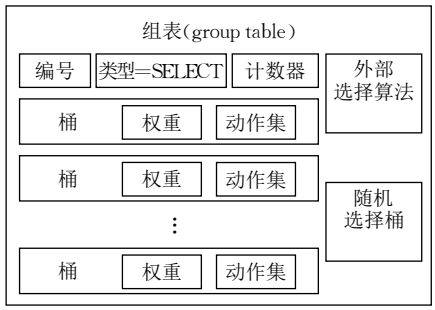


图 6　OpenFlow 组表结构

一些标准的选择算法已经固化在硬件组表内，例如，使用哈希算法来分类用户配置的包头元组，或简单的令牌环. 至此，OpenFlow 组表解决了上一节提出的两个挑战，即利用随机选择桶算法实现随机，组表天然的又是硬件流水线内的一部分，可以满足线速转发的需求.

下一小节讨论：基于上一节提出的限制条件，我们应该如何设置流表以及组表，即：(1) 出端口不能等于入端口；(2) 流水线随机算法必须简洁高效.

4.2.2　组表桶的设定

桶内的动作集用来指定数据包转发出口，权重表示执行此桶的概率大小. 随机扩散机制要求向邻居转发. 那么桶内的动作集中转发出端口(out_port)的值应该对应自己所有的邻居节点. 显然，数据包出端口不能包括入端口，否则与我们之前的策略相矛盾. 在这里我们有两种方式解决：(1) 结合流表二级匹配，区分并剔除源端口；(2) 采用额外组表选择算法剔除入端口.

(1) 结合流表二级匹配. 流表的匹配域为 in_port，设置对应的 group_id＝in_port 的组表. 此组表内的桶不包含转发到 PORT＝in_port 的动作集. 之后，由令牌环实现简单高效的随机算法. 并且将此流表项配置为最低优先级，保证此流是将要产生 Table-Miss 事件的. 例如，交换机有 N 个邻居，我们需要添加 N 条流表项，以及配置 N 个组表，每个组表内包含 $N-1$ 个 buckets；空间复杂度为 $O(N^2)$. 另外，我们考虑流表溢出的判断：其一，我们可以利用控制器判断流表使用情况，一旦溢出，控制器才向交换机内填写 N 个扩散流表项，但是这会导致较大控制延迟. 其二，我们可以利用外部选择算法，在交换机本地判断，控制器可以将 N 个扩散流表初始化到交换机内.

(2) 采用额外组表选择算法. 将组表内的 selection 算法加以扩充，只需要一个组表，组表内动作集包括转

发向所有的N个邻居和1个控制器(CONTROLLER).如果算法判断交换机流表使用正常没有溢出,直接选择转发向 CONTROLLER 的桶.在溢出的情况下,选择 out_port 时算法同时比较 in_port,随机算法可以用简单令牌环,一旦发现 out_port 与 in_port 相同则向后跳一个桶执行即可.

总结方法 1:空间复杂度大:其中流表$O(N)$、组表$O(N^2)$;初始化延迟大,但是无需用户大量修改组表内的固有的选择算法.

总结方法 2:空间复杂度低:流表$O(1)$,组表$O(N)$,初始化无延迟,但是需要用更换组表内的选择算法.

由于 OpenFlow 协议对于 SELECT 算法并没有强加标准,规定 SELECT 算法是 SDN 交换机内的可选组件,那么我们更倾向于使用第二种方法,选择定义一种可扩展性强的 SELECT 算法.算法 1 列出了选择算法的伪代码.

算法 1. 外部选择算法.

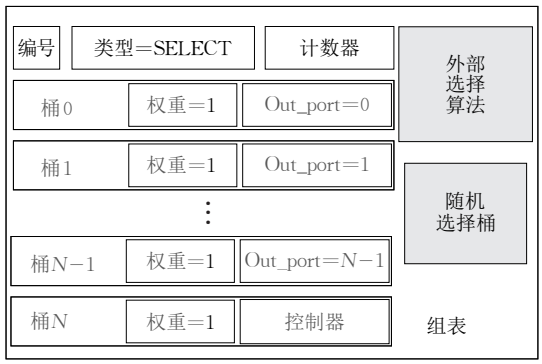
输入:in_port

输出:out_port

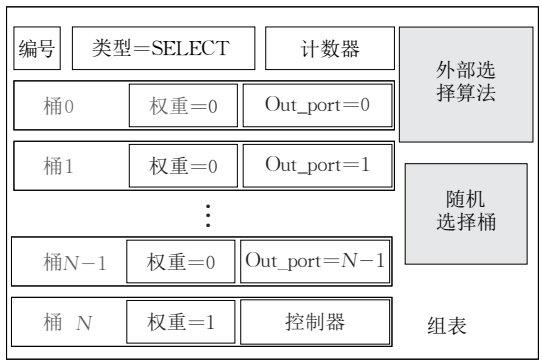
```
1. WHILE New packet arrival
2.    IF not overflow
3.       out_port=CONTROLLER
4.    ELSE IF overflow
5.       out_port=(out_port+1) mod N
6.       IF out_port==in_port
7.          out_port=(out_port+1) mod N
8.       END IF
9.    END IF
10. RETURN out_port
11. END WHILE
```

如图 7(a)所示,方框内部表示标准 OpenFlow 协议规定的组表结构,组表包括了:(1)组表编号;(2)组表类型;(3)计数器;(4)一个或多个桶;(5)随机选择算法;(6)外部选择算法.默认随机选择算法是硬件令牌环算法.

如图 7(b)所示,外部选择算法从逻辑上,是 Table-Miss处理的延伸.这是因为桶内的权重为 1,可被作为 out-port 的候选对象.若某 bucket.weight=0,选择算法会将此 bucket 对应的 action 屏蔽.如果所有 bucket 的 weight 都为 0,此数据包会被丢弃.所以目前 Table-Miss 的处理方式可以由 group 方式简单表达.除能扩散算法也很简单,只要将前 N 个 buckets 的 weight 设置为 0 即可.或者,设置某些 bucket 的 weight 为 0,即可关闭向对应邻居扩散.



(a) 使能FTS配置



(b) 不使能FTS配置

图 7 组表的配置方式

至此已完成流表溢出缓解机制的设计介绍,下面我们根据实验结果,对流表共享机制进行评估.

5 实 验

实验测试分为 2 个部分,分别是:(1)利用软件仿真平台,对比 FTS 与优化之前转发性能的优劣;(2)在真实拓扑上测试流表共享机制的实际部署开销.

实验的第一部分转发性能测试使用了软件仿真平台,由于软件仿真在性能上与真实转发设备有一定的差距,因而我们需要讨论软件仿真实验的系统误差估计,主要针对转发时延.计:在流表溢出后的流量总时延为 $T_{溢出}$,经过 FTS 优化后的流量时延为 T_{FTS},单个数据包转发时延 T_{fwd},单个数据包传输时延 T_{trans},传输时延统计误差 ΔT_{trans},转发时延统计误差 ΔT_{fwd}.由于控制器与交换机的实例化都在同一台计算机的网卡内,因此数据包的传输时延将远小于真实值.即

$$\text{真实 } T_{trans} = \text{实验 } T_{trans} + \Delta T_{trans} \tag{3}$$

实验使用软件交换机,由于软件性能无法与真实转发硬件相比,因而转发操作造成的转发时延会略大于实际值.即

$$\text{真实 } T_{fwd} = \text{实验 } T_{fwd} - \Delta T_{fwd} \tag{4}$$

综上，实验的系统误差来自于转发时延的测量，和传输时延的测量. 即

$$\zeta_{\text{流量总时延误差}} \propto \Delta T_{\text{trans}} + \Delta T_{\text{fwd}} \tag{5}$$

由图 7 可知，软件交换机通常情况下转发延迟小于 0.2 ms，对于真实转发硬件转发时延将会远远小于 0.2 ms，因而：

$$\Delta T_{\text{fwd}} \propto 10^{-1}\ \text{ms} \tag{6}$$

一般对于远程 TCP/IP 数据包传输的时延，可以估计在 100 ms 数量级. 因而：

$$\Delta T_{\text{trans}} \propto 10^{2}\ \text{ms} \tag{7}$$

实验总时延 T 由转发时延和传输时延组成：

$$T_{\text{实验总时延}} \propto T_{\text{fwd}} + T_{\text{trans}} \tag{8}$$

真实总时延 T 由实验 T 与误差组成，并且根据式(3)～(8)得到：

$$\begin{aligned} \text{真实时延}_{\text{溢出}} &\propto T_{\text{fwd}} - \Delta T_{\text{fwd}} + T_{\text{trans}} + \Delta T_{\text{trans}} \\ &= \text{实验 } T_{\text{溢出}} + 99.9\ \text{ms} \end{aligned} \tag{9}$$

$$\begin{aligned} \text{真实时延}_{\text{FTS}} &\propto T_{\text{fwd}} - \Delta T_{\text{fwd}} + T_{\text{trans}} \\ &= \text{实验 } T_{\text{FTS}} - 0.1\ \text{ms} \end{aligned} \tag{10}$$

针对式(9)～(10)给出的结果，我们可得出结论：

(1) 将测量值的误差引入后，溢出后的总时延高于其它值两个数量级以上，并且误差不改变自身测量值的数量级(均为 10^2 ms 数量级).

(2) 由于时延误差 10^{-1} ms≪7.56 ms 的测量结果，则此误差可以忽略，因此 FTS 优化后的真实值也满足对其它两个数据的数量级差距. 因而我们证明了性能测试实验使用软件仿真完全具有代表性，也能保证结果的准确度. 此外对于性能测试实验来说，获得一个真实的 SDN 交换机造价昂贵，且由于不易配置导致实验手段不灵活，此番论证后可显著降低实验的困难程度，易于科研工作者进行重复测试.

5.1　性能测试：转发时延与控制消息数量

我们评估流表溢出对通信速率和安全通道消息量与正常通信情况下的差异，并且将之与实施流表共享机制后的结果相比较. 实验平台为 x86 服务器，Intel i3 CPU，8 GB 内存，ubuntu14.04，远程 ryu 控制器. 如图 8 实验拓扑图所示，性能测试中使用 MININET[22] 仿真工具建立了单节点转发拓扑，试用 iperf 工具分别测试 RTT 以及转发速率. 利用 wireshark 抓包工具分析安全通道内控制消息的数量. 经过多次 iperf 流量导入测试后得到结果，当流表溢出时 UDP 数据包有 15% 的丢包率，TCP 数据包没有丢包.

如图 9 的右边子图所示，当流表溢出时，UDP 与 TCP 通信均产生了大量控制消息，分别比正常情况高出 1 个数量级到 2 个数量级. 由于流表溢出导致既有表项被删除后又重新下发流表消息，导致控制消息大量增长. 而 TCP 通信消耗量大通过 TCP 的重传机制可以合理的解释：TCP 为保证所有包都到达过目的地，因而在丢失的情况下会重传数据包，这部分多出来的数据包传输量将会消耗更多的控制信道数据包. 当交换机采用 FTS 策略时，流表溢出后控制消息数量只比正常情况多出 13.5%，比最差情况优化了 2 个数量级.

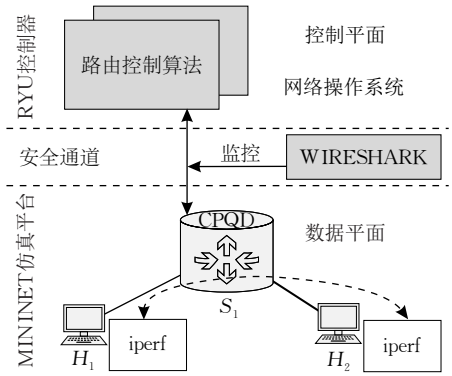


图 8　性能测试实验拓扑图

如图 9(a)所示，正常通信时一个数据包的转发平均时延为 0.227 ms，当流表溢出后转发平均时延增加为 768 ms，通信时延增加了 4 个数量级. 这是由于流表溢出后正常通信流表项会被后到的新流替换掉，导致流量中断，重新装载流表项需要再次耗费建立时间. 这将导致总的转发时延增大，以及吞吐率的下降. 同时我们看到，当经过 FTS 优化后，再次发生流表溢出时，转发延迟从 768 ms 下降到 8 ms 以内，优化了 2 个数量级. 这是由于 FTS 机制寻找到其它转发路径并建立转发，即可大大降低控制器的反复控制操作次数，并最多只需要多耗费建立一次流表项的时间.

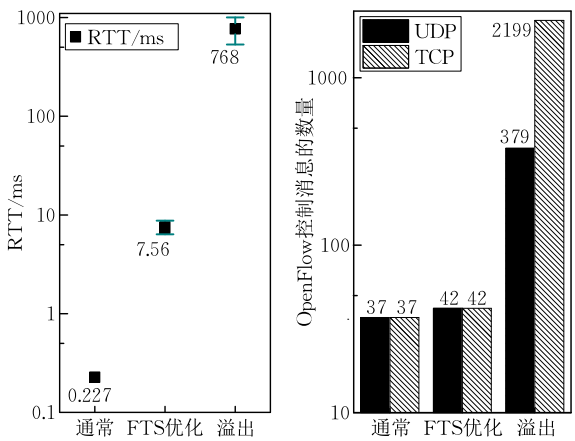


图 9　FTS 与普通交换机在流表溢出时的性能对比

5.2 代价测试:额外消耗流表项资源与路径

我们测试,实施 FTS 之后所需要的额外的转发资源,以及在流表容量冲突下,建立非最短路径后额外的转发距离.

我们选取 CAIDA 公布的自治域真实拓扑[1]① 统计扩散后数据包到达目的地的总转发跳数.此拓扑包含 6744 个结点和 26501 个边,结点度数分布可见图 10.我们将此拓扑的每个自治域抽象成一个转发节点,这样可以保留真实拓扑数据,也可以简化实验.

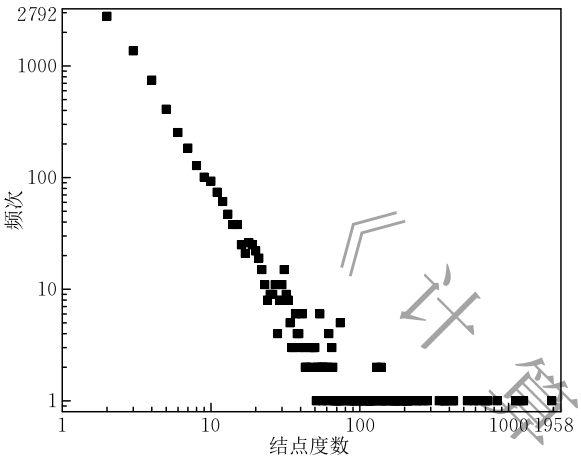


图 10 实验用的真实拓扑结点度数特征分布

为减小流表溢出对交换机带来的副作用,作为代价,FTS 会使用其它资源来完成转发.由于扩散的随机性,到达目的地会出现比最短路径距离长的情况.我们计算了路径增加、总路径长度、路径长度增加的比例,以及额外流表使用量.由于受到通用仿真工 MININET 的性能限制,上千个节点的仿真无法用其完成.

第二个实验主要观察路由计算以及流表项数量的统计,因而其实无需使用 MININET 完善的仿真能力,反而浪费了实验的计算性能.我们使用 C 语言来实现本实验,只考虑转发数据包的逻辑行为,保留交换机的拓扑关系特征,忽略交换机的其它仿真特性(例如,转发时延、端口、流水线等).同时也忽略 SDN 网络的立体结构(控制平面、管道、转发平面),将网络能力抽象为理想的扁平网络,网络功能只包括路由和流表共享算法.

仿真网络的拓扑和功能简化之后,还需要流量才能组成完整的实验.真实网络中的流量巨大,链路多,端口链接复杂,我们不可能得到完整的网络真实流量细节,其次即使得到所有流量信息我们也不可能在仿真平台上再现当时网络情况.目前,用真实流量进行网络仿真,一般需要专用的流量捕获和回放设备[23],这需要专门的软硬件系统,造价昂贵,设备体积庞大.在网络中的每个端口都需要串接一台该种设备,才能捕获同一时间的所有流量.由于带宽大,捕获设备只能保存几分钟,甚至几秒的流量.在测试时,也只能针对单节点网络设备进行测试.用此方式完成全网数据仿真,无异于重新搭建整体网络,显然不现实.因此我们计划直接通过算法得到统计值,或者最好能得到概率密度.

我们的基本思路如下:

(1) 流量仿真,生成拓扑的端口到端口完全覆盖所有点对点的流,这样可以保证点点之间互达,流表信息也最完整.

(2) 溢出仿真,假设每次仿真均有一个节点发生流表溢出.针对这个流表溢出的节点,实施流表共享算法.分别得到向所有邻居节点随机转发后的变量结果,得出统计值后归一化,以表示统计数据的概率密度.

此番假设的不足之处:我们假设网络中点点均可达并通信,然而实际情况下在同一时间内网络流量不是点点互达的.

因此通过实验我们只能得到一个统计值的长时间平均概率密度,这是整体网络长时间运行后的一个真实统计值的渐近线.但面对庞大无法完成的仿真任务量,在实验精度上的这种取舍依然是值得并且有效的.

图 11(a)显示了 FTS 建立连接过程中使用流表项数目的累积概率分布.由于 FTS 借用了邻居交换机的流表项,流表总使用率与邻居个数成正相关.三角图线是正常情况下的流表使用数量,圆形线条是对新流实施 FTS 所需要的流表项数量,方块线条是对正常流突然中断后使用 FTS 立即恢复通信所需要的流表项数量.我们还计算了扩散之后流表占用数目的变化情况.方块线条表示一条活动的流被删除后发生扩散所占用的流表资源;圆形线条表示新流因为流表溢出而无法建立连接的情况.前者平均新增占用流表 5.9,后者平均新增占用流表项 8.7.正常情况下新增流表项为 3.9.90%概率下新增流表项不会超过 15 条.99%概率下新增流表项不会超过 45.最差情况为增加 67 个流表项.

图 11(b)显示了 FTS 对数据包传输跳数的影响.三角图线是正常情况下数据包的跳数,圆形图线

① The CAIDA UCSD Macroscopic Skitter Topology Dataset www.caida.org/tools/measurements/skitter〈AS-level topology graphs〉

表示 FTS 会增加到达路径的平均长度，方形图线表示 FTS 的流对正常情况下多出的跳数. 我们看到 95%的概率 FTS 增加的跳数不会高于 2 跳.

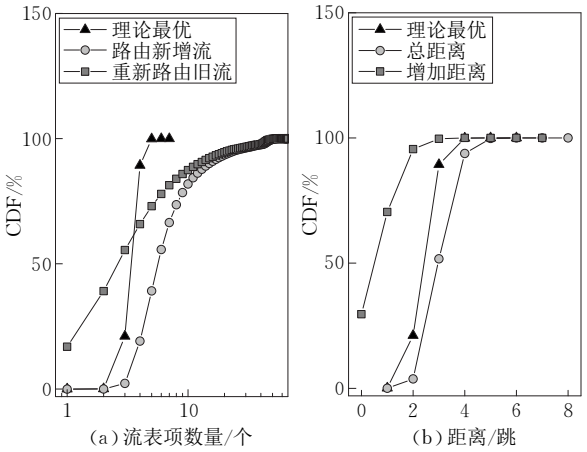


图 11　当流表溢出时，FTS 建立一条新流所消的耗额外流表项资源以及每个数据包从源到目的地的跳数累积概率分布

图 12 示出 FTS 处理一条新流走过的路径的长度增长情况，在 95%的情况下路径增长率不会达到最短路径长度的 199%. 我们将仿真代码和拓扑数据以及实验结果公开到 github① 上.

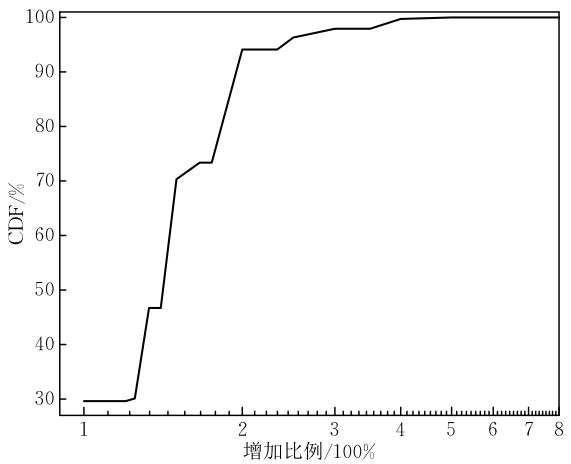


图 12　相比最短路径，FTS 完成路由增加的转发路径比例

FTS 不能保证同一 TCP 流往同一邻居扩散，并且其路径长度也不尽相同. 但是从图 9(a)的实验结果得到：包与包之间的路径差不会超过 7(8－1＝7)跳. GE[24] 网口的标准转发时延 200 μs，所以总时间差不会超过 2 ms. 这会导致 TCP 乱序重排机制启动，还有可能会增加传输时延. 但是，我们主要寄希望于 FTS 能够缓解因流表溢出所造成巨大的性能损失，跟流表溢出后转发设备的性能对比，乱序导致的性能下降其实是可以忽略的.

6　总　结

近年来新兴的 SDN、OpenFlow 交换机是学术界的研究热点，相比于传统网络的交换设备，SDN 网络交换设备将需要更大的查找表存储容量，流表溢出而带来的严重性能下降. 本文基于流表分布，从理论分析的基础上，提出了流表共享机制(FTS)，扩展了 OpenFlow 交换机的 Table-Miss 处理方式. 可有效缓解流表溢出现象带来的巨大的网络通信性能损失.

文中从两个角度验证了流表共享机制，从实验结论中可得出 FTS 机制有以下三个优点：(1) 在流表溢出时段内，FTS 相比于现在常用的 Table-Miss 处理模式可有效降低控制消息数量和 RTT 时延 2 个数量级；(2) 我们提出了一种有效的快速的外部组表选择算法；(3) FTS 易于在真实交换机上实施和控制，占用硬件资源少，现行 Table-Miss 处理模式是 FTS 机制的一种特殊形式.

转发硬件资源消耗量大和转发设备成本高，是目前 SDN 网络向广域网推广的重要困难之一. 其中流表资源的匮乏又是该问题的核心点，本文提出的缓解流表溢出机制，将为后续 SDN 全面发展提供一个可参考的容错机制. 另外，提高实验的精度是发现本机制存在问题的重要手段，这将作为下一阶段工作的重点. 通过实验的验证，可以进一步认识该网络系统的逻辑正确性，也能为 SDN 提供一个更好的落地.

致　谢　西安交通大学智能网络与网络安全教育部重点实验室 SDN 小组的同学之前在 SDN/OpenFlow 方面做了很多研究工作，为本篇论文的完成奠定了良好的基础. 实验室的老师给予了本课题很大的帮助和指导，感谢通信网络信息传输与分发技术重点实验室、国家自然科学基金的资助，在此衷心表示感谢，同时感谢在百忙之中评阅论文的各位专家！

参　考　文　献

[1] ONF Market Education Committee. Software-Defined Networking: The New Norm for Networks. Palo Alto, USA: Open Networking Foundation, 2012

① https://github.com/qiaosiyi/test_overflow_random_foward

[2] McKeown N, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74

[3] Heller B. OpenFlow switch specification v1.5.0. Palo Alto, USA: Open Networking Foundation, 2014

[4] Shahbaz M, Choi S, Pfaff B, et al. Pisces: A programmable, protocol-independent software switch//Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference. Florianopolis, Brazil, 2016: 525-538

[5] Katta N, Alipourfard O, Rexford J, et al. Infinite CacheFlow in software-defined networks//Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking. Chicago, USA, 2014: 175-180

[6] Qiao S, Hu C, Guan X, et al. Taming the flow table overflow in OpenFlow switch//Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference. Florianopolis, Brazil, 2016: 591-592

[7] Kuźniar M, Perešíni P, Kostić D. What you need to know about SDN flow tables//Proceedings of the International Conference on Passive and Active Network Measurement (ICPACM 2015). New York, USA, 2015: 347-359

[8] Allesina S, Bondavalli C. WAND: An ecological network analysis user-friendly tool. Environmental Modelling & Software, 2004, 19(4): 337-340

[9] Kannan K, Subhasis B. Compact TCAM: Flow entry compaction in TCAM for power aware SDN//Proceedings of the Distributed Computing and Networking (DCN 2013). Berlin, Germany, 2013: 439-444

[10] Meiners C R, Liu A X, Torng E. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. Transactions on Networking (ToN), 2012, 20(2): 488-500

[11] Kannan K, Subhasis B. FlowMaster: Early eviction of dead flow on SDN switches//Proceedings of the Distributed Computing and Networking (DCN 2014). Berlin, Germany, 2014: 484-498

[12] Li L E, Mao Z M, Rexford J. Toward software-defined cellular networks//Proceedings of the Conference on European Workshop Software Defined Networking (EWSDN). Darmstadt, Germany, 2012: 7-12

[13] Benson T, Anand A, Akella A, et al. MicroTE: Fine grained traffic engineering for data centers//Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies. New York, USA, 2011: 8

[14] Cerrato I, Annarumma M, Risso F. Supporting fine-grained network functions through Intel DPDK//Proceedings of the Conference on European Workshop Software Defined Networking (EWSDN). London, UK, 2014: 1-6

[15] Jiang W, Wang Q, Prasanna V K. Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup//Proceedings of the 27th Conference on Computer Communications. Phoenix, USA, 2008: 1786-1794

[16] 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

[17] Bosshart P, et al. P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95

[18] Curtis A R, et al. DevoFlow: Scaling flow management for high-performance networks. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 254-265

[19] Koponen T, et al. Onix: A distributed control platform for large-scale production networks. Operating Systems Design and Implementation, 2010, 10: 1-6

[20] Yu Minlan, et al. Scalable flow-based networking with difane. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 351-362

[21] Katta N, et al. Infinite CacheFlow in software-defined networks//Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HOTSDN 2014). Chicago, USA, 2014: 175-180

[22] Lantz B, Heller B, McKeown N. A network in a laptop: Rapid prototyping for software-defined networks//Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. New Delhi, India, 2010: 19-24

[23] Qiao S, Xu C, Xie L, et al. Network recorder and player: Fpga-based network traffic capture and replay//Proceedings of the International Conference on IEEE Field-Programmable Technology. Shanghai, China, 2014: 342-345

[24] Liang Yong-Sheng, Zhang Ji-Hong, Zhang Nai-Tong. Measurement and analysis of forwarding delay in Ethernet architecture within tolerances of the IEEE specifications. Acta Electronica Sinica, 2008, 36(1): 46-50(in Chinese)
(梁永生, 张基宏, 张乃通. IEEE 标准容限内以太网转发时延的测试与分析. 电子学报, 2008, 36(1): 46-50)

[25] Kleinrock L. Queueing Systems Volume 1: Theory. New York, USA: Wiley, 1975

[26] Bolch G, et al. Single station queueing systems//Proceedings of the Queueing Networks and Markov Chains Modeling and Performance Evaluation with Computer Science (MPECS 1998) Applications (1998). New York, USA, 1998: 209-262

[27] Barbeau M, Kranakis E. Principles of Ad-Hoc Networking. New York, USA: John Wiley & Sons, 2007

附录 1.

假设 SDN 交换机流表容量为 C 条,大流的到达形式是具有参数 λ 的一个泊松过程,大流生存时间是具有速率为 μ 的指数分布,N_t 为在 t 时刻 TCAM 中的大流个数. 在 t 时刻,N_t 是一个增消过程. 如果到达强度 $\rho>1$,N_t 将无边界增长,

所以需要满足：

$$\left(\rho=\frac{\lambda}{c\mu}\right)<1 \tag{11}$$

N_t的稳态分布概率密度函数为[25-26]

$$\pi_0=\left[\sum_{k=0}^{C-1}\frac{(C\rho)^k}{k!}+\frac{(C\rho)^C}{C!}\cdot\frac{1}{1-\rho}\right]^{-1} \tag{12}$$

$$\pi_k=\begin{cases}\pi_0\dfrac{(C\rho)^k}{k!}, & 0<k<C\\ \pi_0\dfrac{\rho^kC^C}{C!}, & C\le k\end{cases} \tag{13}$$

π_k代表流表中有 k 个流表项的概率.

参考爱尔兰公式[25]，流表充满的概率为

$$C(C,\lambda/\mu)=\frac{\left(\dfrac{(C\rho)^C}{C!}\right)\left(\dfrac{1}{1-\rho}\right)}{\sum_{k=0}^{C-1}\dfrac{(C\rho)^k}{k!}+\dfrac{(C\rho)^C}{C!}\cdot\dfrac{1}{1-\rho}}=\frac{1}{1+(1-\rho)\left(\dfrac{C!}{(C\rho)^C}\right)\sum_{k=0}^{C-1}\dfrac{(C\rho)^k}{k!}} \tag{14}$$

平均到达交换机的流数量[27]：

$$Avr=\frac{\rho}{1-\rho}C(C,\lambda/\mu)+C \tag{15}$$

策略分配时粒度越细，交换机内的流表容量需求量就越多. 由式(13)中超过 C 的概率密度，以及式(14)，显然不论 ρ、C、μ 取何值总有概率 $C(C,\lambda/\mu)$ 使得 $k\ge C$.

QIAO Si-Yi, born in 1990, Ph. D. candidate. His research interests include software defined networking, network security and Internet of Things.

HU Cheng-Chen, born in 1981, Ph. D., professor. His research interests include networking system, networking measurement and monitor, datacenter network.

LI Hao, born in 1987, Ph. D., lecturer. His research interests include networking measurement and SDN.

GUAN Xiao-Hong, born in 1955, Ph. D., professor. His research interests include complex network optimization, network security.

ZOU Jian-Hua, born in 1964, Ph. D., professor. His research interest covers image processing, complex systems analysis, and networked control systems.

Background

Software Defined Networking (SDN) is an emerging network architecture, which decouples the control plane from the data plane and operates the global network with elaborate abstraction. For the opening and the extensive use of OpenFlow, the OpenFlow switch is a *de facto* specification of SDN. The flow table plays an important role in an OpenFlow Switch (OFS) and is the key to support the SDN/OpenFlow abstraction. However, to provide wire-speed processing, fast memory (e. g., TCAM, QDR, SRAM) is utilized to form the flow table. Unfortunately, the development of such kind of fast memories is far behind the hungry requirement on its usage. As a result, the flow table installed in OFS has large risk to be overflowed, possibly leading to large number of Packet-In/Packet-Out messages between OFS and controller.

In this paper, we investigate how to mitigate the overhead when occurring Table-Miss events based on the phenomenon of uneven flow entry distribution. The basic idea is to distribute the packets facing Table-Miss event in heavily loaded switch to other lightly loaded switches instead of triggering Packet-In message always come from hot switches. The new mechanism proposed in this paper to handle the Table-Miss event is named Flow Table Sharing (FTS). The evaluations have demonstrated that FTS reduces both control messages quantity and RTT time by two orders of magnitude compared to current state-of-the-art OpenFlow Table-Miss handler. Even during the flow table overflow period, denial of service for new flows does not happen.

This paper presents a new architecture named FTS to overcome the big performance disaster caused by the flow table overflow in SDN switches. It has several advantages: (1) FTS reduces both control message quantity and RTT time by two orders of magnitude compared to current state-of-the-art OpenFlow Table-Miss handler. (2) We show the validity and fastness of the external user switch-computed select algorithm. And (3) It is Easy to implement, easy to control and the current state-of-the-art OpenFlow Table-Miss handler is a special case of FTS.

This paper is mainly supported by projects of the National Nature Science Foundation of China (61272459), the National High Technology Research and Development Program (863 Program) of China (2013AA013501), open project of the Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory (ITD-U15004/KX152600013), the Ministry of Education New Century Talents Support Program (NCET-13-0450).