应用于协议无感知转发交换机的流缓存方法

曹作伟[①②] 陈 晓*[①] 倪 宏[①] 叶晓舟[①]

[①](中国科学院声学研究所国家网络新媒体工程技术研究中心 北京 100190)

[②](中国科学院大学 北京 100049)

摘 要：协议无感知转发支持任意协议的解析和处理，增强了软件定义网络的可编程能力。为提高转发性能，该文提出一种应用于协议无感知转发交换机的流缓存方法，通过识别匹配和动作的依赖关系，得到匹配字段的绝对位置，用以预先解析报文。为确保流缓存的加速效果，根据匹配类型与表项数量选择应用流缓存的流表。此外，该文对比了单流表缓存与多流表缓存对转发性能的提升，并提出了根据网络流量实际情况的自适应切换策略。通过扩展POFSwitch实现所提方法，并用实际规则与骨干网流量进行验证，应用流缓存后，交换机报文转发速率提升了220%。流缓存可以为可编程数据平面提供更高的转发性能。

关键词：软件定义网络；协议无感知转发；可编程数据平面；流缓存

中图分类号：TP393 **文献标识码**：A **文章编号**：1009-5896(2018)11-2772-07

DOI: 10.11999/JEIT180042

Flow Caching in Protocol Oblivious Forwarding Switches

CAO Zuowei[①②] CHEN Xiao[①] NI Hong[①] YE Xiaozhou[①]

[①](*National Network New Media Engineering Research Center*, *Institute of Acoustics*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*)

[②](*University of Chinese Academy of Sciences*, *Beijing* 100049, *China*)

Abstract: Protocol Oblivious Forwarding (POF) supports the arbitrary protocol processing, enhancing the programmability of Software Defined Networking (SDN). In order to improve the forwarding performance, a flow caching method is proposed. To parse the packet in advance, absolute positions of matching fields are obtained by identifying the dependency of matching and actions. To guarantee the acceleration effect of flow caching, flow tables are selected according to their matching types and number of entries. In addition, the single-flow table cache and multi-flow table cache are compared and an adaptive switching strategy is proposed based on the actual situation of network traffic. The POFSwitch is extended to implement the proposed method and it is validated under the real rules and backbone traces. The switch packet forwarding rate is increased by 220% after applying flow caching. Flow caching can provide higher forwarding performance for programmable data planes.

Key words: Software Defined Networking (SDN); Protocol Oblivious Forwarding (POF); Programmable data plane; Flow caching

1 引言

软件定义网络(Software Defined Networking, SDN)通过解耦控制与转发功能为网络提供可编程能力，使得网络资源与计算、存储资源一样可以自动感知和管理。OpenFlow[1]使用匹配动作表(Match Action Table, MAT)描述交换机转发行为，是最为流行的一种SDN实现。然而OpenFlow面临协议扩展问题，难以满足更多的网络协议、带状态的数据转发等复杂需求。协议无感知转发(Protocol Oblivious Forwarding, POF)[2]拓展了OpenFlow，通过{类型，偏移，长度}的字段描述方式与扩展的指令动作，实现了对任意协议字段的支持。借助POF，交换机无需理解特定的报文格式，新的网络应用的开发不再需要修改交换机代码，新技术、新业务的迭代速度将得到极大的提升[3]。

在MAT数据路径中，报文处理包括字段解析、多次流表匹配与动作执行，受表项数量与匹配字段的限制，流表匹配成为了交换机转发性能的瓶颈[4]。为提高流表匹配性能，Yingchareonthaworn-

收稿日期：2018-01-11；改回日期：2018-06-14；网络出版：2018-06-30

*通信作者： 陈晓 xxchen@dsp.ac.cn

基金项目：国家科技重大专项项目(2017ZX03001019)

Foundation Item: The National Science and Technology of Major Projects of China (2017ZX03001019)

chai等人[5]提出PartitionSort匹配算法，通过最大独立集的分组方式划分规则，避免规则重叠对流表更新性能的影响，却牺牲了流表匹配性能。Katta等人[6]提出CacheFlow方法，通过缓存流行规则，缓解TCAMs (Ternary Content Addressable Memories)容量受限的问题，然而TCAMs更新慢且复杂，缓存缺失的开销会非常高。作为最流行的OpenFlow交换机，Open VSwitch (OVS)[7]使用流缓存避免了上述问题，通过记录流的首个报文的匹配结果，用一次精确匹配代替多次流表匹配，加速后续报文的处理。网络流量通常都具有很强的时间局部性[8]，同一条流的报文常常会一起出现，通过流缓存可以显著提升转发性能。OVS衍生的交换机大多都沿用了流缓存设计，如SoftFlow[9]和PISCES[10]。

网络协议数据面一般不会在包头域进行算术操作，只是做一些TTL和CRC的计算，而POF流表混合了解析与转发功能，可以处理解析过程中的状态变化，这给应用流缓存带来了困难。PVS[11]对每个流表单独应用流缓存，规避了这个问题，但也导致了匹配次数增加、空间利用率降低等问题。

针对POF协议的限制，本文提出一种应用于POF交换机的流缓存方法，通过识别匹配和动作的依赖关系，得到匹配字段的绝对位置，用以预先解析报文。为确保流缓存的加速效果，根据匹配类型与表项数量选择应用流缓存的流表。此外，本文对比了单流表缓存与多流表缓存对转发性能的提升，并提出了根据网络流量实际情况的自适应切换策略。通过扩展POFSwitch[12]实现所提方法，并用实际规则与流量数据进行验证，应用流缓存后，交换机报文转发速率提升了220%。

2　POF流缓存

2.1　POF交换机的报文处理流程

POF基于MAT模型，报文进入交换机后首先匹配第1个流表，匹配成功则执行报文处理动作，并根据需要将报文数据(通过Goto-Table动作)交给下一个流表处理。POF使用｛类型，偏移，长度｝描述协议字段，“类型”包括报文数据和元数据，对于报文数据字段，“偏移”为相对于协议首部的偏移。报文在POF流表中是逐层处理的，报文偏移(packet offset)用于指示当前协议首部的位置。每一层协议由1个或多个流表处理，通过移动报文偏移，实现字段解析功能。如图1所示，报文包含3层协议首部，分别由3个流表处理，Move-Packet-Offset动作移动报文偏移指向下一层协议首部。若需要同时处理多层协议，使用元数据传递协议字段。

POF与OpenFlow在报文处理上主要有两点不同，第一，POF允许动作使用任意字段修改报文数据，而OpenFlow规定了数据路径一致性(pipeline consistency)，动作使用和修改的字段只能是匹配字段；第二，POF通过字段描述和报文偏移实现字段解析，不需要专门的解析器(parser), OpenFlow的匹配字段基于既有经验定义，由专门的解析器提取报文字段。

2.2　POF流缓存的匹配字段

应用流缓存需要满足两个条件，其一，流缓存匹配的字段能够识别流；其二，这些字段能够预先提取。流的定义依赖于具体的应用，在MAT模型下，称报文匹配的(流表，表项)序列为匹配路径，匹配路径一致的报文，属于同一条流。

OpenFlow可以预先解析匹配字段，并且通过数据路径一致性，保证了预解析的字段能够确定报文匹配路径。基于此，OVS使用预解析的匹配字段识别流，匹配字段相同的报文，其匹配路径也相同。而匹配字段不同的报文，其匹配路径也可能相同，称这种方式识别的“流”为小流(microflow)。

POF的字段解析混合在流表中，无法预先提取

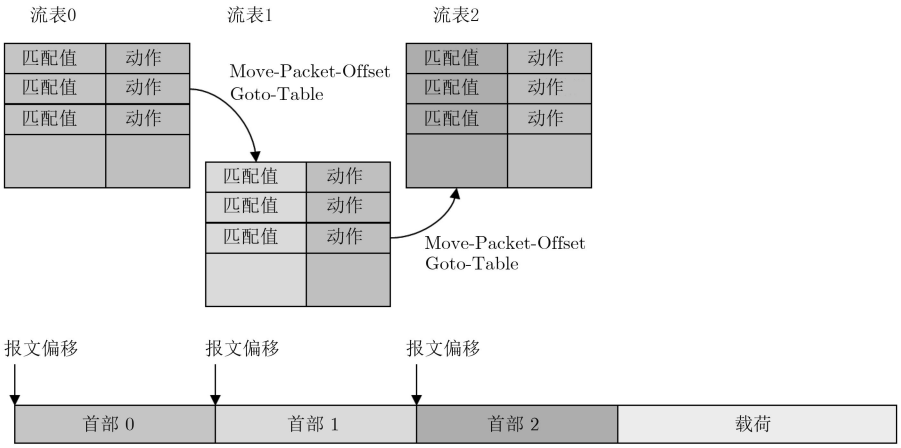


图 1 POF流表的报文处理流程

匹配字段，而且POF并未要求数据路径一致性，仅依靠匹配字段不能完全确定匹配路径。在POF流表中，匹配字段与动作之间存在着依赖关系，如图2所示。若流表T的匹配字段值依赖于动作a的执行，比如动作a修改报文偏移，改变了匹配字段位置，则流表T与动作a存在依赖关系。若动作a依赖于动作b的执行结果，比如动作b修改了动作a的输入字段，则动作a与动作b存在依赖关系。通过遍历所有可能的匹配路径，识别其中的依赖关系，搜集影响流表匹配的字段，这些字段可以预解析，也能识别流。本文即采用这些字段用于流缓存匹配，所识别的“流”也是小流，除非特指，后文所指的流均为小流。

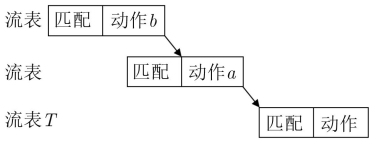


图 2 匹配字段与动作间的依赖关系

2.3 POF流缓存数据路径

根据上一节的分析，通过识别依赖关系，搜集影响流表匹配的字段可实现流缓存匹配。POF中存在部分流表，表项数量少、匹配速度快，应用流缓存不一定能得到加速效果。选择适宜流缓存的流表，得到图3所示的流缓存数据路径。当报文进入流缓存时，提取报文字段用于流缓存匹配，若匹配成功，则执行所属流的动作，否则进行流表匹配，记录其匹配路径，当报文离开流缓存时，作为新流缓存下来。

以流表为节点，表项为弧，合并重弧(multiple arcs)，得到弱连通的有向无环图G。G中有且仅有1个节点的入度为0，其他节点入度大于或等于1。选择应用流缓存的流表，得到多个弱连通的子图$\{G_i\}$。如图4所示，选择流表T_1, T_2, T_3, T_4应用流缓存，构成子图$G_1=\{T_1,\ T_3,\ T_4\}$与$G_2=\{T_2\}$，设T_1, T_2, T_3, T_4的匹配字段集合分别为F_1, F_2, F_3, F_4。以入口流表为单位，对G_1与G_2分别应用流缓存。对于G_1，通过识别匹配路径中的依赖关系，得到T_1处决定匹配T_3时F_3值的字段集合F_3'，与T_1处决定匹配T_4时F_4值的字段集合F_4'，合并F_1, F_3', F_4'得到F。当报文进入T_1时，使用F预解析字段，用于流缓存匹配。对于G_2，则对T_2单独应用流缓存。称G_2这种单个流表应用流缓存的方式为单流表缓存，G_1这种多个流表组合应用流缓存的方式为多流表缓存。

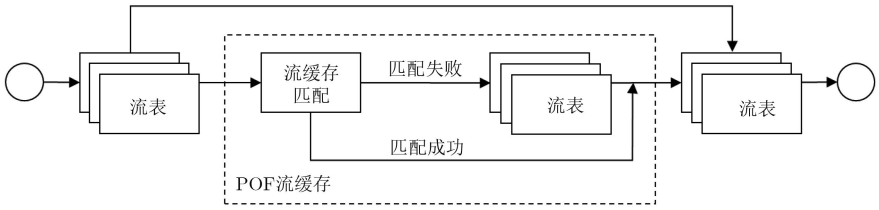


图 3 POF流缓存数据路径

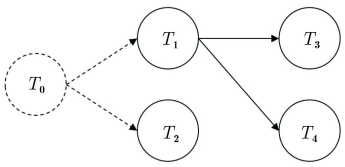


图 4 应用流缓存的流表

当动作使用字段值修改报文偏移时，无法预知其具体值，只能得到F_3'的可能位置，若范围过大，甚至包含整个报文数据，此时，宜对T_3使用单流表缓存。

3 依赖关系的识别

字段与动作的依赖关系反映了字段解析过程与报文分类过程。识别流表中的依赖关系，是POF应用流缓存的必要前提。

按照POF动作对字段的影响，将其分为5类：

(1) 动作会修改字段值，不改变字段定义，如Set-Field, Algorithm等。

(2) 动作会移动报文偏移，改变报文数据字段的位置，如Goto-Table, Move-Packet-Offset。

(3) Add-Field与Delete-Field，会添加或删除字段，同时改变字段位置。

(4) 动作不会对字段造成影响，如Output, Drop等。

(5) Apply-Actions与Group，为其他动作的集合。

给定字段集合F，与动作存在依赖关系的条件是：对于(1)类动作，输出字段与F中的字段重叠；对于(2)类动作，F中存在报文数据字段；对于(3)类动作，F中存在字段位于添加或删除的位置之后；对于(4)类动作，集合中存在影响F的动作。

依赖关系贯穿于整个匹配路径，为搜集影响匹

配路径的字段，需记录当前动作对字段的影响。围绕字段，按照解析路径的逆序识别依赖关系，使用相对偏移判断字段之间的位置关系。用$g(F)$表示经过动作影响后的字段集合，对于(1)类动作，输出字段不会被更早的动作影响，应删除F中与输出字段重叠的部分，$g(F)$应包含动作的输入字段；对于(2)类动作，按照移动报文偏移的反方向修改F中报文数据字段的相对偏移，比如字段f的偏移值为o，若动作前移报文偏移d bit，则$g(f)$的偏移值为$o-d$，若移动的偏移值来自字段，则用范围表示$g(f)$的偏移值，若动作前移报文偏移，则$g(f)$的偏移值变为$(-\infty, o]$；对于(3)类动作，若动作为Add-Field，则应删除F中与添加字段重叠的部分，同时减小添加位置之后的字段偏移，若动作为Delete-Field，则应增加F中包含删除位置的字段的长度，同时增加删除位置之后的字段偏移。若动作对字段无影响，则$g(F)=\varnothing$。

表1描述了依赖关系识别算法，表项e包含多个动作，采用逆序与字段F进行比较，若动作影响了字段，则更新字段定义，合并输入字段，判断下一个动作。

表 1　依赖关系识别算法

算法1 依赖关系识别算法Dep(e, F)

(1) for each action a in entry e in reverse order
(2)　if $a \cdot g(F) \neq \varnothing$
(3)　　$F = a \cdot g(F)$
(4)　　$F = F + \{\text{input fields of } a\}$
(5) return F

4　流表的选择

并非所有POF流表都适用于流缓存。令流缓存命中率为ρ，流表的表项数量为N，流表匹配算法的时间性能为$f(N)$，流缓存匹配采用哈希实现，时间性能为$O(k)$, k为匹配字段的长度。设引入流缓存前的流表匹配的时间性能为$C=f(N)$，则引入流缓存后的时间性能为$C'=(1-\rho)f(N)+O(k)$。当$C'<C$时，流表可以通过流缓存获得加速效果。不同的匹配类型，通常采用不同的匹配算法，表2给出了几种匹配类型的典型匹配算法。

表 2　几种典型的匹配算法

匹配类型	LM	EM	LPM	MM
匹配算法	线性	哈希	PartitionSort	类S-Trie
$f(n)$	$O(1)$	$O(1)$	$O(p\cdot(d+\lg n))$	$O(d\cdot\lg n)$

对于LM和EM，由于$f(N)=O(k)$，故而$C'\geq C$，应用流缓存后得不到加速效果。对于LPM和MM，取$f(n)=\lg n$，计算$C'<C$，得到

$$N > 2^{k/\rho} \tag{1}$$

取$\text{Thr}=2^{k/\rho}$，当表项数量$N>\text{Thr}$时，LPM和MM流表可以通过流缓存获得加速效果，其中缓存命中率ρ可以根据经验得到。

5　多流表缓存与单流表缓存

通过式(1)选择的流表，既可以应用多流表缓存，也可以应用单流表缓存。为比较二者的适用情况，定义流表应用流缓存的收益为$w=C-C'=\rho\cdot f(N)-O(k)$，选择收益高的流缓存方式。流表的匹配字段构成报文空间$U=\{0,1\}^k$，网络中的报文符合齐夫分布，即$P(X=i)=\dfrac{1}{\sum_{j=1}^{K} j^{-\alpha}}\, i^{-\alpha}$。其中$K$为报文的种类，$K\gg 1$。参数$\alpha>0$, α越小，则报文分布越均匀。设不同报文空间上，报文的参数α相同。当流缓存空间有限，而流量无限时，采用最近最常使用替换算法，可保留前n个概率最高的流，$n\leq K$。这n条流所占比例为$P(X\leq n)=\sum_{i=1}^{n} i^{-\alpha}\approx\int_1^n x^{-\alpha}\mathrm{d}x$。得到缓存命中率

$$\rho(n)=\frac{P(X\leq n)}{P(X\leq K)}=\begin{cases}\left(\dfrac{n}{K}\right)^{1-\alpha}, & 0<\alpha<1\\ \log_K(n), & \alpha=1\\ 1-n^{1-\alpha}, & \alpha>1\end{cases} \tag{2}$$

$\rho(n)$为单调递增的凹函数。

设G为应用流缓存的流表构成的一个弱连通子图，包含T_1, T_2, …, T_l共l个流表，其中T_1为其他流表的父流表，流表的匹配字段集合为F_1, F_2, …, F_l，长度为k_1, k_2, …, k_l，对应空间为U_1, U_2, …, U_l，报文种类为K_1, K_2, …, K_l，流缓存空间大小为$|S|$。单流表缓存中，每个流表占用一部分缓存空间，包含的流的数量为n_1, n_2, …, $n_l\geq 0$，有$\sum k_i n_i=|S|$。单流表缓存的整体收益为

$$W_s(\boldsymbol{n})=\sum_i\left(\rho_i(n_i)\cdot f(N_i)-O(k_i)\right) \tag{3}$$

多流表缓存中，匹配字段为F，长度为k，对应报文空间U，含有K条流，有$k\geq k_1$。所有流表共用流缓存空间，缓存命中率相同为ρ，且$kn=|S|$，整体收益为

$$W_m=\rho(n)\cdot\sum_i f(N_i)-O(k) \tag{4}$$

分配缓存空间使单流表缓存整体收益最大，等

效为最优化问题。

$$\left.\begin{aligned}&\min \quad -W_s(\boldsymbol{n})\\&\text{s.t.} \quad n_i\geq 0,\ i=1,2,\cdots,l\\&\qquad \sum k_i n_i-|S|=0\end{aligned}\right\} \tag{5}$$

易知$-W_s(\boldsymbol{n})$为1阶导数连续的凸函数，约束函数1阶导数连续，可行域为超平面与半空间的交集，为凸集。问题式(5)的局部最优解$\boldsymbol{n}^*$即为全局最优解[13]，应满足式(6)：

$$\left.\begin{aligned}&\nabla_n \mathcal{J}(\boldsymbol{n}^*,\lambda)=0\\&c_0(\boldsymbol{n}^*)=0\\&c_i(\boldsymbol{n}^*)\geq 0, i=1,2,\cdots,l\\&\lambda_i\geq 0, i=1,2,\cdots,l\\&\lambda_i c_i(\boldsymbol{n}^*)=0, i=0,1,2,\cdots,l\end{aligned}\right\} \tag{6}$$

其中，$\mathcal{J}(\boldsymbol{n}^*,\lambda)=-W_s(\boldsymbol{n}^*)-\sum_{i=0}^{l}\lambda_i c_i(\boldsymbol{n}^*)$为拉格朗日函数，$c_0(\boldsymbol{n}^*)=k_i n_i-|S|$，$c_i(\boldsymbol{n}^*)=n_i$。

以$0<\alpha<1$为例。求解式(6)，可得到$n_i=\left(\dfrac{(1-\alpha)f(N_i)}{\lambda_0 k_i K_i^{1-\alpha}}\right)^{\frac{1}{\alpha}}$，其中$\lambda_0=\dfrac{|S|}{\sum_{i=1}^{l}\left(\dfrac{(1-\alpha)f(N_i)}{(k_i K_i)^{1-\alpha}}\right)^{\frac{1}{\alpha}}}$，故单流表缓存的整体收益为

$$W_s(\boldsymbol{n}^*)=\sum_i\left(\frac{(1-\alpha)}{\lambda_0 k_i K_i}\right)^{\frac{1-\alpha}{\alpha}}\cdot f(N_i)^{\frac{1}{\alpha}}-\sum_i O(k_i) \tag{7}$$

而多流表缓存的整体收益为

$$W_m=\left(\frac{|S|}{kK}\right)^{1-\alpha}\cdot\sum_i f(N_i)-O(k) \tag{8}$$

计算式(7)与式(8)，选择整体收益最大的缓存方法。在实际环境中，由于无法事先获得不同流表的报文种类K_i，无法直接比较。此时可优先采用多流表缓存，利用数据流算法[14]记录每个流表的流数量，以及多流表缓存的流数量。当$W_s(\boldsymbol{n}^*)>W_m$时，退化为单流表缓存。

基于前述分析，得到选择流缓存流表的算法，如表3所示，采用深度优先搜索遍历所有可能的匹配路径。为减少重复搜索，第2行去除所有(3)类动作，合并动作相同的表项。第3～5行判断当前流表是否应用流缓存。第11～12行识别表项与子流表之间的依赖关系，用$F'(Y)$记录决定子流表匹配字段值的字段集合，$F(Y)$表示流表Y处决定后续解析路径的字段集合。第13～16行，在遍历子孙流表后，用$F(X)$记录合并的字段，用于后续的依赖关系判断。

表 3 流表选择算法

算法2 流表选择算法Select(X)

(1) label flow table X as discovered
(2) delete type (3) actions and merge duplicate entries
(3) if X is LPM or MM
(4) if entry number of X> Thr
(5) label X as worth caching
(6) for each entry e in X
(7) let Y as the table which e leads
(8) if Y is not discovered
(9) let $F'(Y)=\Phi$
(10) recursively call Select(Y)
(11) if X is worth caching and Y is worth caching
(12) $F'(Y)=F'(Y)+\mathrm{Dep}(e,F(Y))$
(13) if X is worth caching
(14) let $F(X)$ =match fields of X
(15) for each child table Y worth caching
(16) $F(X)=F(X)+F'(Y)$

6 验证与分析

6.1 实验环境

为了评估流缓存的可行性与算法的有效性，本文基于POFSwitch实现了所提的POF流缓存方法，应用随机替换策略。POFSwitch分别采用二叉树和线性查找实现LPM和MM，使用单线程处理报文。POFSwitch运行的硬件环境为Linux 2.6.32 64位系统，Intel Xeon E5-2620 CPU 2.00 GHz, 16 GB DDR3内存。使用PCTRL[15]作为控制器，Ixia PerfectStorm ONE作为流量发生器，环境拓扑如图5所示。测试用的流量数据来自CAIDA[16]，包含6.56M个报文，其中五元组的数量为361.3 k, IP地址对的数量为357.2 k。测试用的流表包含L2转发、L3转发和ACL表，如图6所示。其中，L3转发规则来自RouterViewer[17], ACL规则使用ClassBench[18]生成。为便于测试交换机转发性能，为L3表和ACL表添加了默认规则，使所有流量都能经过交换机转发回流量发生器。

6.2 交换机转发性能测试

本文使用测试仪器以500 kpps速率循环发送流量数据，测试了不同流缓存空间下，应用流缓存前

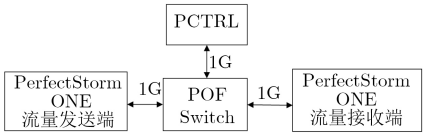


图 5 环境拓扑

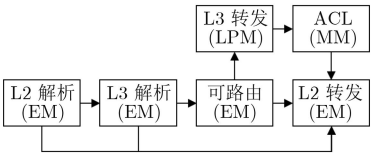


图 6 流表配置

后的交换机转发性能，结果如图7所示。流表选择算法选择了L3转发表与ACL表组成多流表缓存。流缓存空间从0.5 k条流增加到512 k条流，每轮测试执行30 s，取5次测试的平均值。未应用流缓存时，交换机的转发性能为13.2 kpps。应用流表缓存后，0.5 k流缓存空间下的转发性能提升到15.5 kpps，而256 k流缓存空间下的转发性能提升到42.3 kpps，较应用流缓存前提升了220%。简而言之，应用流缓存后，交换机的转发性能得到显著提升，并随流缓存空间增大而升高。

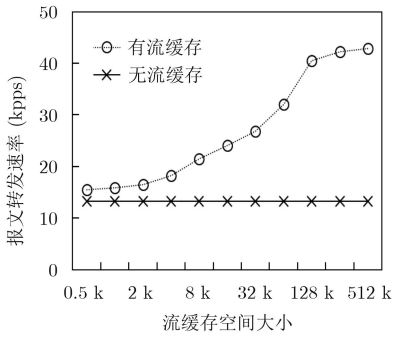


图 7 应用流缓存前后的报文转发速率对比

当流缓存空间增大到128 k后，继续增大流缓存空间，转发性能的增长幅度减缓。原因在于，128 k大小的流缓存空间，已经能缓存大部分流量，继续增大缓存空间，缓存命中率的提升有限。当流缓存空间小于256 k时，缓存空间的占用率接近100%，而256 k流缓存的空间占用率只有54.98%，512 k流缓存的空间占用率只有28.22%。据此计算，流的数量大约为150 k左右，少于五元组的数量(361.3 k)。这是由于测试仪器的发送速率远高于交换机的转发速率，Linux系统会随机丢弃报文，导致被交换机处理的流的数量减少。

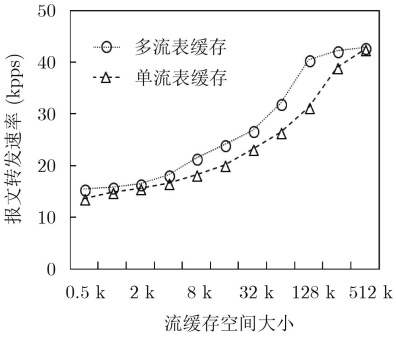
6.3 多流表缓存与单流表缓存比较

本文还比较了单流表缓存与多流表缓存的转发性能。单流表缓存中，ACL表与L3表单独应用流缓存，缓存空间按流数量等比例分配。从图8可以看出，在缓存空间增加到512 k前，单流表缓存ACL表的缓存命中率低于多流表缓存，虽然L3表的命中率高于多流表缓存，但交换机转发性能主要受到ACL表匹配性能的限制。这是因为POFSwitch中，MM算法实现较LPM算法实现要慢许多。当流缓存空间增加到256 k后，继续增加流缓存空间难以继续提升多流表缓存的缓存命中率。当流缓存空间增加到512 k时，单流表缓存ACL表的缓存空间达到256 k，其缓存命中率逼近多流表缓存，二者转发速率接近。总而言之，在本实验中，等比例分配缓存空间的单流表缓存的转发性能要低于多流表缓存。

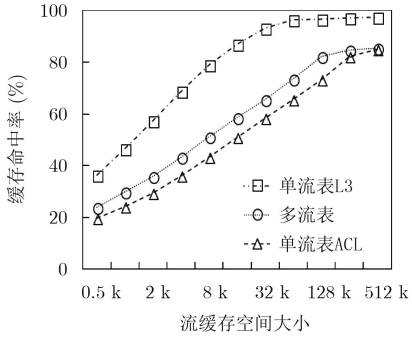
7 结束语

此前的研究验证了流缓存对于提高转发性能的有效性。然而，这些研究没有解决POF协议应用流缓存的限制，在POF交换机中无法实现多流表的匹配加速。针对这个问题，本文提出了一种应用于POF交换机的流缓存方法，通过识别流表中的依赖关系，使POF交换机能够支持多流表的匹配加速。在此基础上，本文对比了单流表缓存与多流表缓存对转发性能的提升，并提出了根据网络流量实际情况的自适应切换策略。本文基于POFSwitch与实际流量规则，对所提方法进行了验证。实验结果表明，所提流缓存方法能够提高POF交换机的转发性能。本文的研究表明，流缓存可以为可编程数据平面的提供更高的转发性能。

未来工作将进一步研究POF流缓存支持表项更



(a) 报文转发速率



(b) 缓存命中率

图 8 多流表缓存与单流表缓存的转发速率与命中率对比

新的方法，以及当流缓存的流表为非连通图时，流缓存空间的分配策略。

参 考 文 献

[1] 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. doi: 10.1145/1355734.1355746.

[2] SONG Haoyu. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane[C]. Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, New York, USA, 2013: 127–132.

[3] 李平, 王雷. 融合SDN的信息中心网络研究综述[J]. 网络新媒体技术, 2017, 6(6): 11–18.
LI Ping and WANG Lei. A survey of information-centric networking fusing SDN[J]. *Journal of Network New Media*, 2017, 6(6): 11–18.

[4] RYGIELSKI P, SELIUCHENKO M, KOUNEV S, *et al*. Performance analysis of SDN switches with hardware and software flow tables[C]. Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools, Brussels, Belgium, 2017: 80–87.

[5] YINGCHAREONTHAWORNCHAI S, DALY J, LIU A X, *et al*. A sorted partitioning approach to high-speed and fast-update OpenFlow classification[C]. 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 2016: 1–10.

[6] KATTA N, ALIPOURFARD O, REXFORD J, *et al*. CacheFlow: Dependency-aware rule-caching for software-defined networks[C]. Proceedings of the Symposium on SDN Research, New York, USA, 2016: 6: 1–6: 12.

[7] PFAFF B, PETTIT J, KOPONEN T, *et al*. The design and implementation of Open vSwitch[C]. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Oakland, USA, 2015: 117–130.

[8] CLAFFY K C. Internet traffic characterization[D]. [Ph.D. dissertation], University of California, 1994: 90–91.

[9] JACKSON E J, WALLS M, PANDA A, *et al*. Softflow: A middlebox architecture for Open vSwitch[C]. 2016 Usenix Annual Technical Conference (USENIX ATC 16), Santa Clara, USA, 2016: 15–28.

[10] SHAHBAZ M, CHOI S, PFAFF B, *et al*. PISCES: A programmable, protocol-independent software switch[C]. Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, New York, USA, 2016: 525–538.

[11] SUN Quanying, XUE Yuhan, LI Shengru, *et al*. Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks[J]. *IEEE Access*, 2017, 5: 24004–24011. doi: 10.1109/ACCESS.2017.2767640.

[12] USTC-INFINITELAB. POFSwitch: The original protocol-oblivious forwarding (POF) switch by Huawei[OL]. https://github.com/USTC-INFINITELAB/POFSwitch, 2016.

[13] FLETCHER R. Practical Methods of Optimization[M]. John Wiley & Sons, 2000: 213–219.

[14] BAR-YOSSEF Z, JAYRAM T, KUMAR R, *et al*. Counting distinct elements in a data stream[C]. International Workshop on Randomization and Approximation Techniques in Computer Science, Cambridge, USA, 2002: 1–10.

[15] USTC-INFINITELAB. PCTRL: A POF controller extending from POX[OL]. https://github.com/USTC-INFINITELAB/PCTRL, 2017.

[16] CAIDA. The CAIDA anonymized Internet traces 2016 dataset[OL]. http://www.caida.org/data/passive/passive_2016_dataset.xml, 2018.

[17] UNIVERSITY OF OREGON. Route Views Project[OL]. http://www.routeviews.org/chicago.html, 2018.

[18] TAYLOR D E and TURNER J S. Classbench: A packet classification benchmark[J]. *IEEE/ACM Transactions on Networking (TON)*, 2007, 15(3): 499–511. doi: 10.1109/TNET.2007.893156.

曹作伟：男，1991年生，博士生，研究方向为协议无感知转发、可编程数据平面等.

陈 晓：男，1964年生，研究员，研究方向为网络通信、新媒体技术等.

倪 宏：男，1964年生，研究员，研究方向为网络通信、新媒体技术等.

叶晓舟：男，1980年生，研究员，研究方向为嵌入式多核网络处理器、新媒体技术等.