doi:10.11959/j.issn.1000-436x.2016204

多单元散列表与 TCAM 结合的 OpenFlow 流表查找方法

李春强[1]，董永强[1,2]，吴国新[1,2]
（1.东南大学计算机科学与工程学院，江苏 南京 211189；2.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：在 OpenFlow 网络中，交换机通过标准化的接口接受基于流的规则，执行基于流的报文处理。流表的查找是 OpenFlow 交换机的核心功能，TCAM 以其优异的性能广泛用于 OpenFlow 流表的查找，然而基于 TCAM 的 OpenFlow 流表查找具有较高的成本与能耗。为了降低流表查找的成本与能耗，提出了多单元散列表与 TCAM 结合的 OpenFlow 流表存储与查找的方法。通过理论分析与仿真测试，给出了查找结构成本优化后的散列表、TCAM 的容量配置；在该配置下，Hash-TCAM 流表查找结构比单纯使用 TCAM 的方案节约 90%以上的成本，有效降低了能耗，同时保持了相近的查找性能。

关键词：OpenFlow；三态内容寻址存储器；散列表；流表
中图分类号：TP393 **文献标识码**：A

OpenFlow table lookup scheme integrating multiple-cell Hash table with TCAM

LI Chun-qiang[1], DONG Yong-qiang[1,2], WU Guo-xin[1,2]
(1.School of Computer Science and Engineering, Southeast University, Nanjing 211189, China;
2.Key Laboratory of Computer Network and Information Integration, Ministry of Education, Southeast University, Nanjing 211189, China)

Abstract: In OpenFlow networks, switches accept flow rules through standardized interfaces, and perform flow-based packet processing. To facilitate the lookup of flow tables, TCAM has been widely used in OpenFlow switches. However, TCAM is expensive and consumes a large amount of power. A hybrid lookup scheme integrating multiple-cell Hash table with TCAM was proposed for flow table matching to simultaneously reduce the cost and power consumption of lookup structure without sacrificing the lookup performance. By theoretical analysis and extensive experiments, optimal capacity configuration of Hash table and TCAM was achieved with the optimized cost of flow table lookup. The experiment results also show that the proposed lookup scheme can save over 90% cost and the power consumption of flow table matching can be reduced significantly compared with the pure TCAM scheme while keeping the similar lookup performance.

Key words: OpenFlow, ternary content addressable memory, Hash table, flow table

1 引言

为了在已有的网络基础设施中构建网络创新研究的实验环境，软件定义网络（SDN, software defined networking）作为一种新型的网络体系结构被提出，OpenFlow[1]技术作为实现 SDN 的一种具体方案，受到了学术界和工业界的普遍关注和广泛研究。OpenFlow 系统实现了数据转发和控制功能的分离，通过独立的控制器来控制网络设备(称为 OpenFlow 交换机)的转发行为；OpenFlow 控制器在逻辑上采用集中式的控制方式，OpenFlow 交换机通过标准化的接口接收来自控制器的报文处理规

收稿日期：2016-01-25；**修回日期**：2016-08-30
通信作者：吴国新，gwu@seu.edu.cn
基金项目：国家高技术研究发展计划(“863”计划)基金资助项目(No.2013AA013503)；国家自然科学基金资助项目(No.61272532, No.61370209)；江苏省未来网络前瞻性研究基金资助项目(No.BY2013095-2-06)
Foundation Items: The National High Technology Research and Development Program of China (863 Program) (No.2013AA013503), The National Natural Science Foundation of China (No.61272532, No.61370209), The Future Networks Prospective Research Program of Jiangsu Province (No.BY2013095-2-06)

则，执行基于流的报文处理。OpenFlow 机制以其良好的可编程、可控制等特性，已被应用于数据中心、企业网、校园网中网络流量的管理与控制。

流表是 OpenFlow 机制的核心，它是流表项的集合，流表项的基本格式如图 1 所示，主要有匹配字段（Match Fields）、计数器（Counters）和操作（Instructions）等部分组成，其中，匹配字段可以包含多个匹配项，涵盖了链路层、网络层和传输层等层次的报文头部字段，计数器根据匹配到的报文进行统计，操作则指明匹配到该流表项的报文应该执行的操作，比如转发、丢弃、修改等。

Match Fields	Counters	Instructions

图 1　OpenFlow 流表项格式

随着 OpenFlow 规范的不断更新，流表项匹配字段从最初的十元组[1]（如图 2 所示），逐步增加了 VLAN 优先级等字段，后续 MPLS 和 IPv6 等协议字段也逐渐扩展到 OpenFlow 标准[2~4]中，流表的匹配字段越来越长，流表的规模越来越大。OpenFlow 交换机收到数据报文后，提取报文的头部信息查询 OpenFlow 流表，根据流表中匹配到的规则对报文执行相应的操作；为了实现高速的流表匹配，OpenFlow 交换机通常采用三重内容可寻址存储器(TCAM, ternary content addressable memory)存储与查找流表[1,2,5]，这种不断扩展的数据转发面抽象，需要占用大量的 TCAM 资源，大大增加了硬件成本。

TCAM 可以完全并行地查找整个数据字段的集合，具有优异的查找性能。然而 TCAM 这种高性能并行查找机制带来了高的能耗与散热问题；同时 TCAM 的成本也非常高，TCAM 单比特成本大约相当于静态随机存储器(SRAM, static random access memory)的 30 倍[6]；TCAM 能耗也明显超出 SRAM 数倍[6,7]，并且会随并行匹配字段的条目数及宽度而增加[7]。设备的成本与性能决定了一项技术能否被推广与普及，因此，降低 OpenFlow 交换机的成本与能耗是目前亟待解决的关键问题。OpenFlow 流表的查找是其核心功能，实现高性能、低成本、低能耗的流表查找成为 OpenFlow 交换机实现的基本要求。

针对基于 TCAM 的 OpenFlow 流表查找方案具有较高能耗与成本的问题，提出了将多单元散列表与 TCAM 结合的 OpenFlow 流表存储与查找的方法。本文的主要贡献包括以下几点。

1) 提出将多单元散列表与 TCAM 结合的 OpenFlow 流表存储与查找的方法，使用连续的存储空间存储位于同一散列桶中的多个查找单元，其中每个查找单元包含一个流表表项的索引信息。

2) 分析了散列表的冲突溢出率，根据散列冲突溢出的变化率，展示了多单元散列表—TCAM 查找结构以及二级多单元散列表—TCAM 具有良好的成本优势。

3) 在散列表中，通过流表表项关键字指纹代替关键字的匹配减少单次存储器读操作的数据位数，并进一步降低了散列表的成本。

4) 通过理论分析有效配置 TCAM 与散列表的容量，优化了 OpenFlow 流表的存储与查找的成本及能耗，并通过仿真测试进行了验证。

2　相关技术介绍

2.1　降低流表存储开销及能耗的研究

随着 OpenFlow 应用范围的扩大，OpenFlow 流表的规模越来越大，匹配字段的位数越来越多，从 OpenFlow1.1 版[2]开始提出了通过多级流表和流水线查找结构来压缩流表存储空间，但其压缩效果与流表表项匹配字段的具体内容密切相关。当级数较多时会显著增加报文处理的时延，同时增加了 OpenFlow 交换机硬件设计的复杂度。文献[8~12]提出了基于分解的 OpenFlow 流表匹配方法，按照流表匹配字段在报文头部的协议层次，将 OpenFlow 流表划分成多个子表；进行规则匹配时，报文头被分割成多个字段，每个字段独立地执行查找操作；根据每个查找操作的返回结果合并后，再执行一次查找获得匹配的流表规则。该类方案的主要问题在于流表一个条目的更新可能会涉及子表中的多个条目，更新过程较为复杂。

文献[13,14]提出了通过压缩流表减少 TCAM

字段	入端口	VLAN ID	以太源地址	以太目的地址	以太帧类型	IP源地址	IP目的地址	协议	传输层源端口	传输层目的端口
位数/bit	由实现决定	12	48	48	16	32	32	8	16	16

图 2　十元组格式的流表项匹配字段

使用的方案，采用启发式方法，求解语义等同的流表集合，结果导致压缩的效果与流表的内容密切相关，压缩比不确定，而且无法支持实时更新流表的条目。文献[15,16]提出了基于决策树的 OpenFlow 流表匹配方案，同样采用启发式算法构建匹配规则的决策树，需要多次查表才能确定匹配的流表条目；匹配字段位数越多，查找的时延越高，查找的性能受到限制。

为了降低 OpenFlow 流表匹配的能耗，文献[5]根据数据报文的局部性，通过增加报文预测电路，通过缓存某段时间内频繁访问的流表条目来尝试减少与 TCAM 中流表进行匹配的操作。该方案虽然考虑了降低 OpenFlow 交换机功耗与时延的问题，但并未降低基于 TCAM 的 OpenFlow 流表存储与查找的成本。

2.2 基于散列表的存储与查找

散列表具有良好的平均查找性能且易于实现，因此将散列表应用到网络报文处理上得到了广泛的研究[17]。当多个关键字通过散列运算映射到同一个散列桶，称为散列冲突，散列冲突是影响散列查找性能的主要因素，如何有效处理散列冲突是散列表实现高效查找的关键。通常，借助于链表（拉链法）或开放地址探测（开放定址法）等方式解决散列冲突。但无论是拉链法还是开放定址法，当待查找的条目存在散列冲突时，需要多次存储器访问才能获得查找结果，因此不能确保最差情形下的查找性能。

文献[18]提出一种基于开放地址探测的散列冲突处理方案，该方案中使用 2 个子散列表，每个条目插入时在 2 个子表中各有一个候选的位置，但只插入到其中一个。当插入一个新条目时，根据关键字使用不同的散列函数计算在 2 个子表中的位置，只要任意一个位置为空则可以将该条目插入该空位置；当 2 个位置均非空时，选择一个子表的位置将新条目插入，并将被替换的条目插入到另一表中的候选位置，如果另一表中的相应位置非空，继续进行迭代，直到找到一个空闲的位置；如果经过一定的迭代次数仍无法找到空闲位置，则需要执行 rehash 操作或者将被替代的条目放入 TCAM[19]。对于该方案而言，通过并行地查找 2 个子散列表，可以确保查找的性能；然而在插入新条目时，可能需要多次移动表中的条目，插入的性能是不确定的，无法满足流表的实时更新要求。

基于散列表的存储与查找方案虽然可以使用 SRAM、短时延动态随机存储器(RLDRAM, reduced-latency dynamic random access memory)等功耗、成本相对较低的存储器，但随着散列表利用率的增加，散列冲突出现的概率也会增加，从而会导致散列表处理性能下降、查找性能不确定的问题，这对高速网络报文处理是难以容忍的；基于 TCAM 的 OpenFlow 流表存储与查找方案，可以实现高效确保的查找性能，但其成本与功耗相对较高，这会影响 OpenFlow 技术的普及与推广。因此，本文将这 2 种方案进行结合，对以流量管理[20,21]或报文转发[22]为主要功能的 OpenFlow 交换机，降低其流表存储与查找的成本及能耗。

3 Hash-TCAM 混合流表存储与查找方案

图 3(a)给出了常规的 Hash-TCAM 混合流表查找结构。由于使用网络处理器(NP, network processor)[23,24]、专用集成电路(ASIC, application specific integrated circuits)执行报文处理是 OpenFlow 交换机常见的实现方式，如果 NP 或 ASIC 带有一定容量的片内 TCAM[25]，则流表的查找结构如图 3(b)所示。

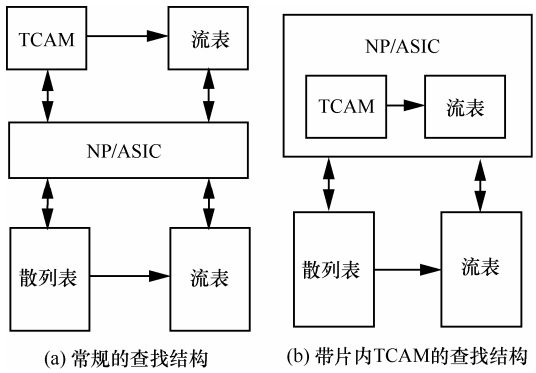


图 3 Hash-TCAM 混合流表查找结构

方案的基本思路是将散列表作为 OpenFlow 流表的精确匹配与存储的主要措施，用 TCAM 处理散列表的冲突溢出。通常，TCAM 每个时钟周期都可以执行一次查找，SRAM 每个时钟周期也可以执行一次读操作，在没有散列冲突溢出的情况下，散列查找只需要执行一次 SRAM 读操作就可以查到对应条目，因此将冲突溢出的条目存储到 TCAM 中，避免因多次 SRAM 读操作导致的查找性能下降，于是每个时钟周期均可以执行一次 OpenFlow 流表的查找操作，确保了流表的查找性能。

3.1 OpenFlow 流表存储成本优化的问题

Hash-TCAM 混合流表方案的目标是通过合理分配散列表与 TCAM 的容量，优化 Hash-TCAM 混

合流表查找结构的成本，降低其能耗，实现高性能的流表存储与查找。由于混合查找结构的成本由散列表与 TCAM 2 部分组成，设散列表的总成本为 C_{H}，TCAM 表的总成本为 C_{T}，总的流表条目数为 N，散列表容量为 H 个散列桶，散列表的负载系数 $\lambda=\frac{N}{H}$，单个散列桶容纳的关键字条目数为 ω，单个散列条目的成本为 C_{h}，散列表冲突溢出的条目数为 Θ，TCAM 的容量为 T 个条目，单个 TCAM 条目的成本为 C_{t}，则 Hash-TCAM 混合流表查找结构的成本优化问题可以记为

$$\min\ C_{\mathrm{total}}=C_{\mathrm{H}}+C_{\mathrm{T}} \tag{1}$$

$$\text{s.t. } T\geqslant\Theta \tag{2}$$

$$C_{\mathrm{H}}=H\omega C_{\mathrm{h}} \tag{3}$$

$$C_{\mathrm{T}}=TC_{\mathrm{t}} \tag{4}$$

其中，式(1)中 C_{total} 表示整个存储与查找结构的总成本，为优化的目标函数；式(2)的不等式是约束条件，表示 TCAM 的容量不少于散列冲突溢出的条目数；式(3)和式(4)分别表示散列表和 TCAM 的成本。取 TCAM 的容量恰好可容纳散列表冲突溢出的条目数

$$T=\Theta \tag{5}$$

对于容量为 H 个散列桶、单散列桶容量为 ω 条目的散列表，散列表容量为 h 个条目，则

$$h=H\omega \tag{6}$$

记单个 TCAM 条目的成本为散列条目成本的 m 倍，即

$$C_{\mathrm{t}}=mC_{\mathrm{h}} \tag{7}$$

将式(5)~式(7)代入式(1)得

$$\min\quad C_{\mathrm{total}}=hC_{\mathrm{h}}+\Theta mC_{\mathrm{h}} \tag{8}$$

由于散列表的成本 C_{H} 是散列表容量的增函数；TCAM 的成本由散列冲突溢出的条目数 Θ 决定，对于相同结构的散列表，散列冲突溢出的条目数 Θ 随散列表容量增加而减少，是散列表容量 h 的减函数，记 $\Theta=\Theta(h)$。通过分析散列容量 h 的变化对总成本 C_{total} 的影响，可以得出 OpenFlow 流表存储与查找结构最优成本下的散列表及 TCAM 各自的容量配置。记 Δh 为散列表条目数的增量，散列表容量 $h+\Delta h$ 下的混合存储与查找结构的成本为 $C_{\mathrm{total}}(h+\Delta h)$，则

$$C_{\mathrm{total}}(h+\Delta h)-C_{\mathrm{total}}(h)=\Delta hC_{\mathrm{h}}+[\Theta(h+\Delta h)-\Theta(h)]\,mC_{\mathrm{h}}$$

当 $C_{\mathrm{total}}(h+\Delta h)-C_{\mathrm{total}}(h)=0$ 时，得到 C_{total} 的极值点，即

$$\Delta hC_{\mathrm{h}}+[\Theta(h+\Delta h)-\Theta(h)]mC_{\mathrm{h}}=0 \tag{9}$$

由式(9)可得出

$$\frac{\Theta(h+\Delta h)-\Theta(h)}{\Delta h}=-\frac{1}{m} \tag{10}$$

当 $\frac{[\Theta(h+\Delta h)-\Theta(h)]}{\Delta h}<-\frac{1}{m}$ 时散列表的容量由 h 增大到 $h+\Delta h$ 时成本下降，即得到了优化；而当 $\frac{[\Theta(h+\Delta h)-\Theta(h)]}{\Delta h}>-\frac{1}{m}$ 时散列表的容量由 h 增大到 $h+\Delta h$ 时成本会增加。因此式(10)的成本极值点是极小值，即当散列冲突溢出的变化率等于 $-\frac{1}{m}$ 时，整个混合存储与查找结构的成本达到最优。为了进一步对式(1)进行求解，计算散列表与 TCAM 的容量配置，下面需要分析散列表的冲突溢出。

3.2　散列表的冲突分析

对于散列查找而言，当出现散列冲突时，可能需要多次访问存储器，而存储器的访问是影响查找性能的关键因素。为了减少查找过程中存储器的访问次数、保持高的查找性能，本文采用多单元散列表，即单个散列桶使用连续的存储空间容纳多个查找单元，其中，每个查找单元包含一个散列条目。通过这样的设计，一次存储器读操作可以读取多个散列条目。

散列表的冲突溢出条目数跟散列表的负载系数及结构密切相关，不同的散列表结构在使用相同容量存储器的情形下，冲突溢出条目数不同，称 $\varepsilon=\frac{\Theta}{N}$ 为散列表的冲突溢出率。下面分析不同结构及负载系数下的冲突溢出率。采用具有良好随机性的循环冗余校验(CRC, cyclic redundancy check)作为散列函数，N 个数据随机分布在 H 个键值中，对于任意一个特定的键值，每个数据进入这一键值的概率为 $\frac{1}{H}$，N 个数据中恰好有 k 个进入这一键值的概率服从二项分布 $B(N,\ \frac{1}{H})$，$f(k;\ N,\ p)$ $=\Pr(K=k)$ $=\mathrm{C}_N^k(p)^k(1-p)^{N-k}$，其中，$p=\frac{1}{H}$；即散列桶中恰好有 k 个条目的概率 $P(k)=\Pr(K=k)\approx\frac{\lambda^k}{k!}\mathrm{e}^{-\lambda}$，其中

$$\lambda=Np=\frac{N}{H}$$

对于单条目散列桶的普通散列表，负载系数为 λ 时，冲突溢出率可按式(11)计算。

$$\varepsilon=\frac{H\sum_{i=2}^{\infty}(i-1)P(i)}{N}=\frac{\mathrm{e}^{-\lambda}+\lambda-1}{\lambda} \tag{11}$$

对于散列桶容量为 ω 的多单元散列表，负载系数为 λ 时，冲突溢出率可按式(12)计算。

$$\varepsilon=\frac{H\sum_{i=\omega+1}^{\infty}(i-\omega)P(i)}{N}=\frac{\sum_{i=0}^{\omega-1}(\omega-i)P(i)-(\omega-\lambda)}{\lambda} \tag{12}$$

为了比较多单元散列表与普通散列表在相同容量下的冲突溢出率，取散列桶数目为 N、负载系数为 1 的多单元散列表与散列桶容量为 1、负载系数为 $\frac{1}{N}$ 的普通散列表进行分析。根据式(11)和式(12)可得出两者的冲突溢出率随散列表容量的变化趋势，如图 4 所示。

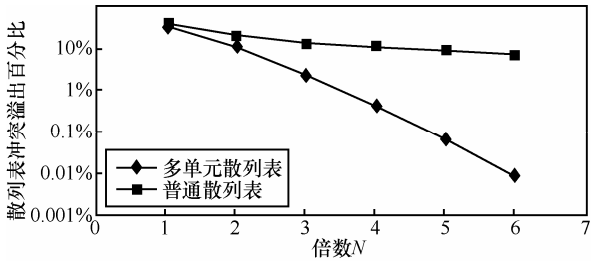


图 4 散列冲突溢出率随散列表的容量增加的变化趋势

从图 4 可以看出，在相同散列表容量情形下，多单元散列表的冲突溢出率明显低于普通散列表的冲突溢出率；而且随着散列表容量增加，多单元散列表的冲突溢出率下降幅度也明显大于普通散列表。因此，在数据总线宽度允许的条件下，应尽可能选择单元数多的多单元散列表存储与查找 OpenFlow 流表，以优化流表的存储与查找成本。

根据式(12)，表 1 给出了多单元散列表不同的散列桶单元数目、负载系数 λ 下的冲突溢出率。对于表 1 中散列桶单元数为 ω、负载系数为 λ 的散列表，散列桶的数目为 $\frac{N}{\lambda}$，表的容量为 $\frac{\omega N}{\lambda}$；散列桶单元数同为 ω，负载系数分别为 λ、λ+1 相邻的 2 个表格单元，其容量差为 $\frac{\omega N}{\lambda(\lambda+1)}$，根据式(10)可知，当两者的冲突溢出率差值接近 $-\frac{\omega}{\lambda(\lambda+1)m}$ 时，其存储与查找结构的成本接近最优值。

3.3 多单元散列表的结构及处理方法

3.3.1 存储与查找结构

由于 OpenFlow 流表表项的匹配关键字包含的位数非常多(OpenFlow1.0 规范中至少包含 228 bit)，为了能够实现多个单元的并行比较，需要对匹配关键字进行压缩，可以使用具有良好随机性的散列函数将流表的匹配关键字压缩成位数较少的固定长度的指纹(fingerprint)[26]，在散列查找时通过指纹的比较发现匹配的流表表项。基于多单元散列表的 OpenFlow 流表结构如图 5 所示，每个散列桶包含 ω 个查找单元，每个查找单元由标识位、指纹、流表指针 3 个部分组成，其中，标识位表示该查找单元中的流表条目是否有效，在删除流表表项时只要将对应的标识位置为无效即可，表项指针指向具体的流表条目。

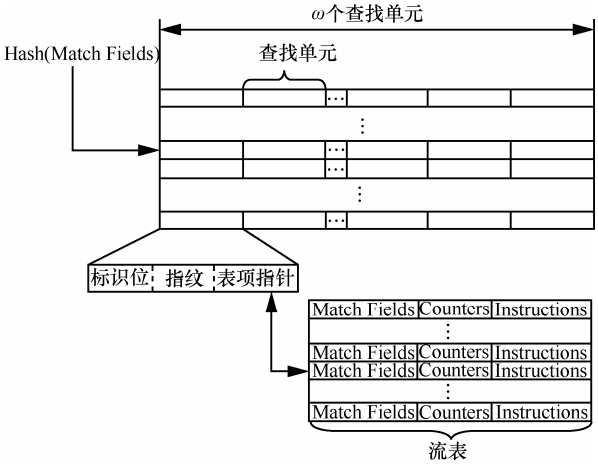


图 5 基于多单元散列表的 OpenFlow 流表结构

表 1 多单元散列表冲突溢出率

λ	2 单元	3 单元	4 单元	5 单元	6 单元	7 单元	8 单元
λ=1	10.364%	2.334%	0.435%	0.069%	0.009 5%	0.001 1%	
λ=2	27.067%	10.901%	3.757%	1.124%	0.296%	0.069 5%	0.014 7%
λ=3		22.404%	10.645%	4.487%	1.690%	0.573%	0.176%
λ=4			19.537%	10.258%	4.886%	2.119%	0.841%
λ=5				17.547%	9.866%	5.109%	2.442%
λ=6					16.062%	9.501%	5.234%
λ=7						14.900%	9.168%
λ=8							13.959%

3.3.2　处理流程

图 6 是 Hash-TCAM OpenFlow 流表查找的处理流程，其输入为匹配关键字 key。TCAM 的查找操作与 SRAM 的读操作可以并行的执行，当 TCAM 查找操作返回的流表项指针有效时，直接根据该流表指针读取流表条目，不必再判断寄存器 Registers 中的内容及后续操作。当 TCAM 查找操作返回的流表指针无效时，则需要根据寄存器中的指纹判断散列表中是否存在匹配的流表表项，如果存在匹配的指纹，则根据对应的流表指针读取相应的流表条目，并比较流表条目中的关键字与查找使用的关键字是否一致，如果不一致，则说明存在指纹冲突需要插入新的流表条目。当在 TCAM 与散列表中均未查找到匹配的流表条目，则请求插入新的流表条目。

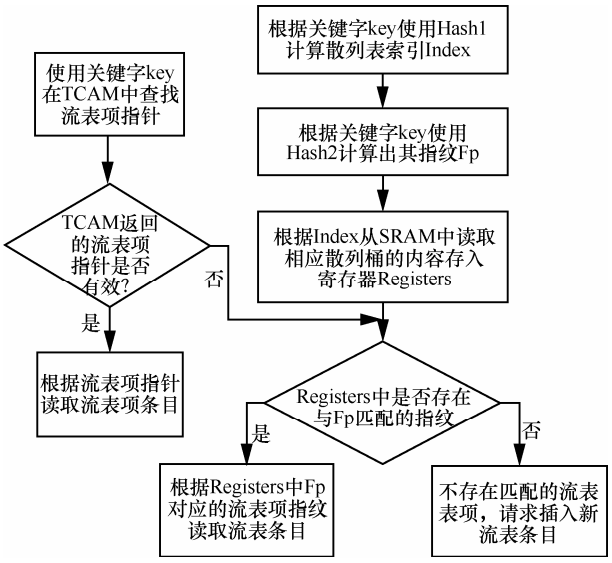


图 6　Hash-TCAM OpenFlow 流表查找流程

图 7 是 Hash-TCAM OpenFlow 流表表项插入的处理流程，其输入为流表关键字 key 及流表条目，在当前流表中未发现流表关键字 key 对应的流表条目时，则执行流表表项的插入。插入新的流表条目时，首先尝试向散列表中插入，如果由于散列冲突溢出或指纹冲突，则插入到 TCAM 中。在删除散列表中的流表条目时，只需要将标识位置为无效即可。

3.4　匹配关键字指纹

将取值范围较大的匹配关键字映射到取值范围较小的指纹，可能会发生不同的匹配关键字映射到同一个指纹的现象，称之为指纹冲突。为了确保查找的性能，需要将发生指纹冲突的匹配关键字存储在 TCAM 中。较少位数的指纹意味着散列表的容量小、成本低；但位数越少，指纹冲突的概率就越大，增加 TCAM 的成本，同时影响查找结构的性能。下面分析指纹冲突的发生率与指纹位数的关系。记位于同一散列桶的 n 个单元的指纹不出现冲突的概率为 $P_{nc}(n)$，指纹的位数为 w，任意关键字映射到指纹后随机地分布在[0, 2^w−1]的范围内，n 个单元均不存在指纹冲突的概率为

$$P_{nc}(n)=\frac{U(U-1)\cdots(U-n+1)}{U^n}=\prod_{i=1}^{n-1}\left(1-\frac{i}{U}\right) \tag{13}$$

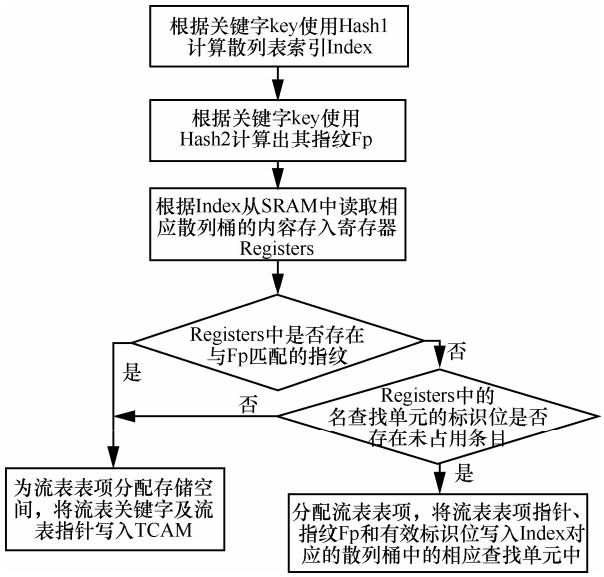


图 7　Hash-TCAM OpenFlow 流表表项插入处理流程

式(13)中 $U=2^w$，指纹冲突率 $P_c\leqslant 1-P_{nc}(n)$，即

$$P_c\leqslant 1-\prod_{i=1}^{n-1}\left(1-\frac{i}{U}\right) \tag{14}$$

根据式(14)，图 8 给出了 2 单元至 8 单元散列表的指纹冲突率上限与指纹位数的关系，从图 8 可以看出，当指纹的位数越多，冲突率的上限越小，对于多单元散列表，单元数目越多指纹冲突率上限越大；当指纹位数超过 21 bit 时，冲突溢出率的上限低于 10^{-5}。指纹发生冲突的概率上限与指纹的位数相关，而与匹配关键字的原始位数无关。

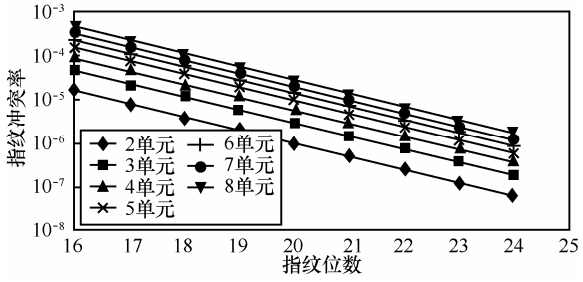


图 8　多单元散列表的指纹冲突率上限

3.5 两级多单元散列表的结构及操作

3.5.1 存储与查找结构

多级散列表[27,28]在实现散列表高利用率的同时，可以有效降低散列冲突的溢出率。然而如果散列表的级数较多，当执行删除操作时，则需要在不同级的子表之间执行散列条目的移动，才能保持低的冲突溢出概率[29]；而散列条目的移动会影响整个散列表的操作性能，而且散列表的级数过多，也会增加电路设计的复杂度，增加查找的成本与能耗。因此，为了进一步降低 OpenFlow 流表的查找成本，提出了两级多单元散列表的 OpenFlow 流表查找结构，如图 9 所示。该结构使用了主表与辅表两级子表，每个子表都使用多单元散列表；为了便于实现，每个子表的散列桶的单元数也相同，每个查找单元的格式与 3.3 节中的相同。

3.5.2 处理流程

图 10 是两级多单元 Hash-TCAM OpenFlow 流表查找的处理流程，其输入为匹配关键字 key。在执行关键字查找时，TCAM 查找与散列查找并行执行；对于散列查找，主表与辅表 SRAM 的读操作也并行地执行；当 TCAM 查找操作返回的流表项指针无效时，需要根据 Register1、Register2 中的内容判断散列表中是否存在匹配的流表项，否则直接根据 TCAM 中返回的流表项指针读取流表条目，不必再判断 Register 中的内容及其后续操作。

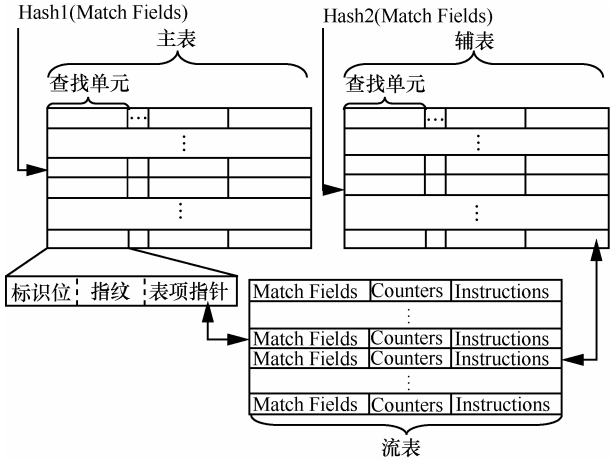


图 9 两级多单元散列表的详细结构

图 11 是两级多单元 Hash-TCAM OpenFlow 流表表项插入的处理流程，其输入为：流表关键字 key 及流表条目。在流表中未查找到流表关键字 key 对应的流表条目时，则执行流表表项的插入。插入新的流表条目时，首先尝试向散列表的主表中插入，如果由于散列冲突溢出或指纹冲突，则尝试向散列表的辅表中插入，如果仍然存在散列冲突溢出或指纹冲突，则插入到 TCAM 中。

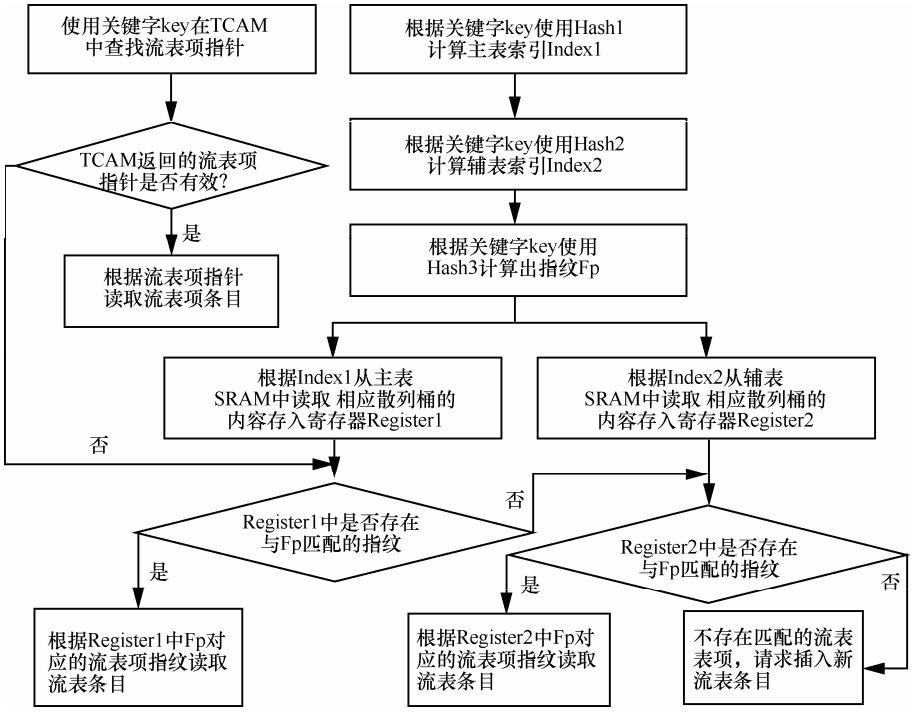


图 10 两级多单元 Hash-TCAM OpenFlow 流表查找处理流程

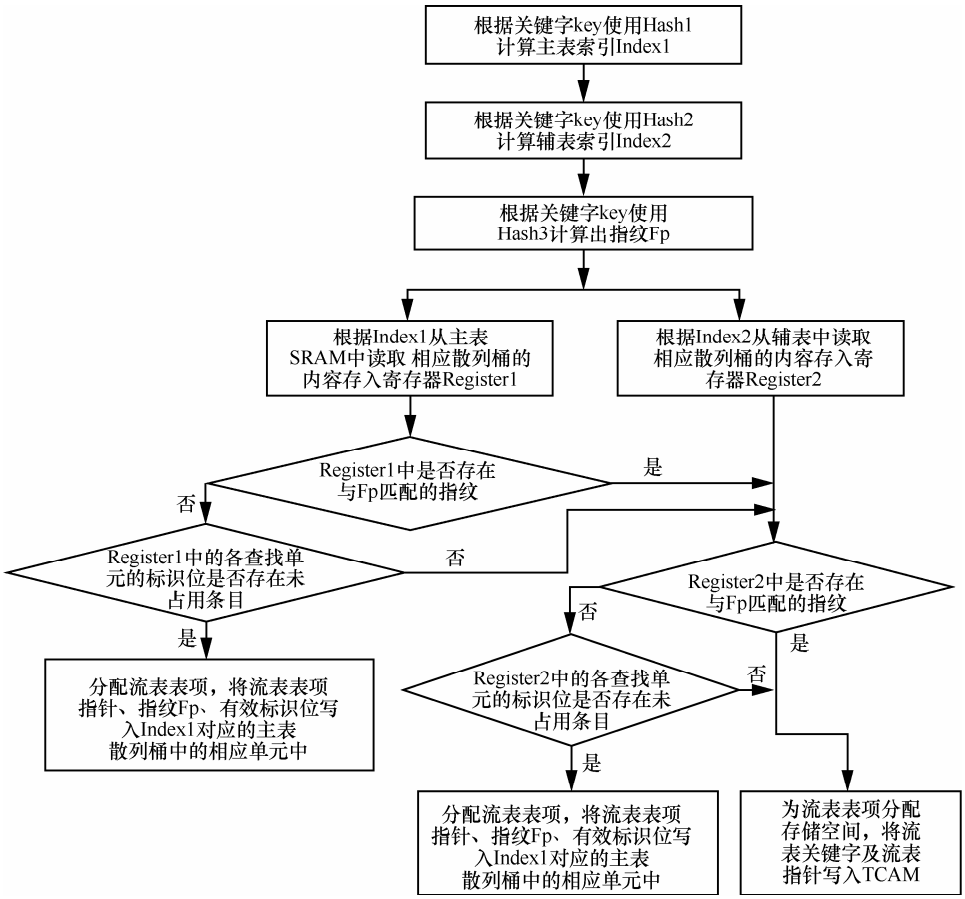


图 11　两级多单元 Hash-TCAM OpenFlow 流表插入处理流程

3.5.3　散列冲突溢出分析

下面分析散列桶单元数相同，两级多单元散列表与单级多单元散列表的冲突溢出率。设单级多单元散列表容量为 H 个散列桶，单个散列桶的容量为 2ω 个查找单元，负载系数为 2λ，冲突溢出率可按式(15)计算

$$\varepsilon(2\omega,2\lambda)=\frac{H\sum_{i=2\omega+1}^{\infty}(i-2\omega)P(i,2\lambda)}{N}=\frac{\sum_{i=0}^{2\omega-1}(2\omega-i)P(i,2\lambda)-(2\omega-2\lambda)}{2\lambda} \tag{15}$$

式(15)中的 $P(i,2\lambda)\approx\frac{\lambda^i}{i!}\mathrm{e}^{-2\lambda}$ 。

设两级多单元散列表主表容量为 H 个散列桶，单个散列桶的容量为 ω 个查找单元，主表的冲突溢出率 ε 可按式(16)计算。

$$\varepsilon(\omega,\lambda)=\frac{H\sum_{i=\omega+1}^{\infty}(i-\omega)P(i,\lambda)}{N}=\frac{\sum_{i=0}^{\omega-1}(\omega-i)P(i,\lambda)-(\omega-\lambda)}{\lambda} \tag{16}$$

其中，$P(i,\lambda)\approx\frac{\lambda^i}{i!}\mathrm{e}^{-\lambda}$。

辅表容量为 $\varepsilon(\omega,\lambda)$的 H 个散列桶，单个散列桶的容量同样为 ω 个查找单元，负载系数为 λ，则两级多单元散列表的冲突溢出率可按式(17)计算。

$$\varepsilon_2(\omega,\lambda)=\varepsilon(\omega,\lambda)\varepsilon(\omega,\lambda) \tag{17}$$

两级多单元散列表的总容量为$(1+\varepsilon(\omega,\lambda))H\omega$ 个查找单元，其冲突溢出率为 $\varepsilon_2(\omega,\lambda)$；单级多单元散列表的容量为 $H\cdot2\omega$ 个查找单元，其冲突溢出率为 $\varepsilon(2\omega,2\lambda)$；在 $1\leqslant\omega\leqslant40$, $1\leqslant\lambda\leqslant\omega$ 的条件下，通过计算可以得出 $\varepsilon_2(\omega,\lambda)<\varepsilon(2\omega,2\lambda)$，即与单级多单元散列表相比，两级多单元散列表在使用更少的存储空间的条件下，可以实现更低的冲突溢出率。

4　测试与仿真分析

散列表冲突溢出率是 Hash-TCAM 查找结构成

本计算的基础，因此散列表冲突溢出率的理论模型的准确性，对 Hash-TCAM 混合查找结构的成本计算非常关键，通过实验测试散列表冲突溢出率的理论值与实际数据的吻合度。流表关键字的指纹冲突会对查找性能产生影响，应该尽量避免指纹冲突，并通过实验测试理论上限分析的合理性。

本节使用自生成的伪随机数与 2 所大学的数据中心报文 Trace[30]数据对流表 Hash-TCAM 混合存储与查找结构中的散列表的冲突溢出率、指纹冲突率进行测试，对散列表与 TCAM 容量配置的有效性进行了分析，考察了混合存储查找结构的成本与能耗。

为了验证散列表的冲突溢出率理论分析的准确性，分别生成了 64K、128K、256K、512K、1M、2M、4M、8M 个伪随机数（其中，K=1 024，M=1 024×1 024），每个伪随机数 96 bit，作为关键字存储到散列表中。

为了验证方案针对实际网络数据的适用性，将 Trace 数据集[30]中的报文头部作为流表中的匹配字段存储到散列表中。由于通过报文头部的<IP 源地址、IP 目的地址、传输层源端口号、传输层目的端口号>便可以唯一地标识一个数据流，因此测试中只取了报文头的这 4 个部分作为流表的匹配字段。提取 UNV1 中的 20 个文件、UNV2 中的 8 个文件，将报文头四元组作为流表的匹配字段，2 个数据集分别包含 556 601 个条目和 191 474 个条目。

测试过程中，使用 CRC-32[31]算法生成散列查找的索引，计算匹配关键字指纹使用的函数为 Jenkins Hash[32]。

4.1 散列表冲突溢出测试

图 12 给出了 2 单元至 8 单元散列表的冲突溢出率，图中的纵坐标为百分比，横坐标为散列表的负载系数 λ。从图中可以看出，无论是使用随机数据作为关键字存储在散列表中，还是数据中心 Trace 数据提取出的报文头作为关键字存储在散列表中，实验得出的冲突溢出率与理论值均非常吻合。

图 12 中的理论值是散列表冲突溢出率的理论期望值，由于散列桶中表项的条目数服从二项分布，散列冲突的溢出条目数的期望值也服从二项分布，根据棣莫弗—拉普拉斯中心极限定理可知，当流表的条目数很大时，冲突溢出的概率非常接近其期望值，因此，散列冲突溢出率实验值与其理论期望值非常接近。

4.2 指纹冲突测试

图 13 给出了 2 单元至 8 单元散列表的指纹冲突率，图中的横坐标为匹配关键字指纹的位数，纵坐标为冲突率，即发生指纹冲突的数目与总条目数之比，其中的指纹冲突率的理论值是冲突率的上限。从图中可以看出，无论是使用随机数据生成的指纹存储在散列表中，还是 Trace 数据集中的报文头计算出的指纹存储在散列表中，指纹冲突率均不超过理论值给出的上限。随着单元数的增加，指纹冲突率呈现增长的趋势。由于发生指纹冲突的概率非常小，几个指纹冲突就会导致测试结果呈现出较大的波动，但总的来看指纹的冲突率均不高于其理论上限。

4.3 成本与能耗分析

对于散列查找而言，需要存储内容包括：匹配关键字，匹配关键字的指纹、有效标识位、流表指针。根据 3.2 节的理论分析与 4.2 节的仿真测试，匹配关键字的指纹长度取 21 bit 以上时，其指纹冲突率低于 10^{-5}，与散列冲突溢出率相比可以忽略不计；有效标识位为 1 bit，流表指针的位数为 lb N，N 为流表条目数。这样，当流表的条目数不超过 8 M 的情况下，对于散列查找，除了匹配关键字以外每个条目还需要不超过 45 bit 的存储开销。OpenFlow 流表匹配关键字的位数通常超过 228 bit，所以单个散列条目的容量不超过 TCAM 条目的 $\frac{45+228}{228}$ 倍。假定 TCAM 的单比特成本是 SRAM 的 30 倍[6]，则单个 TCAM 条目成本是散列条目的 25 倍以上。OpenFlow 交换机的报文处理芯片对外接口等实现因素限制了散列桶的单元数，表 2 给出了散列桶单元数为 2~8 时成本最优的 Hash-TCAM 存储与查找结构中散列表、TCAM 的配置，以及整个查找结构的成本。其中，N 为总的流表条目数，散列表的主表、辅表以及 TCAM 配置是根据散列冲突溢出率的理论值计算出的。其中 UNV1 与 UNV2 的 TCAM 条目数是由 3.1 节中的测试得出，N_1=556 601，N_2=191 474；UNV1 与 UNV2 的成本是归一化的成本。

从表 2 的数据可以看出，随着散列桶单元数的增加，Hash-TCAM 流表存储与查找结构的成本逐渐降低，即使散列桶为 2 单元的情形下，$\frac{2.242}{25}$=8.97%，Hash-TCAM 存储与查找结构的最优成本可以比只使用 TCAM 的方案节约 90%以上

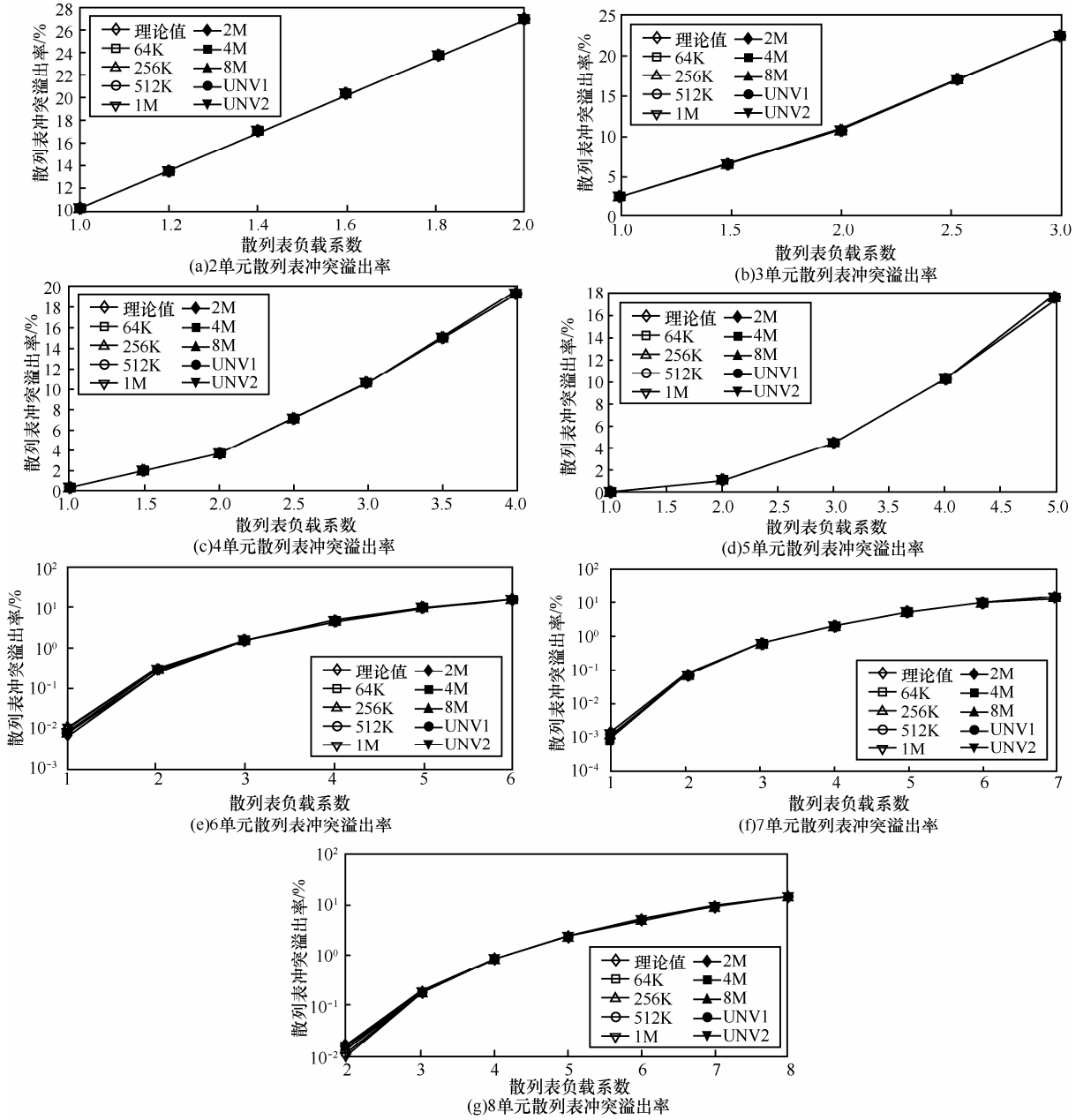


图 12　多单元散列表冲突溢出率

的成本。

为了确保 Hash-TCAM 混合结构的查找性能，当执行查找时并行地查找散列表的主表、辅表以及 TCAM，因此其能耗包括这 3 部分的查找能耗。在 0.18 μm 生产工艺、工作电压为 1.8 V 的情况下，当 TCAM 容量超过 256 Kbit 情形，其能耗相当于相同容量 SRAM 访问能耗的 15 倍以上[7]，且 TCAM 的能耗随并行匹配字段条目数及宽度增加[6,7]；即使 1K 条目的 OpenFlow 流表，占用的 TCAM 也超过 256 Kbit，因此计算时取 TCAM 的能耗是相同容量 SRAM 的 15 倍。为了方便计算，记容量为 N 个流表条目的 TCAM 能耗为 15，表 2 中 UNV1 与 UNV2 的能耗是归一化后的能耗。仍以散列桶为 2 单元的散列表为例，$\frac{1.962}{15}=13.08\%$，Hash-TCAM 存储与查找结构成本最优条件下，其能耗与只使用 TCAM 的方案相比可以节约 85%以上。

4.4　性能分析

基于两级多单元散列表与 TCAM 的 OpenFlow 流表查找，需要同时并行地查找 TCAM、主表与辅表，包含的操作有：散列表索引的计算、指纹的计算、寄

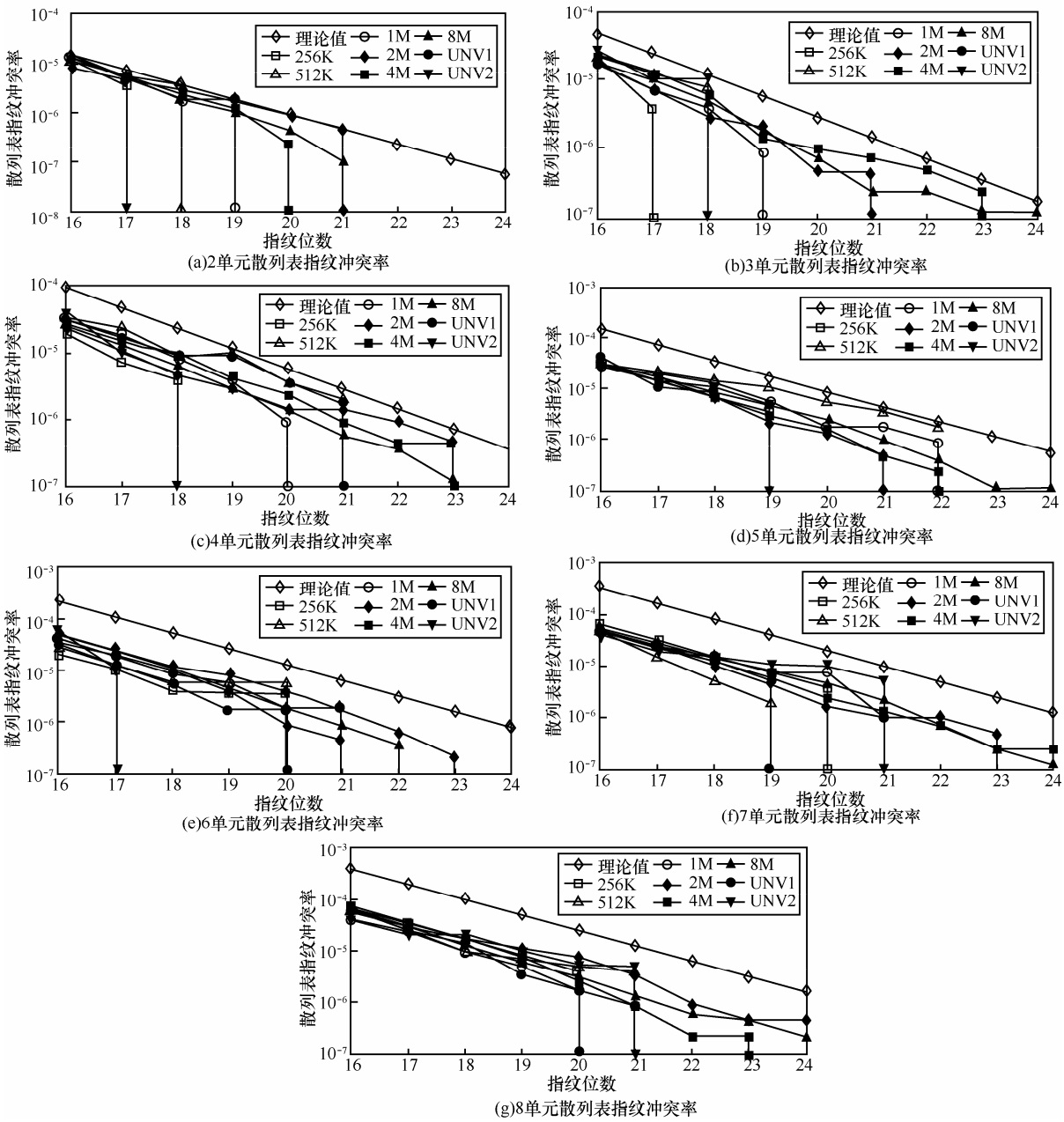


图 13　多单元散列表指纹冲突率

存器内容与指纹的比较、一次 TCAM 查找、一次主表 SRAM 读操作、一次辅表 SRAM 读操作。对报文处理器而言，SRAM 的读写操作以及 TCAM 查找是报文处理器的性能瓶颈；衡量报文处理器查表性能的指标是单位时间的查找次数；对 SRAM 每个时钟周期都可以执行一次读取操作[33]；当 OpenFlow 交换机收到报文执行 OpenFlow 流表查找时，其中，TCAM 查找、主表 SRAM 的读操作、辅表 SRAM 的读操作均可以并行地执行，即每个时钟周期都可以执行一次 TCAM 查找、主表 SRAM 的读取以及辅表 SRAM 的读取；因此基于两级多单元散列表的 OpenFlow 流表查找的性能与基于 TCAM 的流表查找性能是相当的。

SystemC[34]一种应用广泛的系统级建模与仿真的语言，能够完成对电子系统从软件到硬件的全部过程进行建模，因此本节使用 SystemC 对两级多单元 Hash-TCAM 混合 OpenFlow 流表存储与查找结构进行了建模与仿真(具体的软件仿真环境为 SystemC2.1 与 Microsoft VC++6.0)。仿真模型中以数据中心报文 Trace 数据[30]中提取的报文流作为输入数据执行流表的查找与插入；其中，SRAM 与 TCAM 采用相同的时钟频率，目前，常见的 SRAM 及 TCAM 时钟频率有 250 MHz、333 MHz、

表 2　　Hash-TCAM 表的配置与成本及能耗

数据	2 单元	3 单元	4 单元	5 单元	6 单元	7 单元	8 单元
主表	$\frac{N}{2}$	$\frac{N}{3}$	$\frac{N}{4}$	$\frac{N}{5}$	$\frac{N}{6}$	$\frac{N}{7}$	$\frac{N}{8}$
辅表	0.271N	0.224N	0.098N	0.088N	0.054N	0.037N	0.0349N
TCAM 理论值	0.028N	0.005 2N	0.007 3N	0.002N	0.002N	0.003 2N	0.001 2N
Univ1-TCAM	0.027 7N_1	0.005 18N_1	0.007 42N_1	0.002N_1	0.002 58N_1	0.002 98N_1	0.001 22N_1
Univ2-TCAM	0.027 5N_2	0.005 35N_2	0.007 82N_2	0.001 44N_2	0.002 7N_2	0.003 33N_2	0.001 26N_2
成本理论值	2.242	1.802	1.575	1.49	1.392	1.339	1.309
Univ1-成本	2.235	1.802	1.578	1.49	1.389	1.334	1.309 7
Univ2-成本	2.225	1.816	1.579	1.48	1.392	1.342	1.310 7
能耗理论值	1.962	1.75	1.502	1.47	1.365	1.307	1.297
Univ1-能耗	1.958	1.75	1.503	1.47	1.363	1.304	1.297
Univ1-能耗	1.955	1.752	1.509	1.462	1.365	1.309	1.298

450 MHz，即 TCAM 可以实现每秒 250M、333M、450M 次的查找，SRAM 可以实现每秒 250M、333M、450M 次的读写操作，因此，仿真时分别采用这些时钟频率，仿真过程中关键字的指纹采用了 23 bit。分别对 2~8 单元的两级多单元 Hash-TCAM 混合查找结构和纯 TCAM 查找结构的查找性能进行了测试，图 14 所示的仿真结果表明，两级多单元 Hash-TCAM 混合查找结构在相同时钟频率的情况下，与纯 TCAM 查找方案具有相近的查找性能。

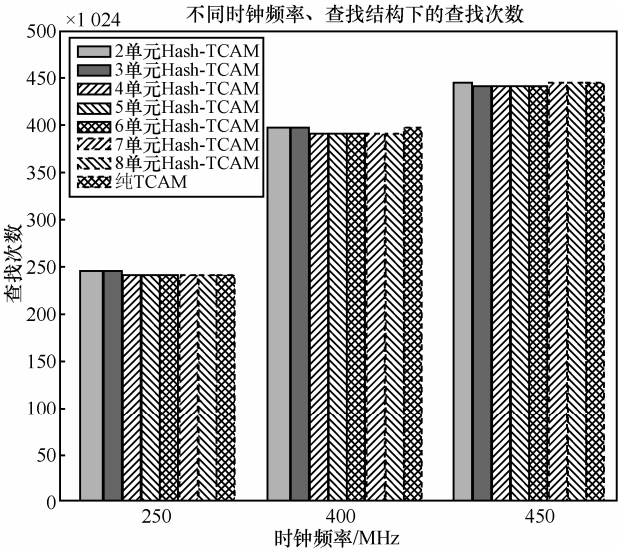


图 14　Hash-TCAM 与纯 TCAM 的查找次数对比

5　结束语

针对基于 TCAM 的 OpenFlow 流表存储与查找机制存在的高成本、高能耗问题，提出了基于多单元 Hash-TCAM 的混合流表查找机制。为了优化 OpenFlow 流表存储与查找的成本与能耗，进一步采用两级多单元散列表机制，在确保流表查找性能的同时，降低散列表的成本。

理论分析与仿真测试表明，基于多单元 Hash-TCAM 混合流表查找结构，针对精确匹配的流表条目，在优化配置情况下，可以节约成本 90% 以上，同时有效降低了其能耗；所提出的方案非常适合流量管理或报文转发为主要功能的 OpenFlow 交换机。尤其是当执行报文处理的 NP 或 ASIC 带有一定容量的片内 TCAM 时，只需要配置适当容量的 SRAM 或 RLDRAM 用作流表的散列匹配，便可以实现高性能、低成本、低能耗的 OpenFlow 流表查找。

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.

[2] Open Networking Foundation. OpenFlow switch specification Version 1.1.0 (Wire Protocol 0x02)[S/OL].https://www. opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/ openflow/openflow-spec-v1.1.0.pdf, 2011.

[3] Open Networking Foundation. OpenFlow switch specification Version 1.3.0 (Wire Protocol 0x04) [S/OL]. https://www. opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/ openflow/openflow-spec-v1.3.0.pdf, 2012.

[4] Open Networking Foundation. OpenFlow switch specification Version 1.5.0 (Protocol version 0x06) [S/OL]. https://www. opennetwork-

ing.org/images/stories/downloads/sdn-resources/onf-specifications/ openflow/openflow-switch-v1.5.0.pdf, 2014.

[5] CONGDON P T, MOHAPATRA P, FARRENS M, et al. Simultaneously reducing latency and power consumption in OpenFlow switches[J]. IEEE/ACM Transactions on Networking, 2014, 22(3): 1007-1020.

[6] TAYLOR D E. Survey and taxonomy of packet classification techniques [J]. ACM Computing Surveys, 2005, 37(3): 238-275.

[7] AGRAWAL B, SHERWOOD T. Modeling ternary CAM power and delay model: extensions and uses[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008, 16(5): 554-564.

[8] 刘中金, 李勇, 苏厉, 等. TCAM 存储高效的 OpenFlow 多级流表映射机制[J]. 清华大学学报(自然科学版), 2014, 54: 437-442.
LIU Z J, LI Y, SU L, et al. TCAM-efficient flow table mapping scheme for OpenFlow multiple-table pipelines[J]. Journal of Tsinghua Univ Sci&Technol, 2014, 54: 437-442.

[9] GE J, CHEN Z, WU Y, et al. H-SOFT: a heuristic storage space optimization algorithm for flow table of OpenFlow [J]. Concurrency and Computation: Practice and Experience, 2015, 27(13):3497-3509.

[10] CHEN Z, WU Y, GE J, et al. A new lookup model for multiple flow tables of OpenFlow with implementation and optimization considerations [C]//IEEE International Conference on Computer and Information Technology (CIT). Xi'an, IEEE, 2014:528-532.

[11] 鄂跃鹏, 陈智, 葛敬国,等. 一种高效的 OpenFlow 流表存储与查找实现方法[J]. 中国科学:信息科学, 2015, 45(10): 1280-1288.
E Y P, CHEN Z, GE J, et al. An efficient implementation of storage and lookup for flow tables in OpenFlow [J]. Scientia Sinica Informationis, 2015, 45(10): 1280-1288.

[12] SUN H, SUN Y, VALGENTI V C, et al. OpenFlow accelerator: a decomposition-based hashing approach for flow processing[C]//24th International Conference on Computer Communication and Networks (ICCCN). Las Vegas: IEEE, 2015: 1-10.

[13] ZHU H, XU M, LI Q, et al. MDTC: An efficient approach to TCAM-based multidimensional table compression[C]//IFIP Networking Conference. Toulouse, IFIP, 2015:1- 9.

[14] MCGEER R, YALAGANDULA P. Minimizing rule sets for TCAM implementation[C]//IEEE INFOCOM. Rio de Janeiro, IEEE, 2009: 1314-1322.

[15] VEERAMANI S, KUMAR M M, NOOR S K. Hybrid trie based partitioning of TCAM based openflow switches[C]//IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). Chennai, IEEE, 2013: 1-5.

[16] LIM H, LEE S, SWARTZLANDER E E J. A new hierarchical packet classification algorithm [J]. Computer Networks, 2012, 56(13): 3010-3022.

[17] CORMODE G, THOTTAN M. Algorithms for next generation networks [M]. London: Springer, 2010: 181-218.

[18] PAGH R, RODLER F. Cuckoo hashing[J]. Journal of Algorithms, 2004, 51(2): 122-144.

[19] KIRSCH A, MITZENMACHER M, WIEDER U. More robust hashing: cuckoo hashing with a stash[C]//16th Annual European Symposium on Algorithms, Karlsruhe (L). Springer Verlag, Heidelberg, 2008: 611-622.

[20] CURTIS A R, KIM W, YALAGANDULA P. Mahout: low overhead datacenter traffic management using end-host-based elephant detection[C]//IEEE INFOCOM. Shanghai, 2011: 1629-1637.

[21] MOHAMMAD A F, RADHAKRISHNAN S, RAGHAVAN B, et al. Hedera: dynamic flow scheduling for datacenter networks[C]// USENIX Association NSDI. San Jose: USENIX, 2010: 281-296.

[22] LI D, SHANG Y, CHEN C. Software defined green data center network with exclusive routing[C]//IEEE INFOCOM. Toronto, IEEE, 2014: 1743-1751.

[23] FERKOUSS O E, SNAIKI I, MOUNAOUARO, et al. A 100Gig network processor platform for OpenFlow[C]//Conf Network and Service Management (CNSM), Paris: IEEE, 2011.

[24] LUO Y,CASCON P, MURRAY E, et al. Accelerating OpenFlow switching with network processors[C]//ACM/IEEE Symposium on Architecture for Networking and Communications Systems(ANCS). Princeton, ACM, 2009: 70-71.

[25] RECEP O. Intel® Ethernet Switch FM6000: SDN with OpenFlow[EB/OL].http://www.intel.com/content/www/us/en/switch-silicon/ethernet-switch-fm6000-sdn-paper.html, 2014.

[26] MICHAEL O R. Fingerprinting by random polynomials[R]. Center for Research in Computing Technology Harvard University Report TR-15-81, 1981.

[27] BRODER A Z, KARLIN A R. Multilevel adaptive hashing[C]// ACM-SIAM Symposium on Discrete Algorithm. San Francisco, ASSOC COMP MACHINERY, 1990: 43-53.

[28] KUMAR S, TURNER J, CROWLEY P. Peacock hashing: deterministic and updatable hashing for high performance networking[C]//IEEE INFOCOM. Phoenix, IEEE, 2008: 101-105.

[29] KIRSCH A, MICHAEL M. On the Performance of multiple choice hash tables with moves on deletes and inserts[C]//46th Annual Allerton Conference on Communication, Control, and Computing, 2008: 1285-1290.

[30] THEOPHILUS B. Data set for IMC 2010 data center measurement[EB/OL]. http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html, 2010.

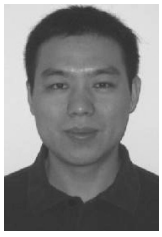
[31] BRAYER K, HAMMOND J L. Evaluation of error detection polynomial performance on the AUTOVON channel[C]//National Telecommunications Conference. New Orleans, IEEE, 1975: 21-25.

[32] https://en.wikipedia.org/wiki/Jenkins_hash_function[EB/OL].

[33] GIRISH K. QDR®-II, QDR-II+, DDR-II, and DDR-II+ Design Guide[EB/OL].http://www.cypress.com/file/38596/download, 2015.

[34] http://accellera.org/downloads/standards/systemc[EB/OL].

作者简介：



李春强（1975-），男，山东沾化人，东南大学博士生，主要研究方向为数据中心网络体系结构及传输控制、报文转发及查找算法等。

董永强（1973-），男，河南渑池人，博士，东南大学副研究员，主要研究方向为网络体系结构、移动网络计算、高效内容分发等。

吴国新（1956-），男，安徽芜湖人，东南大学教授、博士生导师，主要研究方向为网络协议、网络安全和自组网等。