一种保持 OpenFlow 功能完整性的 TCAM 流表压缩模型*

席孝强[1]，兰巨龙[1]，孙鹏浩[1]，江逸茗[1]，刘 博[2]
（1. 国家数字交换系统工程技术研究中心，郑州 450002；2. 信息工程大学，郑州 450001）

摘 要：OpenFlow 协议版本的更新带来流表项匹配域支持字段的增加，而表项匹配域支持的任意通配符依赖 TCAM 进行匹配处理，导致设备中的 TCAM 存储空间面临很大压力。为此，提出一种减小 TCAM 中流表存储空间的数学模型 FICO（function-integral TCAM-saving compression model for flow table of OpenFlow）。FICO 首先根据匹配域不同字段间的关系，将字段之间的冗余分为三种，基于冗余提出三种预压缩算法，分别为域间字段合并、字段映射、域内字段压缩，最终组合为更小位宽的表项被送往 TCAM 中进行流匹配。通过仿真表明在保持 OpenFlow 功能完整性的前提下，较未压缩流表，FICO 可以节省 60% TCAM 存储空间。并且随着流表规模的增大，压缩性能保持稳定。

关键词：OpenFlow；TCAM；流表；存储空间优化；SDN
中图分类号：TP393.04 **文献标志码**：A **文章编号**：1001-3695(2018)05-1464-06
doi:10.3969/j.issn.1001-3695.2018.05.040

Function-integral TCAM-saving compression model for flow table of OpenFlow

Xi Xiaoqiang[1], Lan Julong[1], Sun Penghao[1], Jiang Yiming[1], Liu Bo[2]
(1. *National Digital Switching System Engineering & Technological R&D Center*, *Zhengzhou* 450002, *China*; 2. *Information Engineering University*, *Zhengzhou* 450001, *China*)

Abstract: In recent years, the fine-grained control ability of OpenFlow in data forwarding plane has increased greatly due to the constant upgrade of protocol version. However, expansion of supported fields in OpenFlow brings additional pressure to the storage space of TCAM (ternary content addressable memory) in physical device, since the arbitrary wildcard support in match field relies heavily on TCAM for matching process. To solve this problem, this paper proposed a mathematical model aiming at storage space reduction of the flow table in TCAM, which named with FICO: a function-integral TCAM-saving compression model for flow table of OpenFlow. FICO analyzed the relationships among all the fields and then categorize the redundancy among different fields into three types. Based on the three redundancy types, this paper proposed three compression algorithms named as inter-field merge, field mapping and intra-field compression, which are mainly performed in RAM space. The outcomes of each compression algorithm were flow entries with smaller bit-width which is sent to TCAM for flow matching. In this way, the flexibility of OpenFlow is not harmed, thus maintaining the function integrity of the original flow table. Simulation at the end shows that FICO saves almost about 60% of TCAM space for a given flow table with no damage to the function integrity of OpenFlow, and the compression performance stands stable with the increase of flow table size.

Key words: OpenFlow; TCAM; flow table; storage space optimization; SDN

0 引言

近年来，软件定义网络[1]（software defined networking, SDN）已经被广泛地应用在各种场景下。SDN 将控制平面与数据平面分离，给网络用户和研究者带来了更大的灵活性，如用户定义流控制和开放可编程性，这些在很大程度上推动了未来网络的变革。为了实现上层的灵活性，SDN 需要一个强有力的南向接口来管理数据平面上的基本硬件功能。

OpenFlow[2] 作为现在应用较为广泛的南向接口，由于获得 ONF（open networking foundation）的支持，OpenFlow 版本得到持续的更新。在 OpenFlow 1.3 版本中，匹配域拥有 40 个字段，占据 1 227 bit 的存储空间。虽然匹配域字段的增加扩大了 OpenFlow 的应用范围，但是 TCAM 有限的存储空间严重限制了流表的规模。并且 TCAM 具有花费密度大（$350/1Mbit）与高耗能（15 Watt/1 Mbit）的缺点[3]，这些严重限制了商用设备中 TCAM 的大小。

为了在常规的 TCAM 中存储更多的流表项，通常考虑对流表进行压缩。OpenFlow 流表压缩不同于 IP 路由表、ACL 表

收稿日期：2017-01-12；**修回日期**：2017-03-13 **基金项目**：国家“973”计划资助项目（2013CB329104）；国家自然科学基金资助项目（61372121，61309019，61502530）；国家自然科学基金创新群体资助项目（61521003）；国家“863”计划资助项目（2013AA013505，2015AA016102）

作者简介：席孝强（1989-），男，河南固始人，硕士研究生，主要研究方向为新型网络体系结构（1052418674@qq.com）；兰巨龙（1962-），男，教授，博士，主要研究方向为宽带信息网络、新型网络体系结构；孙鹏浩（1992-），男，硕士研究生，主要研究方向为宽带信息网络；江逸茗（1984-），男，助理研究员，博士，主要研究方向为宽带信息网络；刘博（1995-），男，主要研究方向为宽带信息网络.

等的压缩,不同点表现为以下两个方面:

a)一条表项可能会包含很多字段,而其中的一些字段对某种数据流来说是多余的。例如,在 OpenFlow 1.3[4] 中,一条表项可能同时包含 TCP 与 UDP 协议的源/目的地址,实际上,TCP 和 UDP 协议是两种不同的传输层协议,不可能同时存在于同一个数据流中。

b)表项的每个字段都可能成为流表的掩码位。在传统的 IP 路由表中,广泛采用通配符匹配,且 IP 协议较长的前缀匹配策略决定了 IP 地址的掩码位必须是连续出现在地址的末尾。但是,由于 OpenFlow 中任意一个字段都可能成为掩码位,所以掩码位可能出现在表项的任何位置并且不必是连续的。

上述内容表明 OpenFlow 流表压缩有别于先前的工作,针对上述问题,本文的贡献有以下几个方面:

a)提出一种 OpenFlow 字段结构分析模型。OpenFlow 1.3 定义了 40 个匹配域字段,然而字段数量的大幅度增加并不意味着表项数量的大幅度增加。事实上,通过分析字段间的某些特殊关系,并建立相应的模型,可以作用于压缩工作。

b)提出一种针对 TCAM 的预压缩模型。根据字段间的关系,提出一种映射机制来压缩 TCAM 中某些匹配域字段的长度,进而减小了 TCAM 中表项的存储开销。并且,这种压缩机制允许协议中增加新字段,这样就保证了机制的可扩展性。

c)进行仿真实验,并给出详细的性能分析。本文在 OpenFlow 测试网络中获取流表,并执行相应的压缩算法,结果表明所提方法的压缩率可达 60%。

1　相关工作

OpenFlow 流表匹配与传统报文分类具有一定的相似性。OpenFlow 在数据平面的灵活性很大程度上依赖于它所支持的匹配域字段种类。然而,匹配域字段数量的增加使得流表规模变大,给设备的存储和控制器的维护带来很大压力。

随着 IP 网络的大规模部署,表压缩已经成为一个热点研究课题。在 IP 网络中,报文分类也是旨在处理报文头部的多个字段,这与 OpenFlow 类似。现有关于报文分类的研究有助于 OpenFlow 网络中流表压缩的研究。RFC[5] (recursive flow classification)旨在提高报文分类的查询速度并获得高吞吐量,但是高存储开销与低的表更新速率给表的规模带来了限制。FRFC[6] 提高了 RFC 的表更新速率,但是存储空间开销依旧较大。文献[7]通过简化报文分类中的搜索树,可以降低存储开销,但是复杂的树型结构限制了更新速率。HyperCuts[8] 主要关注低维字段,当字段维度增加时,就会面临存储空间和查询效率等问题。

在传统的路由表与接入控制表(ACL)中,一些匹配策略与 OpenFlow 中相似。文献[9]研究了 ACL 优化问题,通过优化不同规则之间的价差与包含情况来减少总体 ACL 规则数量。ACL 压缩器[10] 阐明了一种高压缩率模型,通过分治方法可以实现近 50.22% 的压缩率。这种分解与重组的处理方法非常适于多字段压缩,但是这种方法不能适用于 OpenFlow 对任意位置掩码的支持。

由于 OpenFlow 匹配域支持任意位置掩码,其查询过程非常依赖 TCAM,这给 OpenFlow 流表的规模带来了很大的限制。Curtis 等人[11] 指出了 OpenFlow 流表与传统路由表的区别,并提出 DevoFlow 模型来减小流表规模,但是所提算法扩展性很差。DIFANE[12] 通过将规则集分散到不同的交换机中来减小单个交换机中流表规模,并通过重定向流量到中间交换机来确保表项规则的正确性。然而,在实际网络中,规则是动态变化的,所以频繁的规则变化会给 DIFANE 网络中的控制器带来额外的压力。Kannan 等人[3] 提出一种压缩 TCAM 的模型,并详细讨论了 OpenFlow 对 TCAM 的依赖与 TCAM 的缺点。在模型中,通过给流添加标签来替代整个表项的匹配,可以减小流表的规模,但是在出口交换机中添加标签的过程依旧需要匹配整个 OpenFlow 表项。LightFlow[13] 分析了流表中通配符匹配与精确匹配的关系,基于该关系提出一种基于 GPU 的并行算法。但是,LightFlow 并不适用于 OpenFlow 中匹配域合并。H-SOFT[14] 提出一种字段拆分算法来降低存储空间,该方法是通过减少子表中表项数量实现的。但是,这种拆分处理不支持流表中的任意掩码位,所以会破坏 OpenFlow 细粒度的控制能力。

OpenFlow 也会包含范围匹配字段,比如源端口和目的端口值。这些字段引发的范围扩张问题会给 TCAM 存储空间带来很大压力。DIPRE[15] 与 LIC[16] 方案利用每个 TCAM 表项中未使用的比特位来缓解范围扩张问题,并实现很好的性能,但是依赖大量额外的比特位。因此,压缩 OpenFlow 表项位宽可以帮助缓解范围扩张问题。

2　数学模型

本章提出一种压缩的数学模型,并在 OpenFlow 1.3 上执行该流表压缩算法。由于流表结构在压缩模型中起到关键作用,所以首先分析流表结构。

流表压缩的目的是压缩流表规模的同时保持表项功能的完整性。大体上,流表存储媒介可以分为 RAM 和 TCAM。引言中已经指出 TCAM 具有高匹配速率并支持任意通配符匹配的优点,以及高耗能与高硬件花费的缺点。相反地,RAM 存储更加经济,但是其查询效率较低,并且不支持任意通配符匹配。本文所提模型就是基于 TCAM 与 RAM 的这种关系,利用有限的硬件资源来最大化设备的流表存储能力。

普遍地,当 TCAM 存储面临压力时,可以使用 RAM 来替代存储。在 RAM 中,多字段流表搜索过程常基于 T 查找树结构,但是该结构不适用于 OpenFlow,因为 OpenFlow 需要支持任意位置掩码。在传统多字段匹配表中,由于通配符不是任意的,所以多余的树节点将被移除,节点总数与表项数量同处一个数量级。但是,随着通配符的使用,多余的节点可能是具有掩码值的节点,所以这些节点会被保留,结果存储空间占用过大的问题依旧无法避免。

为了解决上述问题,本文引入一种基于 RAM 的预压缩模型,并提出相应最优化问题的算法。在该模型中,报文头部首先被拆分成不同的子集,然后送到 RAM 中作压缩处理。在 RAM 中执行预压缩算法处理后,一个具有更小位宽的新报文头部被送往 TCAM 中做最终匹配。由于 RAM 也是一种有限的资源,所以模型中也考虑到 RAM 与 TCAM 的平衡。

流表压缩的基本原理是一种协议中的字段都是为将来扩

展而设计的,如端口类型,16 bit 字段可以容纳 65 536 个类型码,但是实际应用中只分配数百个,并且其中只有数十个经常使用。又如 IP 地址,32 bit 的地址长度定义了总容量为 2^{32} 的地址空间,实际上,一个路由器或交换机只处理其中一小部分 IP 地址,约 10 万个。类似地,当考虑一个设备处理的实际值时,源端口、目的端口和端口类型等字段在字段长度上也有冗余。因此,可以利用头部字段的这种性质来执行流表压缩。

定义 1 正交字段组 P_O,表示两个字段$\langle F_i, F_j\rangle$之间有冲突,且不能同时存在于报文头部,即如果 F_i 存在,则 F_j 必不存在,反之亦然。如〈TCP_SRC, UDP_SRC〉可以用 P_O 表示,且〈TCP_DST, UDP_DST〉也可以用 P_O 表示。

定义 2 演进字段组 P_E,表示两个字段$\langle F_i, F_j\rangle$中 F_j 是 F_i 的升级版本。P_E 展现了演进的特点,即 F_j 具有 F_i 的所有功能,如〈IPv4_SRC, IPv6_SRC〉可以表示为 P_E。P_E 在字段映射中起关键作用,并且有助于处理流头部拆分,下文有详细说明。

定义 3 共存字段组 P_C,表示两个字段$\langle F_i, F_j\rangle$中,当且仅当 F_i 存在时,F_j 存在。P_C 展现了一种依赖关系,所以在字段拆分过程中,P_C 用来提高算法有效性。

定义 4 存储系数 η。在一定程度上减少 TCAM 存储开销是以增加 RAM 消耗为代价的。为了反映这种情况,η 定义为每消耗 1 bit RAM 所减少的 TCAM 消耗。实际上,η 可以帮助算法适用于具有不同大小存储空间的硬件设备。

下面给出了一种字段间关系的矩阵。关系矩阵 Π 是基于 OpenFlow 1.3 所必须的 13 个字段(元组)间关系得出的,从中可看出,某些字段拥有不止一种类型的关系。这种情况下,根据下文列出的压缩率对关系优先级进行排序,即 $P_E > P_O > P_C$。在真实的部署环境中,Π 可能会改变,如当 P_E 数量增加时。表 1 给出了模型中主要变量的含义。

$$\Pi = \begin{bmatrix}
P_C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & P_C & P_C & P_C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & P_C & P_C & P_C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & P_C & P_C & P_C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & P_C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & P_C & P_C & P_E & P_O & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & P_C & P_C & P_O & P_E & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & P_E & P_O & P_C & P_C & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & P_O & P_E & P_C & P_C & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_C & P_C & P_O & P_O\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_C & P_C & P_O & P_O\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_O & P_O & P_C & P_C\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_O & P_O & P_C & P_C
\end{bmatrix}$$

表 1 模型中主要变量的含义

变量	含义	变量	含义
E	所有流表项集合	N_i	子集 i 中字段数量($1\leqslant i\leqslant k$)
F	模型中不同字段的集合	M_i	子集 i 中表项数量($1\leqslant i\leqslant k$)
k	子集数量	l_i	字段 i 的位宽($1\leqslant i\leqslant k$)
M	E 中表项总数	l'_{mn}	压缩后 P_o 位宽
N	F 中表项总数	w_i	RAM 中子集 i 位宽($1\leqslant i\leqslant k$)
e_m	流表项 $m(1\leqslant m\leqslant M)$	w'_i	子集 i 搜索输出的位宽($1\leqslant i\leqslant k$)
f_n	字段 $n(1\leqslant n\leqslant N)$		

2.1 字段拆分

如上文所述,几乎所有的协议在字段长度上都有冗余。一方面,对于未来协议的扩展来说冗余是必不可少的,而从时间角度上看,新版本协议可能会引入新字段,从用户角度看,新用户的加入可能会需要使用更多的地址;另一方面,由于每个字段都有自己的冗余,所以一组字段的整体冗余相当大,浪费了宝贵的存储空间。因此,可以给两个或更多的字段分配一些冗余,这样整体冗余度减少的同时还能给字段组预留足够的空间来扩展应用范围。

在 RAM 中执行压缩算法时,不同字段应该被不同的方法处理。因此,字段拆分在减少 RAM 空间中起到关键作用,这样不仅减少子集中表项数目,而且通过并行处理可以提高预压缩速率。假定 $E = \{e_m \mid m = 1, 2, \cdots, M\}$ 是 OpenFlow 交换机流表中的 M 个表项的规则集,$F = \{f_n \mid n = 1, 2, \cdots, N\}$ 表示流表中所有 N 个匹配字段的集合。

根据上文定义的 P_O、P_C,将 F 拆分成 k 个子集,其中 k 的值通过下文等式计算,且给每个子集建立一个子表。对于每个字段,是否将其分配给一个子集受限于系数 η,η 可以平衡 TCAM 空间减少与 RAM 空间的损耗。在第 i 个子集中,字段的数量是 N_i,子集可以表示为 $F_i = \{f_n \mid n = 1, 2, \cdots, N_i\}$,其中 $F_i \subseteq F$,且 $\forall i, j, i \neq j$ 满足 $F_i \cap F_j = \varnothing$。$N_i$ 满足:

$$\sum_{i=1}^{k} N_i = N \tag{1}$$

在每个子集中,由于其只包含子集中字段的信息,所以表项数量小于 M。设每个子集中表项数量为 M_i。

2.2 域间字段合并

域间字段合并应用于 P_O 与 P_C。在每个子集中,如果存在 $f_m, f_n \in F_i$ 使$\langle f_m, f_n\rangle$是 P_O 或者 P_C,可以对其执行相应的压缩处理。设 l_m、l_n 分别是 f_m 与 f_n 的位长,在压缩前,给这两个字段分配的存储空间为$(l_m + l_n)$位。实际上,f_m 和 f_n 使用的值的数量远小于 $2^{(l_m + l_n)}$,如前所述。因此,可以给它们分配一个新的长度为 l'_{mn} 的字段,该长度不仅可以容纳 f_m 与 f_n 所有可能值,而且为将来的扩展预留足够的空间。结果是,这两个字段可以用一个字段来代替,所以减少了总的存储空间,其中 P_O 的压缩率 r_{Po} 可以用式(2)表示。

$$r_{Po} = \frac{\sum_{i=1}^{M_i} l'_{mn}}{\sum_{\text{unique}(f_m)} l_m + \sum_{\text{unique}(f_n)} l_n} \tag{2}$$

式(3)给出存储系数的等式,其中 $\text{unique}(f_m)$ 是子表中字段 f_m 可能值的数量。

$$\eta_{Po} = \frac{\sum_{i=1}^{M} (l_m + l_n - l'_{mn})}{\sum_{\text{unique}(f_m)} l_m + \sum_{\text{unique}(f_n)} l_n + \sum_{j=1}^{M_i} l'_{mn}} \tag{3}$$

P_O 和 P_C 的压缩不会影响报文头部最终的匹配结果,原因如下。在 TCAM 存储的表项结构会根据压缩后的表项格式进行重构,如两个字段$\langle f_m, f_n\rangle$被合并为一个字段。由于这种操作主要会影响 P_O 的压缩,可以将相应情况分为以下两类:

a)在两个正交字段 f_m 和 f_n 中,一条表项只有一个确定的字段值,另一个字段为掩码。在这种情况下,TCAM 将存储 P_O 中合并字段的匹配值。现在假定这条表项中字段 f_m 的值为 A,此时 f_n 是掩码位(反之分析方法相同)。实际上,一个报文头部可能会包含字段 f_m 或者 f_n(正交字段不会同时存在)。对于 f_m、f_n 不同的字段值,RAM 中搜索结果是不一样的,所以只有值是 A 的报文头部才能匹配这条表项。因此,f_m 与 f_n 在合并后不会出现冲突。

b)压缩前 f_m 与 f_n 都是掩码。这种情况下,该表项在压缩

后,TCAM 中合并字段 P_O 的值存储为掩码。因此,这种情况下压缩操作不会改变 TCAM 中匹配结果。

2.3　字段映射

OpenFlow 1.3 匹配域字段间存在另一种特殊关系,字段 f_n 是 f_m 的升级版本,如 IPv6 的源、目的地址与 IPv4 的源、目的地址之间。

在 IPv6 中,用 128 bit 来表示源地址与目的地址,这样可以给主机预留足够多的地址空间。实际上,IPv6 几乎包含了 IPv4 在网络层的所有功能,还包括地址空间的扩展以及特殊应用的扩展。因此,IPv6 可以执行 IPv4 的所有功能。由于提倡从 IPv4 到 IPv6 的平稳过渡,所以 IPv6 与 IPv4 会共存于网络中。

在所有从 IPv4 平稳过渡到 IPv6 的技术中,协议转换技术是最流行的一个。RFC[6052][17] 详细说明了从 IPv4 到 IPv6 的地址转换技术。图 1 为地址转换表。

PL	0----32----40----48----56----64----72----80----88----96----104----
32	prefix　v4(32)　u　suffix
40	prefix　v4(24)　u　(8)　suffix
48	prefix　v4(16)　u　(16)　suffix
56	prefix　(8)　u　v4(24)　suffix
64	prefix　u　v4(32)　suffix
96	prefix　v4(32)

图1　地址转换表

在 TCAM 中,只关注前缀的长度。在地址从 IPv4 转换到 IPv6 时,表项中原来 IPv4 字段的掩码位在 TCAM 中依旧是掩码位。RFC[6052]提倡六种不同的前缀长度,从中为本文模型选取一个合适的前缀长度。在 IPv4 转换成的 IPv6 地址中,流表项的新前缀结合了 IPv4 地址中的原始前缀与新添加的前缀。字段映射中的压缩率为

$$r_{P_E}=\frac{\sum_{\text{unique}(f_n)+\text{unique}(f_m)} l_n}{\sum_{\text{unique}(f_m)\times\text{unique}(f_n)}(l_m+l_n)} \tag{4}$$

由于映射处理只涉及分配操作,所以不额外消耗 RAM 资源,故 $\eta_{P_E}=\infty$。

2.4　域内字段压缩

对于其他字段,依旧采用基于 RAM 的搜索算法来减小字段长度,如 48 bit 的源端口和目的端口字段也能被压缩。在这种方式下,可以采用哈希算法来加速压缩,所以应当认真处理冲突以确保每个输入值都有唯一的输出值。通常用 $w_i(1\leqslant i\leqslant k)$ 定义 RAM 中子集 i 的位长,其中 w_i 是原始字段长度与压缩后字段长度的总和,且用 w'_i 来表示要送往 TCAM 中的 RAM 中预处理后的字段长度。通过字段拆分,流表中一个字段的重复值将被合并为子集流表的一个值,所以可以减少表项数目。由于总的表项数量可以用上文提到的 M 表示,所以用 M_i 表示子集 i 的表项数目。因此,子集 i 的压缩率 r_i 用式(5)表示,存储系数 η 用式(6)表示。

$$r_i=\frac{M_i\times w_i}{M\times l_i}(1\leqslant i\leqslant k) \tag{5}$$

$$\eta_i=\frac{w_i-w'_i}{w_i}(1\leqslant i\leqslant k) \tag{6}$$

2.5　算法整体分析

实际上,通过设置 η_0 的值可以使每个子集的压缩变得可控。$f(\eta_i)$的值表示如果该字段需要被压缩且实际上也被处理,见式(7)。所提模型的整体存储压缩率见式(8)。

$$f(\eta_i)=\begin{cases}1 & \eta_i\leqslant\eta_0\\ 0 & \eta_i>\eta_0\end{cases}\qquad(1\leqslant i\leqslant k) \tag{7}$$

$$r(\eta_0)=\frac{M\times\sum_{i=1}^{k}w'_i\times f(\eta_i)+\sum_{i=1}^{k}M_i\times w_i\times f(\eta_i)+M\times\sum_{i=1}^{k}(w_i-w'_i)\times(1-f(\eta_i))}{M\times\sum_{n=1}^{N}l_n} \tag{8}$$

在式(8)中:分子的值表示该模型的存储空间,包括 TCAM 和 RAM。$\sum_{i=1}^{k}M_i\times w_i\times f(\eta_i)$ 和 $M\times\sum_{i=1}^{k}w'_i\times f(\eta_i)$ 表示在预压缩处理中 RAM 和 TCAM 的存储空间,而 $M\times\sum_{i=1}^{k}(w_i-w'_i)\times(1-f(\eta_i))$ 表示未进行预压缩处理的 TCAM 存储空间。由于本文的目的是降低 TCAM 存储开销,所以最优化问题就变为最大化地减小 TCAM 开销,同时应该最小化 r_{TCAM} 并获得最好的压缩性能 ρ,目标函数见式(9)~(11)。

$$\max\rho(\eta_0)=1-r_{\text{TCAM}}(\eta_0)=1-\frac{\sum_{i=1}^{k}w'_i\times f(\eta_i)+\sum_{i=1}^{k}(w_i-w'_i)\times(1-f(\eta_i))}{\sum_{i=1}^{N}l_i} \tag{9}$$

$$\text{s.t.}\quad \eta_0>0 \tag{10}$$

$$w'_i<l_i<w_i \text{ and } 1\leqslant i\leqslant k \tag{11}$$

3　算法分析

本章主要给出执行所提压缩模型的相应算法。整个算法主要包含三个部分,即产生子集、子集预压缩和头部字段重组。算法 1~3 给出了具体的处理过程。

算法 1　依据关系矩阵 Π 划分子集

输入:F,Π,η,$f(\eta_i)$

输出:字段的子集,η_i

```
begin
precompression_sort(F,Π,η)//根据 η 值降序排列 Π 中关系对
  while F is not NULL
      R = max_extract(F,η)//取 F 中最大 η 值的关系对
      η = pre_compress(R)
      if(f(η_R))
          create a sub-set for R
          remove f_m and f_n of R from F
      else
      end if
  end while
  for all field f_i ∈ F
      if(f(η_i))
          create a sub-set for f_i
          calculate η_i
      end if
  end for
  set_sort (sub-sets,F,η_i)//根据 F 中字段位置来排列所有子集
end
```

算法 1 根据关系矩阵 Π 中 P_O、P_E 和 P_C 将 F 中所有字段拆分成不同的子集,对不同类型的字段组执行相应预压缩算法。根据存储系数 η 来产生子集,且用户可以根据设备情况

调节存储系数临界值 η_0 来控制 FICO 的复杂度。函数 set_sort (sub-sets,F)可以根据其内部字段顺序调整每个子集的顺序,这个处理确保了算法 3 遵循头部字段输入顺序进行重组处理。

算法 2　子集中流表压缩

输入:字段的子集,原始流表 E

输出:压缩后的子集流表

```
begin
for each sub-set
    create a sub-flow table
    for i = 1 to M//E = {e_m | 1≤m≤M} 在 2.1 节中定义
    (hash_based_search,linear_based_search,1) = type_choose(f_j)
        if(hash_based_search)
            insert_table(table[j],hash(f_ij))//f_ij 是 e_i 中字段 f_j 的值
        else if(linear_based_search)
            insert_node(search_tree[j],f_ij)
        end if
    end for
    assign each sub-flow table to a RAM space
end for
end
```

算法 2 可以根据子集调整原始流表产生相应的子集流表,在调整过程中,每个表项的字段值作为节点插入树中或哈希表中。对于像协议数字这样的字段,值大都是连续的,所以这种类型字段的搜索过程是基于二进制搜索,其搜索时间复杂度为 $O(\log_2(w'_i))$,其中 w'_i 是 RAM 中相应字段的位宽。

算法 3　P_O 的压缩算法

输入:原始头部字段,子集流表的搜索结果

输出:要送往 TCAM 中的重组后的头部字段

```
begin
initialize re-composed header field
for i = 1 to N//N 是字段的数量
    if f_i is in a sub-set[j]
      if f_i is not in a relationship pair
          f'_i = search(sub-set[j],f_i) //获取 f_i 压缩后的值
      else if pair of f_i is not searched
          f'_i = search(sub-set[j],f_i)
      else if f_i is a field to be mapped
          f'_i = map(f_i)
      end if
    end if
  entail f'_i to re-composed header field
end for
send re-composed header field to TCAM
end
```

算法 3 反映了一个报文头部的匹配过程。在设备中,每个子集流表的搜索过程是并行的,这样确保了该模型的整体有效性。函数 search(sub-set[j],f_i)整合了不同类型子集流表中的搜索过程。算法 3 的时间复杂度为 $O(N)$,其中 N 是字段数量。

在模型中,算法 1 遍历了关系矩阵 Π 并决定字段如何拆分,该过程是压缩处理的前提。注意到算法 1 只需要被执行一次,所以改变流表的规模不会影响算法 1。算法 2 遍历每个表项的每个字段,然后将一个完整的流拆分成不同的子集。对于每个字段的压缩处理,这些操作可以处理这些字段的原始和被压缩后的值。因此,算法 2 存储复杂度为 $O(MW)$,其中 M 是表项数量,W 是每条表项的位宽。算法 3 将所有不同子集流表的压缩结果合并成一个完整的流表。对于每个子集流表的搜索过程,由于对不同子集流表的操作是并行的,所以其时间复杂度是 $O(\log_2(\max(w'_i)))$,其中 $\max(w'_i)$ 是所有字段长度的最大值。重组过程的时间复杂度是 $O(N)$,因此,FICO 整体存储复杂度是 $O(MW)$,且时间复杂度是($O(\log_2(\max(w'_i)))$ + $O(N)$)。

4　仿真与分析

由于 OpenFlow 还没有广泛部署于网络中,所以很难在大规模 OpenFlow 网络中获得真实数据对模型进行评估。本文实验中的流表来源于国家宽带网络与应用工程技术研究中心的测试网络,总体上可以反映实际 OpenFlow 网络中的流表状况。FICO 中算法在 MATLAB 中进行仿真,采用 Core i7-4790 CPU 和 8 GB 内存。实验结果表明处理每百个表项的时间大约为 1 s。

4.1　FICO 中 TCAM 开销的减小

FICO 旨在通过预压缩机制来减少 TCAM 开销,该机制减小了在 RAM 预处理中表项的大小。考虑到设备的实际情况,FICO 可以在不同的应用环境下进行自适应。图 2 展示了性能指数 ρ、不同流表规模以及不同存储系数 η 之间的关系,该图是流的拆分数 k 为 9 的情况。从图 2 中可以看出,随着存储系数 η 的减小,压缩率 ρ 逐渐增大,而流表规模增大时,压缩率变化缓慢。例如,当流表规模从 5 000 增加到 25 000 时,压缩率 ρ 保持在 0.6 附近。这个结果有效地证明了 FICO 模型增加了 TCAM 存储开销减小的稳定性,这在实际中是一个很重要的特性。

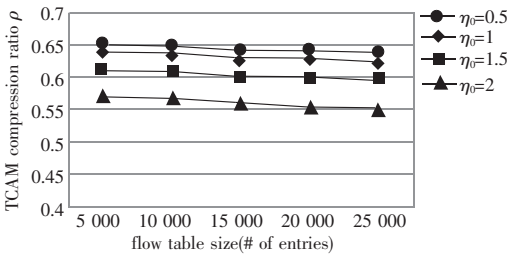

图 2　FICO 的压缩率图

4.2　FICO 与 H-SOFT、Optimal Solution 功能完整性对比

OpenFlow 一个很重要的优点是其对数据流细粒度的控制能力。根据 OpenFlow 白皮书,OpenFlow 支持表项中字段的任意组合,这可以满足很多特殊用户的需求。然而,现在网络中的路由表常常只关注路由功能的一个或多个字段。因此,像 H-SOFT 这样仅从传统网络到 OpenFlow 进行版本转换,会遗漏很多可能的 OpenFlow 字段组合,这些不存在于传统网络中。

在 H-SOFT 中,字段被分成许多字段组,匹配处理由其中某个字段组完成。即使仔细辨别字段间的关系如冲突、共存等,组划分也只能在一定程度上避免功能完整性的破坏,H-SOFT 这么多的组划分难以支撑 OpenFlow 中大量多样化的字段在实际使用中的未知组合。仅仅将传统路由表转换成 OpenFlow 流表掩盖了字段组合的多样性,即 H-SOFT 中的分治策略会破坏 OpenFlow 功能完整性。

H-SOFT 是一种启发式的压缩方法,该方法只将涉及拆分或合并操作的流表中表项在子流表内部进行分配。而文献[14]中提到的 Optimal Solution 在压缩存储空间方面的思路

是,当流表经过拆分或合并操作,会将流表中所有表项都在子流表间进行分配调整,所以该方法字段间组合机会略大于 H-SOFT,因此,该方法对于功能完整性的破坏程度小于H-SOFT。

在 FICO 中收集的 OpenFlow 流表中,匹配域字段的多样化组合大致反映了 OpenFlow 的灵活性。在这种情况下,H-SOFT 和 Optimal Solution 对功能完整性的破坏就暴露出来了。在对 H-SOFT、Optimal Solution 进行仿真时,出现功能性错误情况的数量随着流表的增加出现明显增长,Optimal Solution 的错误率低于 H-SOFT,而 FICO 在实验中无功能性错误。图 3 给出了 FICO 与 H-SOFT、Optimal Solution 中表项功能正确率对比情况。当拆分子集数量 k 变大时,表项功能正确率明显下降,所以对 H-SOFT 中 $k>5$ 的情况不再仿真,其错误率太高。

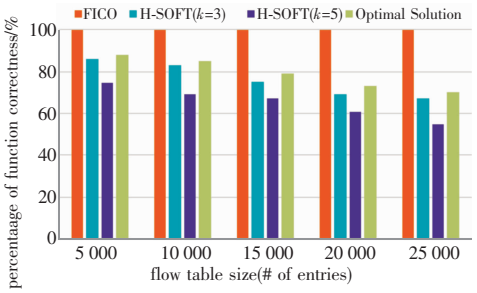

图 3　不同压缩机制间功能完整性对比

4.3　FICO 与 H-SOFT、Optimal Solution 压缩率对比

H-SOFT 提出一种基于分治原理的流表压缩算法,通过划分子表,数据流会选择某个子表执行匹配操作。然而,当需要完全发挥 OpenFlow 细粒度的控制能力时,流表中任何位置都可能出现通配符,这将会严重影响 H-SOFT 的性能。分治原理应用的一个前提是流表中匹配字段类型的分类,但是 OpenFlow 流表项匹配域字段的分布随机性大,很难对其进行分类。Optimal Solution 思路与 H-SOFT 类似,但由于字段间组合机会略大于 H-SOFT,所以压缩率大于 H-SOFT。在 FICO 中,由于不同子表中的预压缩结果需要重组成一个完整的头部字段,所以 OpenFlow 对任意通配符的支持不受影响。便于对比,并与 H-SOFT、Optimal Solution 的测量指标保持一致性,所以对比 FICO 中 TCAM 和 RAM 的整体开销。图 4 给出了在 H-SOFT 临界值 n 为 10 000 时三种机制的性能对比。图 4 表明当流表规模较小时,FICO 的性能更好;当流表规模增大时,FICO 与 H-SOFT、Optimal Solution 性能相当,但是此时 H-SOFT、Optimal Solution 以破坏 OpenFlow 功能完整性为代价。

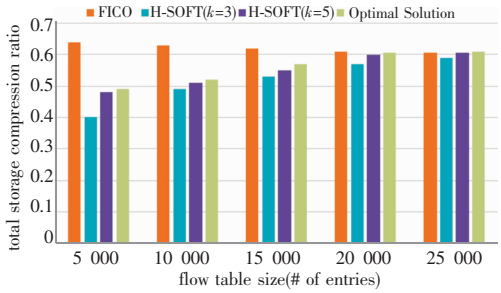

图 4　不同机制间压缩率对比

5　结束语

OpenFlow 流表需要更大的 TCAM 存储空间,本文基于对匹配域不同字段间关系的分析提出一种有效的针对 OpenFlow 交换机的 TCAM 空间节省模型。通过执行包括域间字段压缩、字段映射和域内字段压缩在内的预压缩处理,FICO 有效地减小了 TCAM 中约 60% 的流表规模,因此缓解了 TCAM 中表项存储压力。FICO 可以实现每秒运行上千条表项并且保持较高的部署能力,其存储复杂度为 $O(MW)$,时间复杂度为 $(O(\log_2(\max(w'_i)))+O(N))$。

参考文献:

[1] Montag J. Software defined networking[J]. Network,2014,11(2):1-2.

[2] McKeown N, Anderson T, Balakrishnan H, *et al*. OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review,2008,38(2):69-74.

[3] Kannan K, Banerjee S. Compact TCAM: flow entry compaction in TCAM for power aware SDN[J]. IEEE/ACM Trans on Networking,2012,20(2):448-500.

[4] OpenFlow 1.3.0 specification[EB/OL]. [2012-06-25]. https://www.opennetworking.org/images/stories/downloads/specification/openflowspec-v1.3.0.pdf.

[5] 曹婕,陈兵. 一种内存优化的 RFC 包分类算法 Merge_RFC[J]. 小型微型计算机系统,2012,33(4):865-868.

[6] Pak W, Bahk S. FRFC: fast table building algorithm for recursive flow classification[J]. IEEE Communications Letters,2010,14(12):1182-1184.

[7] Lim H, Lee S, Swartzlander E E. A new hierarchical packet classification algorithm[J]. Computer Networks,2012,56(13):3010-3022.

[8] Jiang Weirong, Prasanna V K. Energy-efficient multi-pipeline architecture for terabit packet classification[C]//Proc of Global Telecommunications Conference. Piscataway, NJ: IEEE Press, 2009: 6270-6275.

[9] Zeng Kuangyi, Yang Jiahai. Towards the optimization of access control list[J]. Journal of Software,2007,18(4):978-986.

[10] Liu A X, Torng E, Meiners C R. Compressing network access control lists[J]. IEEE Trans on Parallel & Distributed Systems,2011,22(12):1969-1977.

[11] Curtis A R, Mogul J C, Tourrilhes J, *et al*. Devoflow: scaling flow management for high-performance networks[J]. ACM SIGCOMM,2011,41(4):254-265.

[12] Yu Minlan, Rexford J, Freedman M J, *et al*. Scalable flow-based networking with DIFANE[J]. ACM SIGCOMM Computer Communication Review,2010,40(4):351-362.

[13] Matsumoto N, Hayashi M. LightFlow: speeding up GPU-based flow switching and facilitating maintenance of flow table[C]//Proc of the 13th IEEE International Conference on High Performance Switching and Routing. Piscataway, NJ: IEEE Press,2012:76-81.

[14] Ge Jingguo, Chen Zhi, Wu Yulei, *et al*. H-SOFT: a heuristic storage space optimisation algorithm for flow table of OpenFlow[J]. Concurrency & Computation Practice & Experience, 2015, 27(13): 3497-3509.

[15] Lakshminarayanan K, Rangarajan A, Venkatachary S. Algorithms for advanced packet classification with ternary CAMs[J]. ACM SIGCOMM Computer Communication Review, 2005, 35(4): 193-204.

[16] Bremler-Barr A, Hay D, Hendler D, *et al*. Layered interval codes for tcam-based classification[C]//Proc of International Conference on Computer Communications. Rio de Janeiro, Brazil: IEEE Press, 2009: 1305-1313.

[17] IETF. RFC 6052, IPV6 Addressing of IPV4/IPV6 Trans Lators[S]. 2010.