ADAFT：SDN大规模流表的适应性深度聚合存储架构

熊兵[1]，袁月[1]，赵锦元[2]，赵宝康[3]，何施茗[1]，张锦[1]
（1.长沙理工大学计算机与通信工程学院，湖南 长沙 410114；2.长沙师范学院信息科学与工程学院，湖南 长沙 410199；
3.国防科技大学计算机学院，湖南 长沙 410073）

摘　要：为解决软件定义网络（SDN）数据平面中的三态内容可寻址存储器（TCAM）资源紧张问题，提出了一种基于内容表项树的SDN流表深度聚合方法，进而构建一种SDN大规模流表的适应性深度聚合存储架构ADAFT。该架构放宽了聚合表项之间的汉明距离要求，构建内容表项树聚合动作集不同的流表项，显著提高了流表聚合程度。设计了一种TCAM装载率感知的内容表项树动态限高机制，以降低流表查找开销。同时，提出了一种TCAM装载率感知的表项聚合适应性选择策略，以均衡流表聚合程度和查找开销。实验结果表明，ADAFT架构的流表压缩率明显高于现有方法，最高可达65.74%。

关键词：软件定义网络；SDN大规模流表；内容表项树；适应性深度聚合；TCAM装载率感知
中图分类号：TP393
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2024059

ADAFT：an storage architecture of large-scale SDN flow tables based on adaptive deep aggregations

XIONG Bing[1], YUAN Yue[1], ZHAO Jinyuan[2], ZHAO Baokang[3], HE Shiming[1], ZHANG Jin[1]
1. School of Computer Science and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China
2. School of Information Science and Engineering, Changsha Normal University, Changsha 410199, China
3. School of Computer Science, National University of Defense Technology, Changsha 410073, China

Abstract: To solve the problem of resource shortage of ternary content addressable memory (TCAM) in the data plane of software defined network (SDN), a deep flow table aggregation method was proposed based on content entry trees, and a storage architecture of large-scale SDN flow tables named ADAFT was established. The architecture relaxed the Hamming distance requirement between ag-gregated flow entries, and a content entry tree was constructed to aggregate flow entries with different action sets, for significantly en-hancing the aggregation degree of flow tables. Then a dynamic limitation mechanism was designed for the height of content entry trees based on the awareness of TCAM load ratio, to minimize the lookup overhead of aggregated flow tables. Meanwhile, an adaptive selec-tion strategy of flow entry aggregation was presented in the light of TCAM load ratio, to strike a balance between the aggregation degree and lookup overhead of flow tables. Experimental results indicate that the ADAFT architecture achieves much higher flow table compression ratios up to 65.74% than existing methods.

Keywords: software defined network, large-scale SDN flow table, content entry tree, adaptive deep aggregation, TCAM load ratio awareness

---

收稿日期：2023–10–12；**修回日期：**2024–02–19
通信作者：赵宝康，bkzhao@nudt.edu.cn
基金项目：国家自然科学基金资助项目（No.U22B2005, No.61972412, No.62272062）；国家重点研发计划基金资助项目（No.2022YFB2901204）；湖南省自然科学基金资助项目（No.2023JJ30053, No.2021JJ30456）；湖南省教育厅基金资助项目（No.22A0232, No.23A0735, No.22B0300）；湖南省研究生科研创新基金资助项目（No.CX20230913）
Foundation Items: The National Natural Science Foundation of China (No.U22B2005, No.61972412, No.62272062), The National Key Research and Development Program of China (No. 2022YFB2901204), The Natural Science Foundation of Hunan Province (No. 2023JJ30053, No. 2021JJ30456), Scientific Research Fund of Hunan Provincial Education Department (No. 22A0232, No.23A0735, No.22B0300), The Postgraduate Scientific Research Innovation Project of Hunan Province (No.CX20230913)

0　引言

为了解决传统网络架构的垂直集成、难以管理配置、缺乏可编程性和细粒度控制等问题，软件定义网络（SDN, software defined network）作为一种数控分离和可编程的网络架构应运而生[1]。SDN 将控制逻辑和数据转发相解耦，并通过以 OpenFlow 为代表的南向接口协议灵活定义数据平面的分组转发行为，通过北向接口为上层应用提供统一的编程接口，可显著提升网络的开放性、灵活性和可管控性，已成为未来网络领域中获得普遍认可并进入实际部署的主流技术方向[2]。当前，广域网和数据中心网络已广泛应用 SDN，以简化网络业务部署，实现网络资源的灵活调度与编排[3]。近年来，随着卫星网络的迅速发展，SDN 开始应用于卫星网络，并进一步扩展到空天地一体化网络，为其提供更强的管控能力，并支持网络的快速更新与升级，同时提升其与现有网络之间的兼容性。

SDN 数据平面通常采用三态内容可寻址存储器（TCAM, ternary content addressable memory）存储流表，以实现 SDN 交换设备的快速分组转发[4]。TCAM 支持“0、1、通配符”3 种状态的数据查找，且并行匹配其整个数据集，并能在一个周期内输出匹配结果。然而，TCAM 集成度低、成本高、能耗大，导致其容量有限[5]。随着网络应用的蓬勃发展和 SDN 的大规模部署，网络中的并发流数量越来越多，导致 SDN 交换机中的流表规模不断增大[6]。这使得 TCAM 容量越来越难以满足 SDN 大规模流表的存储需求。当 SDN 流表规模超出 TCAM 容量时，将出现流表溢出问题，导致溢出流表项对应的分组无法得到及时处理，进而对分组转发性能造成明显的破坏性影响。因此，如何有效解决 TCAM 流表资源紧张，是 SDN 实现广泛应用部署和持续演进亟待解决的一个问题。

为使 TCAM 存储更多的流表项，已有部分研究工作通过表项压缩来减少流表项的匹配宽度，以减少流表项占用空间，如紧凑型 TCAM[7]、匹配域裁剪[8]、独立规则集位提取[9]等。然而这类方法大多涉及专门的硬件电路设计，不仅增加部署的成本和开销，还会影响分组匹配和更新的效率。进一步，很多研究者采用逻辑表达式最小化的方法对流表项压缩优化，如块置换[10]、多流表聚合（Agg-ExTable）[11]、高效的流规则约简（EFRR, effective flow-rule-reducing）[12]等。然而这类方法逻辑运算复杂，规则集的原始状态无法保留，会影响交换机的分组匹配处理速度。此外，不少研究者设计流表聚合算法合并类似的流表项，以减少流表项数量，节省 TCAM 存储空间，如 TCAM Razor[13]、Bit Weaving[14]、快速表项聚合（FFTA, fast flow table aggregation）[15-16]等。然而，这些聚合方法只能聚合匹配字段汉明距离为 1 且动作集相同的流表项，导致流表聚合程度不高，而且未考虑流优先级，容易造成分组转发语义错误。

针对上述问题，本文首先依据各种存储介质的访问特性，构建一种 SDN 大规模流表深度聚合存储架构 ADAFT，提高 TCAM 利用率，有效解决 TCAM 资源紧张问题。然后，设计一种 SDN 流表深度聚合方法，通过放宽表项之间的聚合要求，在保证分组转发语义正确性的同时，提高流表聚合程度。同时，设计一种可动态适应 TCAM 容量的内容表项树限高机制，动态限定内容表项树的高度，以保证访问和查找速度。此外，设计一种表项聚合方式的适应性选择策略，根据实时 TCAM 装载率选择不同聚合方式，进一步降低流表查找开销。在此基础上，给出对应的 SDN 流表深度聚合算法，并利用主干网链路流量样本评估验证算法性能。本文的主要贡献总结如下。

1) 基于不同存储介质的访问特性，构建 SDN 大规模流表深度聚合存储架构 ADAFT，将所有流表项的内容字段剥离出来采用二叉树结构单独存储，并设置聚合加速备份表专门负责聚合流表，在解决 TCAM 资源紧张问题的同时，有效加快表项聚合操作，提高 SDN 流表聚合速度。

2) 针对传统流表聚合方法要求流表项匹配字段之间汉明距离为 1 且动作集必须相同的局限性，设计一种 SDN 流表深度聚合方法，通过构建内容表项树聚合动作集不同的流表项，并将聚合表项的匹配字段之间汉明距离仅能为 1 放宽到 2，显著提高聚合程度。

3) 为有效防止内容表项树过高导致流表查找开销过于繁重的问题，通过实时感知 TCAM 装载率，设计一种内容表项树的动态限高机制，根据聚合流表大小与 TCAM 容量的适配程度动态调整内容表项树的高度限制，在确保 TCAM 能容纳聚合流表的同时，尽可能降低内容表项树的高度，以提

高网络分组的流表查找速度。

4) 针对流表聚合过程中不同表项聚合方式产生不同的聚合程度和收益的情况，通过实时感知TCAM装载率，设计一种表项聚合方式的适应性选择策略，首先将TCAM装载率划分为4个层次，并为每个层次设置不同的聚合方式。在聚合过程中，根据其动态选择适合的聚合方式，使得TCAM容纳大规模流表的同时，进一步降低流表查找开销。

1 相关工作

目前，已有部分研究工作通过表项压缩设法减少每条流表项的匹配宽度，以降低单条规则占用的TCAM空间。最初，文献[7]提出紧凑型TCAM方案，引入流标识符的概念来替代流表项15元组，以减少流表查找操作所需匹配的流表项宽度。与此类似，文献[8]提出一种基于匹配域规则集提取的包分类压缩方法，基于对规则集中匹配域的逻辑关系分析，实现匹配域的合并，以减少匹配域的数量，从而节省TCAM存储空间。进一步，文献[17]提出一种高效的SDN流表拆分压缩算法，通过分析流表匹配字段之间的共存和互斥关系，将流表划分出多个规模较小的子流表，进而针对每个字段建立判定条件，对子流表做进一步压缩，以实现SDN流表的高效存储，从而有效节省TCAM存储空间。在此基础上，文献[9]提出了一种基于独立规则集位提取的压缩方法，对合并后的规则集进行独立规则子集分割，将分割后的子集进行可区分的位提取并压缩，进一步缩减TCAM所用空间。上述方案通过减短SDN流表项宽度增加TCAM的存储效率，但是大多都涉及专门的硬件电路设计，不仅增加部署的成本和开销，还会影响分组匹配和更新的效率。

很多研究者采用卡诺图[18]、QM（Quine Mc-Cluskey）算法[19-21]和Espresso算法[22]等逻辑表达式最小化的方法压缩流表。文献[23]首先将原始规则进行逻辑运算后映射到卡诺图，然后将其中相邻且具有相同转发动作指令的区域进行合并，进而应用最小序列覆盖算法压缩流表。针对卡诺图中动作指令分布稀疏从而导致压缩效率下降的问题，文献[10]提出了一种基于块排列的流表压缩方法，通过调整卡诺图中规则的排列顺序提高其相同动作指令的分布密集程度，从而提高压缩率。此外，文献[11]提出了一种多流表聚合方法Agg-ExTable，采用剪枝和QM算法对多个流表进行周期性聚合，以节省TCAM存储空间。进一步，文献[12]提出一种EFRR算法，采用最优路由表构造算法压缩非前缀规则，进而应用改进的QM算法进行二次压缩，从而显著减少SDN交换机中的流规则数量。然而，该类方法的逻辑运算复杂，且无法保留规则集的原始状态，导致交换机中未压缩规则的分组匹配速度下降。

不少研究者在保持转发语义不变的情况下，设计流表聚合算法合并相似的流表项，以减少流表项数量，节省TCAM存储空间。文献[13]提出一种多字段前缀规则聚合算法TCAM Razor，将多维规则分割成多个一维规则列表，通过防火墙决策图算法减少TCAM中的冗余规则。然而，SDN流表项的匹配字段通常为非前缀形式，无法直接应用此类算法。对此，文献[24]将非前缀形式的流表项通过比特分解转换为前缀形式，并构建前缀树合并表项，但聚合开销随待分解的通配符数量呈线性增长。类似地，文献[14]提出一种三元非前缀表项聚合算法Bit Weaving，通过比特列交换对流表项进行正交分组，从而将非前缀流表项转换为前缀形式，再利用传统前缀聚合算法进行聚合。然而，该方法操作复杂，聚合速度慢。对此，文献[15-16]提出一种FFTA方案，利用二叉搜索树直接合并流表项，聚合速度快，但牺牲了一定的聚合程度。为提高聚合程度，文献[25]提出一种基于聚类算法的泛型匹配框架GenMatcher，根据流表项之间的相似度进行分组，以增加聚合深度，提高流表聚合效果。然而，上述方案要求待聚合的流表项动作集一致，导致流表聚合程度不高，而且未考虑流优先级，容易造成分组转发语义错误。

2 SDN大规模流表聚合存储架构ADAFT

针对TCAM容量有限难以容纳SDN大规模流表的问题，目前一类典型的应对方案是采用流表聚合方法压缩SDN流表规模，使TCAM能存储所有流表项。然而，目前的流表聚合方法只能聚合汉明距离为1的流表项，且要求动作集相同，聚合要求相当严格，聚合程度极为有限。对此，本文聚合汉明距离为1且动作集不同的流表项，并为聚合表项构建包含动作集的内容表项树，以保证分组转发语

义的正确性。在此基础上，将聚合表项之间的汉明距离要求从仅能为 1 放宽到 2，以增大表项聚合机会，提高流表聚合程度。此外，将内容字段从流表项中剥离出来，采用静态随机存取存储器（SRAM, static random-access memory）单独存储，从而释放大量的 TCAM 存储空间，用于存储流表项的匹配字段。依据上述设计思路，建立如图 1 所示的 SDN 大规模流表聚合存储架构 ADAFT，以有效解决 TCAM 资源紧张问题，并实现 SDN 流表的快速查找。

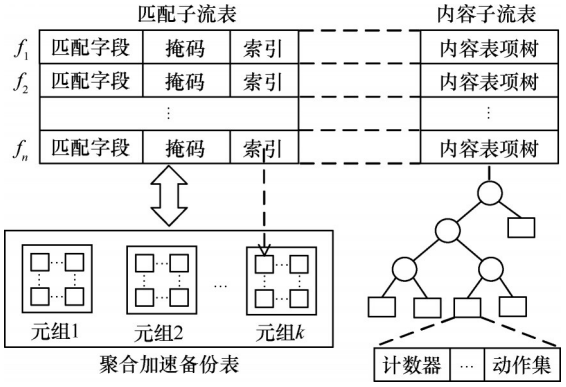


图 1　SDN 大规模流表聚合存储架构 ADAFT

ADAFT 架构由匹配子流表、聚合加速备份表和内容子流表组成。匹配子流表采用 TCAM 存储 SDN 流表聚合后所有表项的匹配字段，以实现数据包的快速查找。其中，每条流表项的匹配字段用于标识流，掩码字段则用于标记匹配字段中通配符的位置，索引字段指明聚合加速备份表中的对应流表项。聚合加速备份表将匹配子流表中的所有流表项按照掩码划分成若干元组进行存储，以便快速进行表项聚合。内容子流表存储 SDN 流表聚合后所有表项的内容表项树，与匹配子流表的所有表项一一对应，通常采用 SRAM 存储以实现快速访问与查找。每棵内容表项树存储被聚合表项的内容字段（含动作集），以保证分组转发语义的正确性。

3　适应性的 SDN 流表深度聚合方法

针对 ADAFT 架构，本节首先通过构建内容表项树提出一种 SDN 流表深度聚合方法，进而根据 TCAM 实时装载率设计内容表项树的动态限高机制，最后制定一种表项聚合方式的适应性选择策略。

3.1　基于内容表项树的流表深度聚合方法

针对传统流表聚合方法对聚合表项要求苛刻的问题，本文为 ADAFT 架构设计了一种基于内容表项树的 SDN 流表深度聚合方法。流表深度聚合原理如图 2 所示。该方法将 SDN 原始流表按照掩码划分为若干元组，同一个元组中的所有流表项具有相同的掩码，即其通配符位置完全一致。对于每个元组，根据任意 2 条流表项之间的不同精确位数量即汉明距离进行合并。对于被合并的表项，首先取消其动作集相同的限制要求，然后将其汉明距离要求为 1 放宽到 2，以加大聚合机会。在此基础上，将每次合并后的表项根据其掩码放置到对应的元组中继续聚合。依次循环操作，直到无法聚合。通过上述方法，可以有效提高元组聚合程度，进而显著提升流表聚合效果。

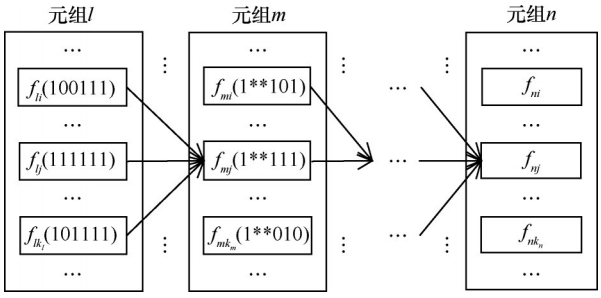


图 2　流表深度聚合原理

为支持聚合动作集不同的流表项，本文为每条聚合表项构建一棵如图 3 所示的内容表项树。在树中，每个叶子节点存储原始表项的内容字段（含动作集），非叶子节点记录表项聚合过程中的合并比特位置。对于聚合表项匹配的网络分组，根据非叶子节点记录的比特位置，从树根开始回溯对应表项的聚合过程，最终锁定的叶子节点即对应原始表项的内容字段。此时，可根据其中的动作集正确转发分组。为了保证分组转发效率，必须控制内容表项树的查找开销。考虑内容表项树的查找长度不会超出其高度，本文为内容表项树的高度设置一个上限值。在表项聚合过程中，当聚合表项的内容表项树高达到该上限值时，则停止聚合。通过构建内容表项树，有效保证了网络分组转发的语义正确性和高效性。

内容表项树的构建过程如下。若元组中存在 2 条可合并的流表项，即其匹配字段之间的汉明距离为 1 或 2，则执行表项合并过程，新建内容表项树。若其匹配字段之间的汉明距离为 1，则首先新建一个根节点，记录其匹配字段之间比特不相同的位

置，然后将2条流表项的内容表项（树）分别置为对应的孩子节点；若其匹配字段之间的汉明距离为2，则新建根节点记录其匹配字段之间比特不相同的一个位置，然后将比特不相同的另一个位置分别记录至根节点的左右孩子节点，同时将2条待合并流表项的内容表项（树）分别置为对应的子孙节点，从而形成新的内容表项树。此外，若该元组中还有其他流表项被包含在合并后的表项中，则将其内容表项（树）置为对应的叶子节点。在流表聚合过程中，不断重复上述操作，直至内容表项树的高度达到上限值或无法聚合，最终构建出聚合表项的内容表项树。

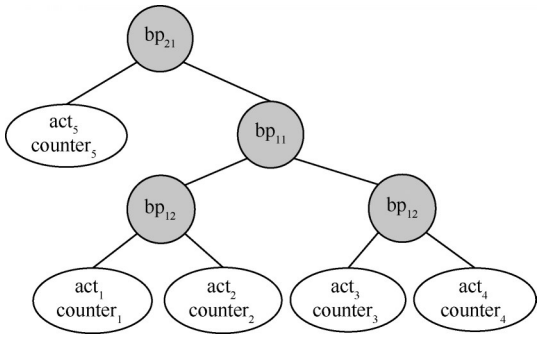


图3　内容表项树

3.2　TCAM装载率感知的内容表项树动态限高机制

在深度聚合方法中，随着流表聚合过程的不断进行，内容表项树将不断增高，容易导致网络分组的流表查找开销过于繁重。对此，本文设计了一种TCAM装载率感知的内容表项树动态限高机制，如图4所示，以权衡流表聚合程度和分组查找开销。该机制为所有聚合表项的内容表项树高统一设置一个阈值，并根据聚合流表大小与TCAM容量的实时适配程度动态调整该阈值。本文定义TCAM装载率为聚合流表大小与TCAM容量的比值。当TCAM装载率较小时，无须大量聚合流表，树高阈值可设置较小，以有效控制流表查找开销；当TCAM装载率较大时，树高阈值应设置较大，以加大流表聚合程度，从而使TCAM能够容纳聚合后的流表。该机制根据实时TCAM装载率适应性调节内容表项树高阈值，使TCAM容纳大规模流表的同时，尽可能降低流表查找开销。

在该机制中，内容表项树高阈值H的设置主要取决于TCAM装载率r，且满足如下条件。

1) H和r具有正相关关系。

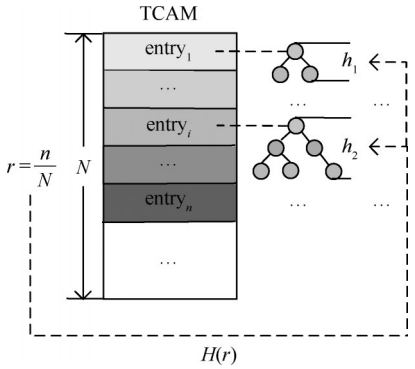


图4　TCAM装载率感知的内容表项树动态限高机制

2) 当r趋于0时，H应趋于1。

3) 当r趋于1时，理论上H应趋于+∞。

对此，本文设计一种内容表项树高阈值函数$H(r)$，如式(1)所示。

$$H(r)=\left[1-\ln(1-r)\right]^{k} \tag{1}$$

其中，k为一个大于1的常量，表示该函数的变化曲率，可根据实际应用场景下的TCAM容量限制、流表大小等因素进行设置。随着TCAM装载率的不断上升，内容表项树高阈值将快速增大，以加大流表聚合程度，从而使TCAM能够容纳聚合流表。

3.3　TCAM装载率感知的表项聚合适应性选择策略

上述聚合方法构建内容表项树聚合动作集不同的流表项，在实现流表深度的同时，也增大了网络分组的流表查找开销。对于新插入的某条流表项，若有与其汉明距离为1的流表项，则2条流表项合并构建的内容表项树增高1层，同时会使汉明距离为2的流表项丧失聚合机会；若有与其汉明距离为2的流表项，则合并构建的内容表项树增高2层。此时，若合并后的表项还包含其他流表项，则可直接汇入其中，实现3条流表项的聚合；否则，2条流表项合并为一条。总之，不同的聚合方式产生了不同的聚合程度，带来了对应的聚合收益。同时，不同的聚合方式也使得内容表项树增加了不同的层数，付出了对应的聚合代价。考虑聚合收益与代价之比，本文提出了一种TCAM装载率感知的表项聚合适应性选择策略，如图5所示。该策略将TCAM装载率划分为4层，并为每层设置不同的聚合方式。随着TCAM装载率层的不断增高，逐步放宽聚合要求，以加大聚合程度。

第1层：TCAM装载率满足$r<r_1$。新插入的流表项只与汉明距离为1且动作集相同的流表项合

并。此情况下，不构建内容表项树，从而不会产生额外的流表查找开销。

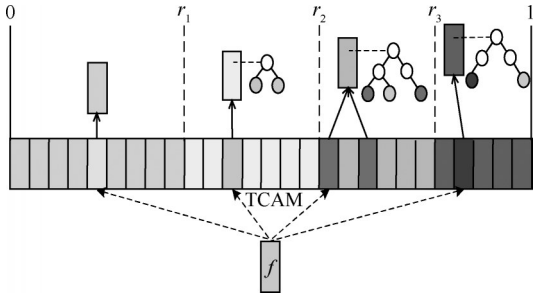


图5　TCAM 装载率感知的表项聚合适应性选择策略

第 2 层：TCAM 装载率满足 $r_1 \leqslant r < r_2$。新插入的流表项若满足第 1 层的聚合要求，则执行对应的聚合方式。否则，合并与其汉明距离为 1 且动作集不同的流表项。此情况下，2 条流表项合并为一条，对应的内容表项树增高 1 层。

第 3 层：TCAM 装载率满足 $r_2 \leqslant r < r_3$。新插入的流表项若满足第 1 层的聚合要求，则执行对应的聚合方式。若存在与其汉明距离为 2 的流表项可合并，且有其他流表项包含于合并表项，则将 3 条流表项合并为一条，对应的内容表项树增高 2 层。否则，执行第 2 层的聚合方式。

第 4 层：TCAM 装载率满足大于 $r \geqslant r_3$。新插入的流表项若满足第 3 层的聚合要求，执行对应的聚合方式。否则，合并与其汉明距离为 2 的流表项。此情况下，2 条流表项合并为一条，对应的内容表项树增高 2 层。

4　SDN 聚合流表操作

本节给出了 ADAFT 架构下各种 SDN 聚合流表操作算法的伪代码实现。

4.1　流表查找

算法 1 给出了 SDN 聚合流表查找算法 FlowTableLookup 的伪代码实现。当 SDN 交换机收到一个网络分组 p 后，首先解析其各层协议首部提取匹配字段 mf，进而查找匹配子流表。若查找成功，则根据返回的匹配表项索引值 index，定位并查找内容子流表中对应的内容表项树 cet（第 1~4 行）。具体来说，根据匹配字段 mf，从树根开始按照非叶子节点记录的合并比特位置 mbp 逐步往下查找（第 5~12 行）。若成功找到一个叶子节点 node，则依据其中的动作集转发处理分组，并更新其中的计数器与时间戳等内容字段（第 13~17 行）。若查找失败，则表明该分组属于一条新流。此时，SDN 交换机将该分组的信息打包成 packet-in 消息上交给控制器，以请求下发对应的流规则（第 19~21 行）。

算法 1　流表查找算法 FlowTableLookup

输入　到达的网络分组 p

输出　流表查找结果

bool FlowTableLookup (Packet p)

1) mf ← ParsePacket(p); //解析分组，提取匹配字段

2) index ← MatchTableLookup(mf); //查找匹配子流表

3) if index is valid, then //匹配子流表查找成功

4)　node ← ContentTable[index].root ; //查找内容表项树

5)　while node.lchild != NULL or node.rchild != NULL, then //根据匹配字段，从根节点开始逐步往下查找

6)　　mbp ← ReadMergedBitPosition(node); //找到非叶子节点记录的合并比特位置

7)　　if mf[mbp]==1, then

8)　　　node = node.rchild;

9)　　else

10)　　　node = node.lchild;

11)　　end if

12)　end while

13)　if node is valid, then //查找叶子节点成功

14)　　ExecuteActions (node.action); //依据动作集转发分组

15)　　UpdateContentEntry (node); //更新计数器与时间戳等

16)　　return true;

17)　end if

18) end if

19) msg ← CreatePacketInMessage(p); //将该分组的信息打包成 packet-in 消息

20) SendMessageToController(msg);//将 packet-in 消息上交给控制器

21) return false.

4.2　流表插入

算法 2 给出了 SDN 流表插入算法 FlowTableInsert 的伪代码实现。当 SDN 控制器下发一条流规则

f时，首先根据其掩码在聚合加速备份表中定位对应的元组T（第1行）。若定位成功，则执行流表项聚合操作（第3行）。若聚合成功，则生成一条新的流表项f'，并根据其掩码放入对应的元组中继续合并（第4~9行）。依次循环操作，直至不能合并。最后，将最终的流表项插入SDN流表，即将其匹配字段插入匹配子流表和聚合加速备份表，内容字段插入内容子流表（第11~14行）。

算法2 流表插入算法FlowTableInsert

输入 控制器下发的流规则f

void FlowTableInsert (FlowRule f)

1) T ← GetTuple(f.mask); //定位对应元组

2) while T != NULL, then

3) f' ← FlowEntryAggregate(T, f); //执行流表项聚合操作

4) if f' != f, then //放入对应的新元组继续合并

5) f ← f';

6) T ← GetTuple(f.mask);

7) else

8) break;

9) end if

10) end while

11) index ← GetEmptyEntry(); //获取空表项

12) InsertMatchTable(index, f.mf); //在匹配子流表中插入匹配字段

13) InsertContentTable(index, f.cf); //在内容子流表中插入内容字段

14) InsertBackupTable(T, f.mf); //在聚合加速备份表中插入匹配字段

4.3 表项聚合

算法3给出了表项聚合算法FlowEntryAggregate的伪代码实现。对于待合并的流表项f，根据其掩码定位对应的元组T，进而查找可与其合并的流表项。首先查找与其汉明距离为1且动作集相同的流表项。若查找成功，则直接合并（第1~3行）。若查找失败且TCAM装载率$r \geqslant r_2$，则查找与其汉明距离为2的流表项，且有其他流表项包含于合并表项。若查找成功且其内容表项树高小于阈值H–1，则合并3条流表项，并新增2层构建内容表项树（第8~21行）。若查找失败且TCAM装载率$r \geqslant r_1$，则查找与其汉明距离为1且动作集不同的流表项。若查找成功且其内容表项树高小于阈值H，则合并2条流表项，并新增一层构建内容表项树（第22~31行）。若查找失败且TCAM装载率$r \geqslant r_3$，则查找与其汉明距离为2的流表项。若查找成功且其内容表项树高小于阈值H–1，则合并2条流表项，并新增2层构建内容表项树（第32~41行）。当流表项成功合并为一条新表项f'时，删除被合并的流表项。若流表项合并失败，则返回原始表项f（第42行）。

算法3 表项聚合算法FlowEntryAggregate

输入 待合并的表项f和其所在的元组T

输出 聚合表项f'

FlowEntry FlowEntryAggregate (T, f)

1) for i ← 0, T.size –1 do

2) if HammingDistance(T[i], f) == 1 && T[i].actions == f.actions, then //查找与其汉明距离为1且动作集相同的流表项

3) f' ← MergeEntryHD1(T[i], f); //合并流表项

4) DeleteEntry(T[i]); //删除被合并的流表项

5) return f';

6) end if

7) end for

8) if $r \geqslant r_2$, then

9) for i ← 0, T.size –1 do

10) if HammingDistance(T[i], f) == 2 && T[i].depth < H–1, then //查找与其汉明距离为2且内容表项树高小于阈值H–1的流表项

11) f' ← MergeEntryHD2(T[i], f); //合并流表项

12) for j ← 0, T.size –1 do //合并被包含的流表项

13) if T[j] ⊂f' && T[j] != T[i], then //构建内容表项树

14) f'.cet ← BuildNewTree(f, T[i], T[j]);

15) DeleteEntry(T[i], T[j]); //删除被合并的流表项

16) return f';

17) end if

18) end for

19)　end if
20)　end for
21) end if
22) if $r \geqslant r_1$, then
23)　for $i \leftarrow 0$, T.size –1 do
24)　if HammingDistance($T[i], f$) == 1 && $T[i]$.depth < H, then //查找与其汉明距离为 1 且内容表项树高小于阈值 H 的流表项
25)　$f' \leftarrow$ MergeEntryHD1($T[i], f$); //合并流表项
26)　f'.cet ← BuildNewTree($T[i], f$); //构建内容表项树
27)　DeleteEntry($T[i]$); //删除被合并的流表项
28)　return f';
29)　end if
30)　end for
31) end if
32) if $r \geqslant r_3$, then
33)　for $i \leftarrow 0$, T.size –1 do
34)　if HammingDistance($T[i], f$) == 2 && $T[i]$.depth < H–1, then //查找与其汉明距离为 2 且内容表项树高小于阈值 H–1 的流表项
35)　$f' \leftarrow$ MergeEntryHD2($T[i], f$); //合并流表项
36)　f'.cet ← BuildNewTree($T[i], f$); //构建内容表项树
37)　DeleteEntry($T[i]$); //删除被合并的流表项
38)　return f';
39)　end if
40)　end for
41) end if
42) return f.

4.4　流表删除

算法 4 给出了 SDN 流表删除算法 FlowTableDelete 的伪代码实现。当 SDN 交换机需要删除一条流表项 f 时，首先根据其匹配字段查找匹配子流表（第 1 行）。若成功找到一条完全匹配的流表项，则直接删除（第 3~4 行）。若成功找到一条没有内容表项树的匹配表项，则更新匹配子流表和聚合加速备份表中的对应表项，即将其匹配字段中的不同比特由通配符变为与待删除表项相反的精确位（第 5~7 行）。若成功找到一条有内容表项树的匹配表项，则根据待删除表项 f 的匹配字段查找其内容表项树。具体来说，从树根开始，按照非叶子节点记录的合并比特位置依次向下查找（第 8~17 行）。若成功找到一个叶子节点 node，则删除该节点，并不断向上回溯删除所有无孩子节点的父节点 father（第 18~28 行）。否则，向控制器发送 error 消息，报告流规则删除失败结果（第 32~34 行）。

算法 4　流表删除算法 FlowTableDelete
输入　待删除的流表项 f
输出　流表项删除成功与否

void FlowTableDelete (FlowEntry f)
1) index ← MatchTableLookup(f.mf); //根据匹配字段查找匹配子流表
2) if index is valid, then
3)　if f.mf == MatchTable[index].mf, then //流表项完全匹配
4)　DeleteEntry(index); //删除流表项
5)　else if ContentTable[index].root == NULL, then //流表项没有内容表项树
6)　UpdateMatchEntry(); //更新匹配子流表
7)　UpdateBackupEntry(); //更新聚合加速备份表
8)　else
9)　node ← ContentTable[index].root; //获取内容表项树
10)　while node.lchild != NULL or node.rchild != NULL, then //从根节点开始，根据匹配字段依次向下查找节点
11)　mbp←ReadMergedBitPosition(node); //读取合并比特位置
12)　if f.mf[mbp]==1, then
13)　node = node.rchild;
14)　else
15)　node = node.lchild;
16)　end if
17)　end while
18)　if node is valid, then
19)　while node != NULL, then //删除叶子节点，并向上回溯删除所有无孩子节点的父

节点

20)　　father ← FindFatherNode (node);

21)　　DelectTreeNode(node);

22)　　UpdateMatchEntry();

23)　　UpdateBackupEntry();

24)　　if father.lchild == NULL && father.rchild == NULL, then

25)　　node ← father;

26)　　end if

27)　　end while

28)　　end if

29)　end if

30)　return true;

31) else

32)　msg ← CreateErrorMessage(); //创建error消息

33)　SendMessageToController(msg); //向控制器发送error消息

34)　return false;

35) end if

5　实验

本节首先给出了SDN流表聚合方法的性能指标和测试数据集，然后实验对比了不同聚合方法的压缩率，最后测试分析了本文所提ADAFT架构的平均查找长度。

5.1　实验方法

实验采用C++编程实现不同SDN流表聚合方法，以验证本文所提流表深度聚合方法的优越性。压缩率是衡量流表聚合方法的主要性能指标之一，通常定义为流表聚合后表项数量减少的百分比。假设原始表项数量为N_{O}，流表聚合后的表项数量为N_{A}，则压缩率CR可表示为

$$\mathrm{CR}=\frac{N_{\mathrm{O}}-N_{\mathrm{A}}}{N_{\mathrm{O}}}\times 100\% \tag{2}$$

另外，由于本文构建了内容表项树，并建立了动态限高机制，因此引入平均查找长度，以全面评估本文所提流表深度聚合方法的性能。

本文选取江苏省计算机网络技术重点实验室发布的2个高速网络流量样本（TRACE20120922和TRACE20130122）作为测试数据集。这2个样本按照1∶4的比例采集于中国教育科研网江苏省边界10 Gbit/s主干链路，网络分组数量均为15 420 235，采集日期分别为2012年9月22日和2013年1月22日，采集时间分别为85 s和106 s。为简化起见，SDN流表的匹配字段选取传统五元组（源IP地址、目的IP地址、源端口、目的端口、协议类型）。依据上述设置，可统计出网络流量样本中的流数量如图6所示。

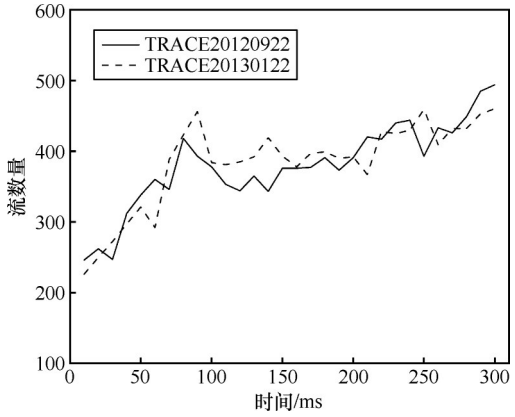


图6　网络流量样本中的流数量

5.2　压缩率

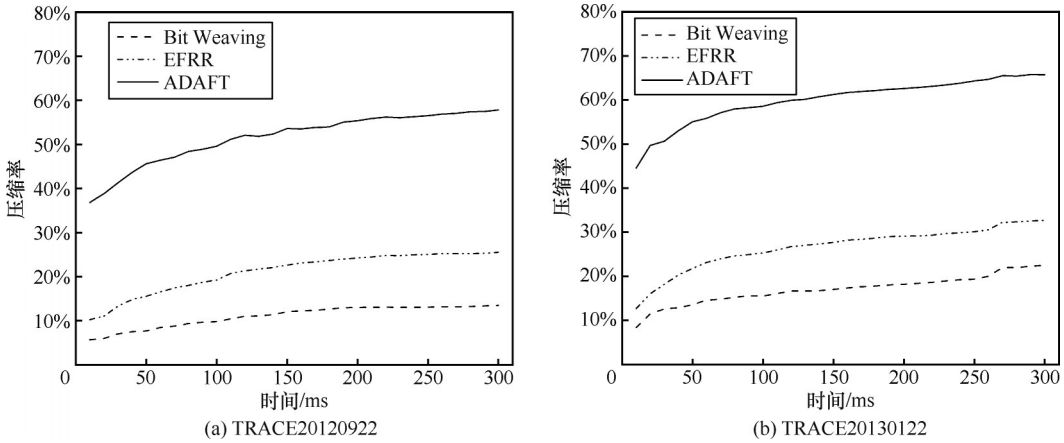
采用上述网络流量样本，实验首先对比了本文所提ADAFT、Bit Weaving方法[14]和EFRR方法[12]的压缩率。对于本文所提ADAFT，首先不限制其内容表项树的高度，以评估其最大聚合效果。从网络流量样本中依次读取分组，执行上述3种SDN流表聚合方法，可得到不同流表聚合方法的压缩率如图7所示。

由图7可以看出，无论哪个流量样本，本文所提ADAFT的压缩率始终明显高于另外2个聚合方法。对于样本TRACE20120922和TRACE 20130122，本文所提ADAFT的压缩率最高可达65.74%，平均分别为51.57%和59.94%，比Bit Weaving方法分别高40.53%和42.91%，比EFRR方法分别高30.48%和33.32%。这主要归因于本文所提ADAFT将表项聚合要求从匹配字段之间汉明距离为1扩展到2，同时可聚合动作集不同的流表项。而Bit Weaving和EFRR方法只能聚合匹配字段汉明距离为1且动作集相同的流表项，聚合程度较低。

当本文所提ADAFT动态限制树高时，根据TCAM实时装载率所在区间选择不同聚合策略，以适应性调整流表聚合程度。TCAM容量设置为8 000，TCAM装载率阈值r_1、r_2和r_3分别设置为

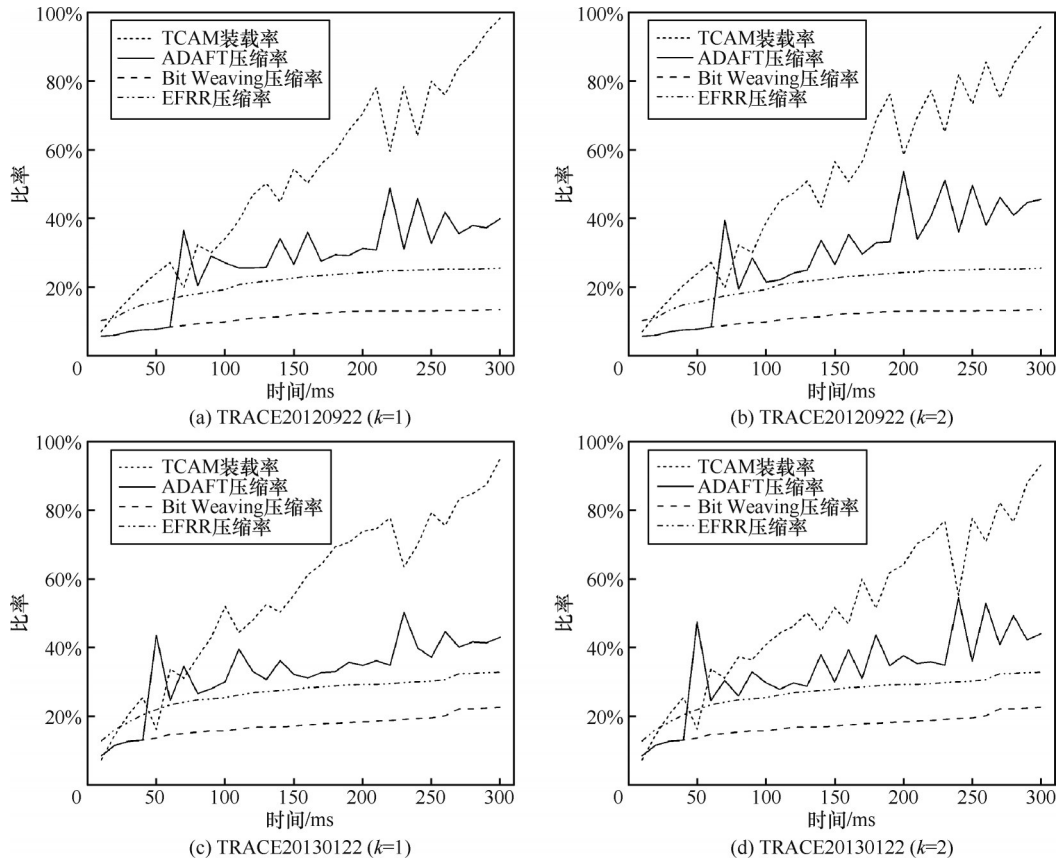
25%、50% 和 75%。将树高阈值计算参数 k 分别设置为 1 和 2，分别从 2 个样本中依次读取分组，执行本文所提 ADAFT，得到对应的 TCAM 装载率和流表压缩率如图 8 所示。

从图 8 中可以看出，无论哪个流量样本和参数 k 取值，在 40 ms 后，本文所提 ADAFT 的压缩率均高于现有方法，且随着 TCAM 装载率的逐渐上升而稳步增大。以图 8(a)为例，即样本 TRACE20220922 在 k =1 的情况，当 TCAM 装载率处于阈值区间 $[0, r_1]$时，本文所提 ADAFT 的压缩率与 Bit Weaving 完全相同；当 TCAM 装载率处于阈值区间$[r_1, r_2]$时，本文所提 ADAFT 的压缩率超出现有方法，最高可达 36.53%；当 TCAM 装载率处于阈值区间 $[r_2, r_3]$时，本文所提 ADAFT 的压缩率继续增大，稳定大于现有方法，最高可达 35.99%；当 TCAM 装载率处于阈值区间$[r_3, 1]$时，本文所提 ADAFT 始



(a) TRACE20120922　　(b) TRACE20130122

图 7　不同流表聚合方法的压缩率



(a) TRACE20120922 (k=1)　　(b) TRACE20120922 (k=2)

(c) TRACE20130122 (k=1)　　(d) TRACE20130122 (k=2)

图 8　TCAM 装载率和流表压缩率

终保持最高的压缩率，最高可达48.85%，比Bit Weaving和EFRR方法分别高35.77%和24.05%。这主要归因于本文所提内容表项树的动态限高机制和表项聚合方式的适应性选择策略。随着TCAM装载率的增大，树高阈值随之增大，同时表项聚合条件逐步放宽，因此流表聚合程度稳步增大。总之，流表聚合程度将根据TCAM实时装载率进行适应性调整，尽可能使聚合流表大小与TCAM容量相适配。从图8中还可以看出，流表压缩率还与k值密切相关。

图9展示了不同TCAM装载率下的树高阈值。结合图8和图9可以看出，无论哪个流量样本，k值越大，内容表项树高的阈值增长越快，流表压缩率的上升幅度越大。这主要是因为式(1)给出的内容表项树高阈值函数，参数k可以调整树高阈值随TCAM装载率上升的增长速度。当k=1时，树高阈值的增速较缓慢，基本上不超过5；当k=2时，树高阈值的增速较快，尤其是TCAM装载率较高时。因此，通过适当设置k值，可有效控制树高阈值的增长速度。

5.3　平均查找长度

采用上述同样的参数配置，从网络流量样本中依次读取分组，执行本文所提ADAFT，可得到对应的平均查找长度如图10所示。结合图8和图10可以看出，随着TCAM装载率的逐渐上升，流表聚合程度不断增大，而平均查找长度随之增大。在前40 ms内，平均查找长度均为1。这是因为在聚合起始阶段，TCAM装载率小于阈值r_1，此时只聚合汉明距离为1且动作集相同的流表项，无须构建内容表项树，因此只需查找TCAM聚合流表，查找长度为1。40 ms过后，平均查找长度开始逐步增大。这是因为TCAM装载率超过阈值r_1后，开始聚合动作集不同的流表项，需要构建内容表项树。同时，内容表项树的高度随TCAM装载率的上升而增高，因而平均查找长度随之增大。此外，从图10还可以看出，k值越大，平均查找长度的增长速度

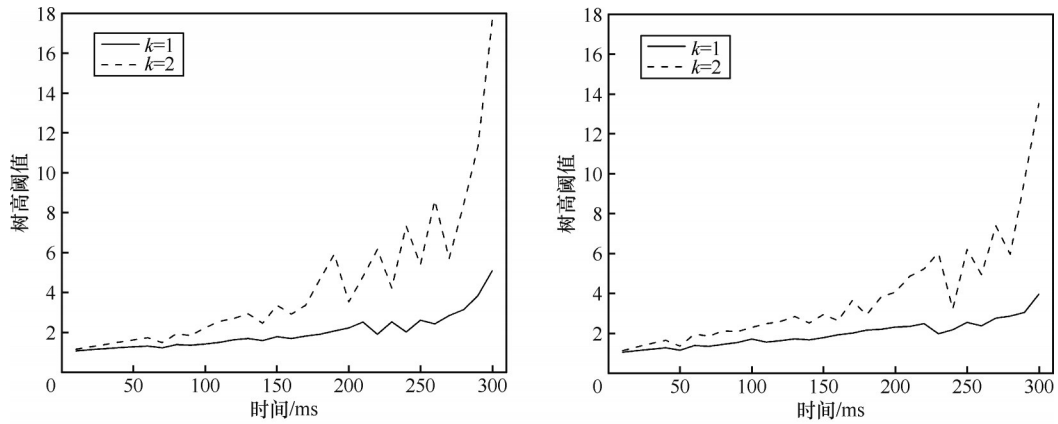


图9　不同TCAM装载率下的树高阈值

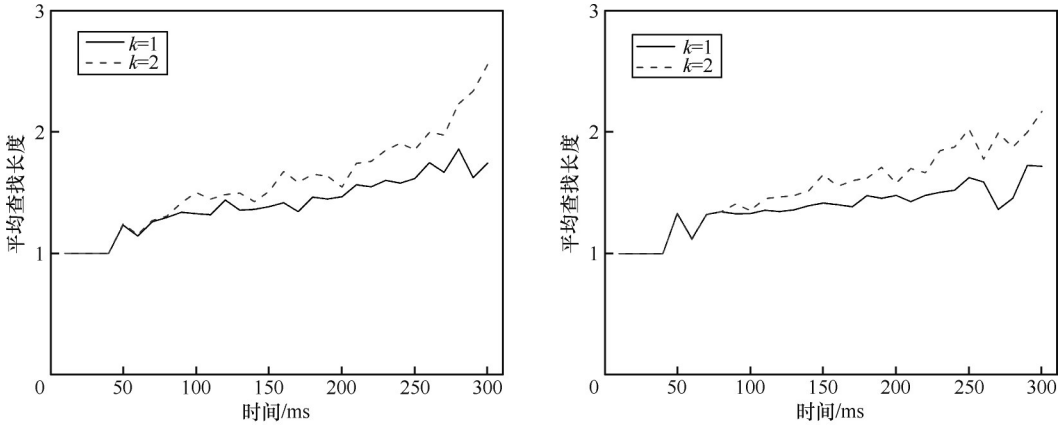


图10　平均查找长度

越快。当k=1时，平均查找长度的增速较缓慢，基本上稳定在2以内。因此，通过适当设置k值，可有效控制聚合流表的平均查找长度，从而保证网络分组的流表查找性能。

6　结束语

针对TCAM容量有限难以满足SDN大规模流表存储需求的问题，本文设计了一种基于内容表项树的SDN流表深度聚合方法，进而构建了一种SDN大规模流表的适应性深度聚合存储架构ADAFT。该架构将表项聚合要求从汉明距离为1放宽到2，并根据合并比特位置构建内容表项树，以聚合动作集不同的流表项，从而显著提高流表聚合程度。在此基础上，给出了一种内容表项树动态限高机制，通过聚合流表大小与TCAM容量的实时适配程度动态限定内容表项树的高度，从而保证分组的访问和查找速度。然后，本文设计了一种TCAM装载率感知的表项聚合适应性选择策略，通过不同的TCAM装载率而选择最适合的聚合方式，从而使TCAM容纳大规模流表的同时，进一步降低流表查找开销。

实验选取CERNET主干链路上采集的真实网络流量样本，评估了本文所提SDN大规模流表存储架构ADAFT的性能。实验结果表明，本文所提ADAFT架构的压缩率显著高于Bit Weaving方法和EFRR方法，可以在取得更好聚合效果的同时，保证网络分组的流表查找性能。

参考文献：

[1] NISAR K, JIMSON E R, HIJAZI M H A, et al. A survey on the architecture, application, and security of software defined networking: challenges and open issues[J]. Internet of Things, 2020, 12(100289): 1-27.

[2] COSTA L C, VIEIRA A B, E SILVA E B, et al. OpenFlow data planes performance evaluation[J]. Performance Evaluation, 2021, 147(102194): 1-23.

[3] SHIRMARZ A, GHAFFARI A. Performance issues and solutions in SDN-based data center: a survey[J]. The Journal of Supercomputing, 2020, 76(10): 7545-7593.

[4] XU S Z, WANG X, YANG G X, et al. Routing optimization for cloud services in SDN-based Internet of Things with TCAM capacity constraint[J]. Journal of Communications and Networks, 2020, 22(2): 145-158.

[5] LI Z Y, HU Y X. PASR: an efficient flow forwarding scheme based on segment routing in software-defined networking[J]. IEEE Access, 2020, 8: 10907-10914.

[6] BABANGIDA I, SOPERI M Z M, MAZNAH B K, et al. Software defined networking flow table management of OpenFlow switches performance and security challenges: a survey[J]. Future Internet, 2020, 12(9): 147.

[7] KANNAN K, BANERJEE S. Compact TCAM: flow entry compaction in TCAM for power aware SDN[C]//International Conference on Distributed Computing and Networking (ICDCN), Berlin: Springer, 2013: 439-444.

[8] 孙鹏浩, 兰巨龙, 陆肖元, 等. 一种基于匹配域裁剪的包分类规则集压缩方法[J]. 电子与信息学报, 2017, 39(5): 1185-1192.
SUN P H, LAN J L, LU X Y, et al. Field-trimming compression model for rule set of packet classification[J]. Journal of Electronics & Information Technology, 2017, 39(5): 1185-1192.

[9] 王孝龙, 刘勤让, 林森杰, 等. 基于独立规则集位提取的包分类压缩方法[J]. 计算机应用, 2018, 38(8): 2375-2380.
WANG X L, LIU Q R, LIN S J, et al. Compression method based on bit extraction of independent rule sets for packet classification[J]. Journal of Computer Applications, 2018, 38(8): 2375-2380.

[10] WEI R H, XU Y, CHAO H J. Block permutations in Boolean space to minimize TCAM for packet classification[C]//Proceedings of IEEE INFOCOM. Piscataway: IEEE Press, 2012: 2561-2565.

[11] WANG C, YOUN H Y. Entry aggregation and early match using hidden Markov model of flow table in SDN[J]. Sensors, 2019, 19(10): 2341.

[12] CHENG M H, HWANG W S, WU Y J, et al. An effective flow-rule-reducing algorithm for flow tables in software-defined networks[C]// Proceedings of the 2020 International Computer Symposium (ICS). Piscataway: IEEE Press, 2020: 25-30.

[13] LIU A X, MEINERS C R, TORNG E. TCAM Razor: a systematic approach towards minimizing packet classifiers in TCAMs[J]. IEEE/ACM Transactions on Networking, 2010, 18(2): 490-500.

[14] MEINERS C R, LIU A X, TORNG E. Bit Weaving: a non-prefix approach to compressing packet classifiers in TCAMs[J]. IEEE/ACM Transactions on Networking, 2012, 20(2): 488-500.

[15] LUO S X, YU H F, LI L M. Fast incremental flow table aggregation in SDN[C]//Proceedings of the 2014 23rd International Conference on Computer Communication and Networks (ICCCN). Piscataway: IEEE Press, 2014: 1-8.

[16] LUO S X, YU H F, LI L M. Practical flow table aggregation in SDN[J]. Computer Networks, 2015, 92: 72-88.

[17] 姜腊林, 张亚南, 熊兵. 一种高效的OpenFlow流表拆分压缩算法[J]. 小型微型计算机系统, 2018, 39(2): 310-314.
JIANG L L, ZHANG Y N, XIONG B. An efficient OpenFlow stream table splitting compression algorithm[J]. Journal of Chinese Computer Systems, 2018, 39(2): 310-314.

[18] RAHMAN M R. Mathematical study for reduction of variables in karnaugh map[C]// Smart Computing and Informatics. Berlin: Springer, 2018: 551-558.

[19] JARA L, GONZÁLEZ A, PÉREZ R. A preliminary study to apply the Quine McCluskey algorithm for fuzzy rule base minimization[C]//Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). Piscataway: IEEE Press, 2020: 1-6.

[20] NUGROHO E D. Development of applications for simplification of Boolean functions using Quine-McCluskey method[J]. Telematika, 2021, 18(1): 27-36.

[21] JOSHI M, SUNORI S K, TEWARI N, et al. Formulation of C++ program for Quine-McCluskey method of Boolean function minimization[C]// International Conference on Machine Learning, Advances in Comput-

ing, Renewable Energy and Communication (MARC). Berlin: Springer, 2022: 341-346.

[22] ASHMOUNI E F, RAMADAN R A, RASHED A A. Espresso for rule mining[J]. Procedia Computer Science, 2014, 32: 596-603.

[23] MCGEER R, YALAGANDULA P. Minimizing rulesets for TCAM implementation[C]//Proceedings of the IEEE INFOCOM. Piscataway: IEEE Press, 2009: 1314-1322.

[24] BRAUN W, MENTH M. Wildcard compression of inter-domain routing tables for OpenFlow-based software-defined networking[C]//Proceedings of the 2014 Third European Workshop on Software Defined Networks. Piscataway: IEEE Press, 2014: 25-30.

[25] WANG P, MCHALE L, GRATZ P V, et al. GenMatcher: a generic clustering-based arbitrary matching framework[J]. ACM Transactions on Architecture and Code Optimization, 2018, 15(4): 51.

[作者简介]

熊兵（1981– ），男，湖南益阳人，博士，长沙理工大学副教授，主要研究方向为未来网络、网络测量、网络安全等。

袁月（1998– ），女，江西萍乡人，长沙理工大学硕士生，主要研究方向为软件定义网络、包分类、流表聚合等。

赵锦元（1980– ），女，湖南邵阳人，博士，长沙师范学院副教授，主要研究方向为未来网络、网络测量、网络建模与优化等。

赵宝康（1981– ），男，湖北天门人，博士，国防科技大学副教授，主要研究方向为网络体系结构与协议、卫星互联网、高性能网络、网络安全等。

何施茗（1986– ），女，湖南永州人，博士，长沙理工大学副教授，主要研究方向为异常检测、矩阵分解、图网络与数据处理、无线网络。

张锦（1979– ），男，河南信阳人，博士，长沙理工大学教授，主要研究方向为人工智能、软件工程。