doi：10. 12068/j. issn. 1005－3026. 2026. 20249055

基于在网计算的高效网络流规则压缩算法

张鹏豪
（山西大学 计算机与信息技术学院，山西 太原 030000）

摘 要：大规模数据处理导致分布式系统流量激增. 针对传统网络流规则压缩方法存在压缩率低、网络处理效率低的问题，提出了一种基于在网计算的流规则压缩算法，该算法的核心在于利用在网计算技术压缩规则以提高网络转发效率，通过流规则去重方法解决范围重叠问题，设计并查压缩方法高效生成表达式规则. 最后将表达式规则卸载到可编程交换机以加速流量转发. 实验结果表明，与传统压缩算法相比，所提算法在网络处理性能上提升40%~60%，最高可实现99. 8%的规则压缩率.

关 键 词：云计算；在网计算；流规则压缩；可编程交换机；表达式流规则
中图分类号：TP 393.02 **文献标志码**：A **文章编号**：1005-3026(2026)03-0030-08

Efficient Network Flow Rule Compression Algorithm Based on In-Network Computing

ZHANG Peng-hao
（School of Computer and Information Technology, Shanxi University, Taiyuan 030000, China. Corresponding author: ZHANG Peng-hao, E-mail: zhangpenghao@sxu.edu.cn）

Abstract: Large-scale data processing has caused a surge in traffic within distributed systems. To address the limitations of traditional network flow rule compression methods, such as low compression rates and poor network processing efficiency, a flow rule compression algorithm based on in-network computing was proposed. The core of the algorithm lies in leveraging the in-network computing technology to compress rules for improving network forwarding efficiency. The problem of overlapping rule ranges through flow rule deduplication was solved; a union-find compression method was designed to efficiently generate expression rules. Finally, the expression rules were unloaded to programmable switches to accelerate traffic forwarding. Experimental results show that compared with traditional compression algorithms, the proposed algorithm improves network processing performance by 40%~60% and achieves a rule compression rate of 99. 8%.

Key words: cloud computing; in-network computing; flow rule compression; programmable switch; expression flow rule

随着云计算的发展，分布式系统的应用范围得到了前所未有的扩展，涵盖了网页排序检索、分布式数据库管理、网络分析以及数据中心存储等多个关键领域. 这些应用场景催生了庞大的网络流量交换需求，对传统交换机构成了严峻挑战. 传统交换机仅支持标准五元组的流规则部署，且更新机制僵化，难以适应分布式系统所带来的爆炸性网络流量传输需求. 软件定义网络（software defined network, SDN）作为一种创新的网络架构，通过将网络的控制面与数据面分离，实现了网络功能的灵活模拟与实现. 但云平台流量的激增对SDN的控制面功能设计及其数据面的内存提出了更为严格的要求. 控制面必须支持更复杂的流量匹配字段，而数据面则需在交换机上部署更多的流规则以确保云平台应用的性能.

在流规则压缩领域，已有研究工作存在一定

收稿日期：2024-11-13
基金项目：国家自然科学基金资助项目（62302281）；山西省基础研究计划资助项目（202303021212016）.
作者简介：张鹏豪（1995—），男，山西吕梁人，山西大学讲师，博士.
通信作者：张鹏豪，E-mail：zhangpenghao@sxu.edu.cn.

的局限性.文献[1-2]优化控制面功能,提高了流规则部署、替换效率,但只能小幅减少流规则数目;文献[3-4]利用决策树模型压缩流规则,难以支持多字段匹配;文献[5]提出增加交换机以扩充存储,但导致网络更加复杂.上述工作通过优化SDN控制面功能、数据面内存或对流规则拆分匹配实现流规则压缩,都存在一定的缺陷.

近年来,可编程交换机等网络硬件可为网络提供计算能力,推动了在网计算(in-network computing,INC)的快速发展.Barefoot公司的P4交换机是业界广泛使用的可编程交换机之一,可提供基于硬件的高速网络计算能力.P4交换机打破传统交换机固定转发数据包的模式,充分解放交换机数据面的编程能力,P4交换机具有灵活的协议解析器和定制的匹配动作转发引擎,可定制多种网络协议;通过P4语言对交换机的解析器、匹配表和寄存器进行定制化编程,并翻译为硬件语言驱动程序处理数据包.此外,P4交换机支持对解析后的数据包进行布尔和算术运算.通过在P4交换机上编译运行定制化程序,交换机数据面接收并解析数据包,提取包头和包内数据,存入相应的寄存器进行运算,对数据包实现分类、过滤等不同功能.P4交换机优化了传统交换机仅能转发数据包的功能,提供了多样化的网络计算功能,还可将端侧性能低的任务卸载到网络侧进行计算,提高系统的整体性能.

鉴于在网计算的优势以及流规则压缩方法的缺陷,本文提出了一种高效的网络流规则压缩算法(flow rules compression with INC),简称为FRC算法.该算法的核心是利用在网计算的优势,将流规则压缩为表达式规则,设计协议将表达式规则卸载至可编程交换机,实现了更高的压缩率和更快的流量转发效率,同时保证了可靠性和正确性.

1　背景研究

1.1　基础定义

流规则R:网络内规则存储在流表中.每个流表T内的规则R定义为一个元组(pri,$\boldsymbol{m}$,a),其中pri是规则的优先级,pri越大,规则R的优先级越高;$\boldsymbol{m}$是匹配向量$(0,1,*)^L$,由长度为L的位序列构成,向量的每个元素表示规则的一个匹配字段,*表示通配符;a是动作值,表示匹配到该规则后交换机执行该动作.

表达式e:表达式e由输入变量、算术运算符和常量组成.例如,表达式e可以写作t×mask+c,其中t是输入变量,mask和c是常量.

表达式规则ExpR:定义表达式流表ET由一组表达式规则ExpR组成.每个表达式规则ExpR定义为一个元组(pri,$\boldsymbol{m}$,e),其中pri和$\boldsymbol{m}$与流规则R的元素含义一致,e为表达式.例如,假设表达式规则ExpR(pri=1,$\boldsymbol{m}$=10*0,e=t&1 000+7),其中pri=1和$\boldsymbol{m}$=10*0分别表示规则的优先级和匹配向量,t&1 000+7是表达式.

1.2　流规则压缩方法

在云计算时代,分布式应用如网页检索、分布式数据库和数据中心等产生的流量规模显著增加,对网络交换机的流规则存储能力提出了挑战.SDN的发展使得部分网络功能得以在服务器端部署,研究工作主要集中在优化SDN控制面和数据面流规则处理过程,以及压缩流规则数目,以解决交换机存储受限问题.DevoFlow[6]通过分离不同规模的网络流量,实现了端侧处理短流和网络侧处理长流,降低了通信频率,但可能导致部分短流的丢弃;文献[7]提出了一种压缩模型,通过将流规则替换为标签,降低了交换机存储需求,但该方法需要在流量被转发前进行二次校验以保证正确性;H-SOFT[8]利用字段分割算法切割流规则,减少了存储占用,但不支持通配符匹配的流规则;文献[9]提出了Razor,通过一维前缀分类器聚合流规则;文献[10]改进了Razor,通过比特交换和比特合并算法支持非前缀流规则合并.

1.3　基于在网计算的流规则压缩

针对流规则压缩工作中存在的压缩率低、压缩效果差以及网络流量处理效率低等问题,本研究提出了一种新型的流规则压缩算法,旨在利用INC技术的优势,实现流规则的高效压缩与卸载,以加速云平台应用的流量转发.在网计算技术通过将部分计算任务卸载到网络设备,如交换机,可以减少端侧处理器的负载,提高分布式架构的整体性能.例如,ATP[11]和SwitchML[12]通过在网计算加速梯度聚合过程,提高了分布式深度学习的训练效率;NetCache[13]通过在网计算过滤多余数据并缓存所需数据,加速了分布式数据库的查询性能;NetSHA[14]利用在网计算过滤搜索过程中的较差结果,提高了分布式系统的检索效率.尽管在网计算在多个领域展现出潜力,但目前尚未有研究利用在网计算实现流规则压缩技术以提升分布式系统的流量处理效率.

FRC算法利用在网计算的优势,通过流规则去重和并查压缩,解决了流规则压缩中的区间重叠与最优压缩问题,并利用可编程交换机重构数据包,将规则卸载至交换机以加速数据包的匹配.

2 算法设计

2.1 FRC算法

FRC是一种基于在网计算加速的流规则压缩算法.首先,FRC使用两种方法(流规则去重、并查压缩)将流规则在服务器端压缩成表达式规则;接着,设计适配协议将表达式规则下发到可编程交换机.图1显示了FRC算法的基本框架,FRC算法包含3个步骤:将流规则R生成BT树结构,通过去重方法,得到互不重叠的流规则,重写为TR规则;将TR规则通过并查压缩方法分成不同的子集,对每个子集的规则通过枚举得到表达式规则ExpR;通过自定义协议将ExpR下发到可编程交换机.

2.2 符号定义

本文规定了一些符号的含义,以便更好地描述FRC算法的实现过程,如表1所示.

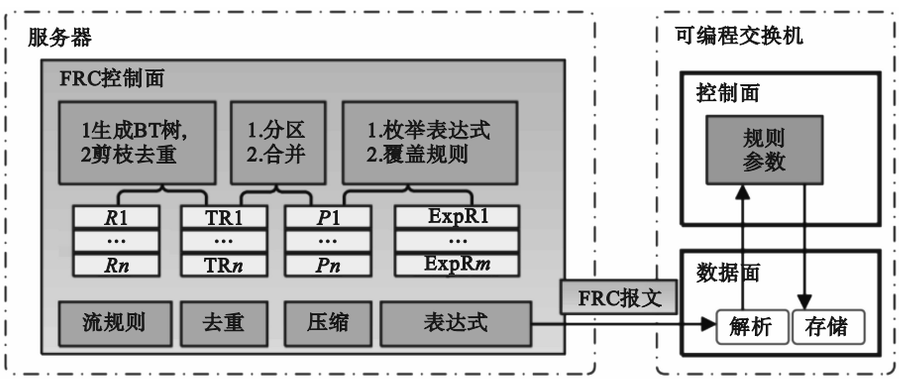

图1 FRC算法框架

Fig. 1 FRC algorithm framework

表1 FRC算法的符号及定义

Table 1 Symbols and definitions of FRC algorithm

FRC算法涉及符号	含义
T	流表,包含n个流规则R
R	流规则,由元组(pri,$\boldsymbol{m}$,a)组成,pri为优先级,$\boldsymbol{m}$为匹配向量,a为动作
BTM	二叉树去重方法
TR	用关键字–掩码定义的流规则(key-mask),由元组(key, mask, a)组成
ET	表达式流表,包含n个表达式规则ExpR
ExpR	表达式规则,由元组(pri,$\boldsymbol{m}$,e)组成,pri和$\boldsymbol{m}$分别代表优先级和匹配向量,e是表达式,$e=t\square\text{mask}\square c$
$\square$	算子,可选择(+,–,<<,>>,∧,∨,⊕),表示加、减、左移、右移、与、或、异或

2.3 流规则去重

FRC算法的第一步是将流规则去重,并解决流规则匹配范围重叠问题.

流规则的匹配向量$\boldsymbol{m}$可表示为$(0,1,*)^L$,定义$\boldsymbol{m}$的长度为L,$\boldsymbol{m}$为由0,1,*构成的位序列,字符*为通配符,可以默认为字符0,即忽略匹配,则$\boldsymbol{m}$可以视为二进制字符串.受二叉树思想的启发,可以将$\boldsymbol{m}$表示成二叉树结构,去重是将二叉树中重复的枝干进行剪枝的过程,得到互不重复的二叉树,再转换回流规则,本文将上述方法称为二叉树去重(BTM)方法.BTM方法包括两步:首先将流规则转换为BT结构;其次实现BT剪枝算法.

算法1 转换流规则为BT结构算法

输入:流规则$R[N]$,BT结构tree$[N]$

输出:N棵二叉树BT,tree$[N]$

定义:节点结构Node,树结构BT,BT的根节点为rt.

1)循环N次,对每个流规则R生成BT结构

2)for t=0;t<N;t++

3) 生成新的二叉树:tree$[t]$=new BT

4) Node rt=tree$[t]$.rt;

5)循环L次,每次生成树结构的一个分支

6) for i=0;i<L;i++

7) if $R[t].\boldsymbol{m}[i]$=0: rt.Left=new Node, rt.mark=1;

8）　if $R[t].\boldsymbol{m}[i]$=1：rt.Right=new Node，rt.mark=2；

9）　if $R[t].\boldsymbol{m}[i]$=*：rt.Left=new Node，rt.mark=0；

10）　rt跳转到新生成的叶子节点

11）　end for

12）　end for

具体来讲，对每个流规则创建一个二叉树的根节点，对应算法的第2至3行；接着遍历流规则的二进制字符串，如果当前遍历的字符是0或*，则为二叉树生成左叶子节点，否则生成右叶子节点.同时为不同节点设置不同标记，如果字符为*时标记为0，代表该节点是通配符.重复遍历字符串，直到生成完整的二叉树结构，对应算法的第5至11行.

接着实现BT剪枝算法，以此得到互不重复的流规则.BT剪枝算法首先通过双重循环遍历所有的二叉树结构，每次采用高优先级的树对低优先级的树进行剪枝.具体剪枝过程为，双重循环遍历流规则$R[i]$和$R[j]$的树结构；如果满足预设的判断条件，则判定存在相同路径；如果未提前终止循环，则在遍历结束后，删除$R[j]$中的重复路径.

例如：有两个流规则$R1$(pri=1，$\boldsymbol{m}$=0*0*，a=1)，$R2$(pri=2，$\boldsymbol{m}$=*1*0，a=3)，图2展示了$R1$和$R2$的BT结构，且$R1$和$R2$的BT树存在相同的路径0100，由于$R1$的优先级更低，将$R1$的重复路径删除后重写为(pri=1，$\boldsymbol{m}$=000*，a=1).

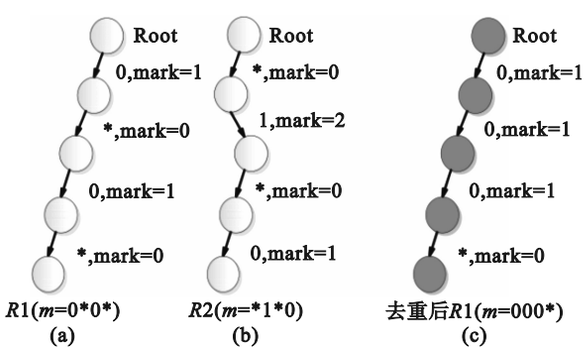

图2　流规则剪枝示例

Fig. 2　Flow rule pruning examples

(a)—$R1$的BT树；(b)—$R2$的BT树；(c)—对$R1$剪枝后的BT树.

复杂度分析：流表T有N个流规则，每个流规则的匹配向量长度为L.流规则转换BT结构的算法需要对每个流规则进行长度为L的遍历，生成最终的BT树，算法时间复杂度为$O(N\times L)$.BT剪枝算法需要先进行双重循环，遍历BT树，时间复杂度为$O(N\times N\times L)$.由于实际中流规则的覆盖范围分散，重叠率低，算法运行时路径不匹配会停止剪枝过程，因此实际运行时间低于最坏情况的运行时间.BT剪枝算法利用高优先级流规则剪枝低优先级流规则的重复区间，得到互不重叠的流规则，原有流规则的重复区间仅会匹配高优先级流规则，BT剪枝算法保证了流规则匹配过程的正确性.

2.4　并查压缩

本文利用并查压缩方法解决了最优覆盖问题.通过BTM方法得到互不重叠的BT树结构后，将其重新转换为流规则.由于所有BT树互不重叠，且为了避免BT树中存在的通配符匹配问题，删除了BT树的优先级字段，将BT树结构转换为key-mask形式，得到新的流规则TR(key，mask，a).

BT树转换为key-mask形式：遍历BT树，如果当前字符为0或1，则如实记录；如果当前字符为*，则记录为0，由此得到key.再次遍历BT树时，如果当前字符为0或1，则记录为1；如果当前字符为*，则记录为0，由此得到mask.例如，对于规则$R2$(pri=2，$\boldsymbol{m}$=*1*0，a=3)，其BT树结构如图2b所示，将其重写为流规则TR(key=0100，mask=0101，a=3).

接着本文提出一种基于并查压缩的合并算法，其基本思想：每个mask可看作一个集合，如果mask_i⊆mask_j，则可以将mask_i的流规则合并到mask_j中，直到无法合并后，将每个mask视作一个分区P，通过最优覆盖求解表达式规则.并查压缩算法的正确性可以通过mask的特性证明，给定两个元组$t1(k1,m1)$和$t2(k2,m2)$，假设$m1\subseteq m2$，存在表达式e满足$e(k1\&m2)=a1$，$e(k2\&m2)=a2$，则对于任意$t1$的取值i，其表达式的结果为$e(i\&m2)=e(i\&m1\&m2)=e(k1\&m2)=a1$，因此$t1$的取值可计算出对应的执行动作$a1$，$t2$的取值可计算出执行动作$a2$，且$k1$和$k2$的范围互不重叠，即合并方法具有正确性.区间合并后，当前区间枚举的表达式规则格式如表1所示，表达式规则以合并区间的mask为常数，与mask进行计算，最终结果必须与流规则的动作字段一致，计算出额外结果的表达式将被丢弃，因此并查合并算法可以保证表达式规则匹配的正确性.

复杂度分析：假设有N个流规则，并查压缩算法包含两部分工作：第一部分通过两重循环合并分区以及流规则，时间复杂度为$O(N\times N)$，流规

则也会随分区合并,实际程序中以地址方式将规则指向对应分区,以均摊的方式合并流规则,每个流规则仅被指向一次,因此合并分区的实际时间复杂度为$O(N\times N)$,假设合并成M个分区;第二部分对每个分区枚举表达式算子,求解最优表达式覆盖,枚举两个位置的算子共7×7种可能情况,时间复杂度为$O(49\times M)$.

2.5 规则下发

FRC算法通过去除流规则的重复范围,利用并查压缩方法获得表达式规则,需要将获得的表达式规则卸载到交换机.为了支持可编程交换机卸载表达式规则,需要对传统网络协议进行扩展,扩展协议简称FRC报文,如图3所示,包括FRC头部与FRC负载.FRC报文使用原IP协议头部的保留服务类型(ToS)字段中的1位标识报文(0表示普通报文,1表示FRC报文).FRC头部包含的NUM字段(1字节)表示规则个数.有效负载是由NUM个规则元组(规则标志、优先级、匹配向量、表达式或者动作)构成的数组,每个元组由1字节的标志Flag,1字节的优先级pri,4字节的匹配向量$\boldsymbol{m}$,和16字节的动作a组成,其中标志位为1代表表达式规则,标志位为0代表流规则.

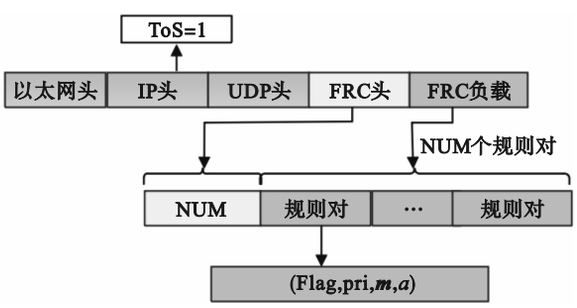

图3 FRC报文格式

Fig. 3 FRC message format

FRC报文可以携带表达式规则与流规则,通过对应标志区分,支持流规则与表达式规则的卸载.图4显示控制面构造并发送FRC报文,其中黑线部分表明FRC报文发送到交换机后被交换机解析、提取规则并部署的过程.首先,在控制面构造FRC数据包并装填规则,通过地址传输到交换机;其次,交换机解析数据包、提取规则并通过控制面将规则下发到寄存器.图4下方显示了数据面处理匹配报文的流程:匹配函数读取寄存器内的匹配向量,若命中则提取寄存器存储的动作字段;若是表达式则将相应参数代入计算结果,依据结果执行相应动作.FRC报文通过扩展UDP协议实现,因FRC更新报文仅需在交换机内更新而无需服务器侧处理,为进一步保证可靠性,FRC报文发送至可编程交换机后,由交换机控制面解析并存入规则.通过交换机内部设置计数器统计规则的优先级(规则pri字段)是否按顺序抵达并被存入,若发生丢包现象,则通过可编程交换机控制面构造数据包告知服务器依照当前优先级重传相应规则.交换机支持百亿规模的数据包并发处理,仅需百万级规则存储与更新,因此交换机处理FRC报文的可靠性完全可以保证.

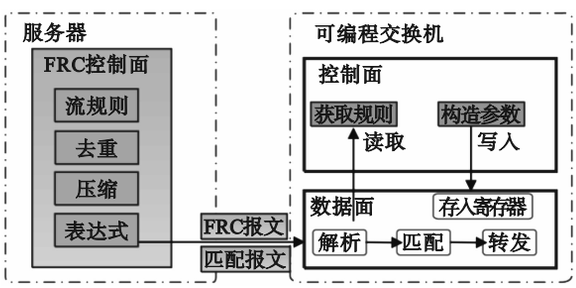

图4 FRC算法在可编程交换机上的卸载规则

Fig. 4 Offloading rules of FRC algorithm on programmable switch

FRC支持表达式规则和普通流规则的更新.为保证原有流规则的完整性以及更新后交换机匹配的正确性,需在软件端进行备份.若在交换机内添加流规则和表达式规则,则依据其类型构造FRC报文并卸载至交换机;对于新更新的流规则和表达式规则,需要设置更高的优先级,以保证匹配的正确性,同时在软件端进行备份.若在交换机内删除规则,则在软件端备份中删除对应流规则和表达式规则,同时构造一个FRC报文,将优先级更高且匹配向量一致的规则卸载到交换机.多次更新会大幅增加交换机的内存占用,FRC设置了内存阈值警告:当内存占用达到交换机可用内存的80%时,将清除交换机内所有规则并重新生成表达式规则.为保证FRC更新机制的准确性且减少处理的数据包,交换机接收并处理完FRC更新报文后,将FRC报文填充结束字段并返回端侧控制面;若端侧控制面在1 ms内未接收到处理完毕的FRC报文,则重传当前FRC更新报文,以保证更新操作的正确性和可靠性.

3 实验评估

3.1 实验环境配置

FRC的SDN控制面采用C++和Python语言实现,可编程交换机采用P4语言实现.在两台服

务器和一台P4可编程交换机组成的实验环境中评估FRC,每台服务器都配备了32核2.5 GHz的CPU,可编程交换机支持25 MB TCAM,可用于流规则和表达式规则的高速匹配.一台服务器作为“发送者”,通过交换机发送数据包到另一台服务器.研究选用ClassBench[15]生成真实网络的规则与数据包(包括安全策略ACL、防火墙FW和IP服务链IPC 3种类型的流规则20万条,数据包1 000万个),ClassBench被广泛应用于模拟真实网络的流规则与数据包,包括学术界数据包分类、数据检索过程中交换机内网络流规则生成与数据包匹配,以及工业界OpenSwitch、分布式数据检索等应用下软件定义网络和交换机内流规则生成与数据包匹配.

本研究设定单个FRC数据报文携带50条流规则,数据集包括两个控制策略流规则集(ACL-1,ACL-2)、两个防火墙流规则集(FW-1,FW-2)和两个IP协议控制流规则集(IPC-1,IPC-2),每个数据集包含20万条流规则和1 000万个数据包,每个流规则包括源地址、目的地址、源端口、目的端口和协议号5个字段.为测试FRC算法在可编程交换机下的更新效果,研究限制交换机使用高速2 MB内存存储所有流规则和表达式规则,单条规则占20 B内存,即交换机最多可存储10万条规则.

在上述实验环境下,将FRC与两种已广泛应用于软件定义网络的流表压缩算法(2016年DATF[16]和2017年IDFA[17])进行比较.本文从不同方面测试了这3种算法,主要包括性能、压缩率、压缩流规则的时间和迭代次数,同时测试了FRC的流规则更新时间.

3.2　数据包处理性能

图5展示了在不同数据集上采用3种不同算法处理网络数据包所需时间的对比分析.实验结果表明,FRC算法在处理数据包方面的性能显著优于DATF算法,其效率达到DATF算法的1.4至1.6倍;与IDFA算法相比,FRC算法的性能提升是其两倍.FRC算法之所以能够实现这一性能提升,主要归功于其基于表达式规则的快速计算机制,该机制大幅降低了数据包处理的延迟.FRC算法借助于在网计算的优势和精心设计的表达式规则,有效减少了数据包匹配过程中的延迟,提升了整体网络的处理性能.

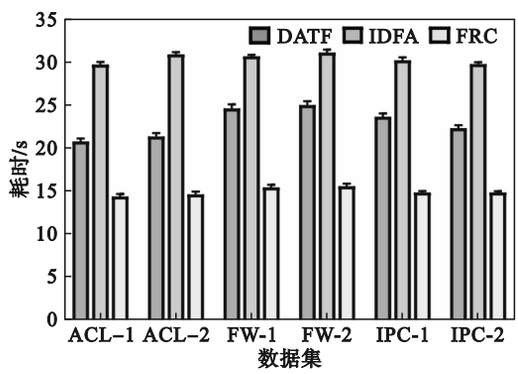

图5　3种压缩算法的处理数据包性能对比

Fig. 5　Performance comparison of three compression algorithms in processing data packets

3.3　压缩率

本文比较了3种压缩算法的压缩率(compression rate,CR).CR的计算公式为:(|*T*|-|CT|)/|*T*|×100%,|*T*|代表算法处理前网络中的原始规则数,|CT|代表压缩后的规则数.表2详细列出了在不同数据集上3种算法(FRC,DATF和IDFA)的压缩规则数量及其对应的压缩率.实验数据显示,FRC算法在压缩效率上表现卓越,平均能够减少99.8%的流规则数量,原因是FRC能利用表达式计算将多个流规则聚合为少数表达式规则,且可编程交换机支持表达式规则的运算,能快速匹配数据包.DATF算法实现了较高的压缩性能,平均压缩率达到96%.相比之下,IDFA算法的压缩效果相对较弱,平均压缩率约为84%.

表2　3种压缩算法的压缩规则数和压缩率对比

Table 2　Comparison of three compression algorithms in compression rule count and compression rate

数据集	规则数/条	DATF		IDFA		FRC	
		规则数/条	压缩率CR/%	规则数/条	压缩率CR/%	规则数/条	压缩率CR/%
ACL-1	20万	5 721	97.4	33 156	83.5	424	99.8
ACL-2	20万	5 543	97.2	31 174	85.4	416	99.8
FW-1	20万	8 826	95.6	34 102	82.9	517	99.7
FW-2	20万	8 924	95.5	34 678	82.7	528	99.7
IPC-1	20万	6 872	96.6	33 679	83.2	443	99.8
IPC-2	20万	7 023	96.4	32 786	83.6	456	99.8

3.4 流规则压缩时间

进一步对3种流规则压缩算法在处理不同规则集时的耗时进行实验分析.如图6所示,FRC算法在各类数据集上展现出卓越的压缩效率,能够在5 s内完成规则集的压缩任务.这一性能表现显著优于其他两种算法:与IDFA算法相比,FRC算法的平均压缩时间减少了70%;与DATF算法相比,则减少了40%.这些数据强有力地证明了FRC算法的时间效率优势.

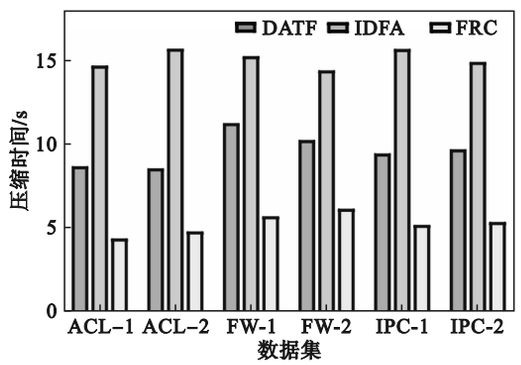

图6 3种算法的压缩规则集的耗时对比

Fig. 6 Time consumption comparison of three algorithms in compression rule sets

FRC算法之所以能够实现这一效率提升,主要归功于其独特的流规则去重和并查压缩方法.这些策略优化了压缩过程,确保了压缩后流规则的准确性和完整性.通过精心设计的算法流程,FRC算法能够快速识别并消除冗余流规则,同时高效地合并相似流规则,从而实现高效率压缩.

3.5 流规则更新时间

本文设定单个FRC数据包携带50条规则更新,系统地测试了FRC算法在不同规模规则更新操作中的耗时表现和交换机的TCAM内存占比,FRC算法在实验中共计使用2 MB TCAM内存.实验覆盖了从0.1到10万条规则的更新耗时和内存变化,包括纯普通流规则更新(100% *R*)、纯表达式规则更新(100% ExpR)以及混合规则更新(50% *R*和50% ExpR).更新耗时如图7a所示,实验结果揭示了FRC算法在更新不同类型规则时的时间效率差异,交换机内存占比变化如表3所示.FRC算法在更新普通流规则时所需的时间比更新表达式规则要长.原因归结为流规则在更新过程中需要频繁访问和修改TCAM存储单元,该过程较为耗时.随着交换机更新规则数量的增加,交换机内存占用量也随之等比例上升,导致交换机匹配数据包的效率下降.更新10万条数量的规则后,交换机的TCAM内存已经存满所有规则.实际情况下,更新至8万条规则时会触发控制面重新执行FRC算法,FRC算法会触发重新压缩规则的操作,并将压缩后的规则集部署到交换机中.图7b展示了更新10万条规则后交换机重新执行FRC算法后的整体时延变化.实验结果表明,更新大规模规则集后,FRC算法基本可以在2.4 s内完成10万条规则的压缩与卸载过程,并有效释放寄存器占用的内存资源,确保了网络流量的快速准确处理.

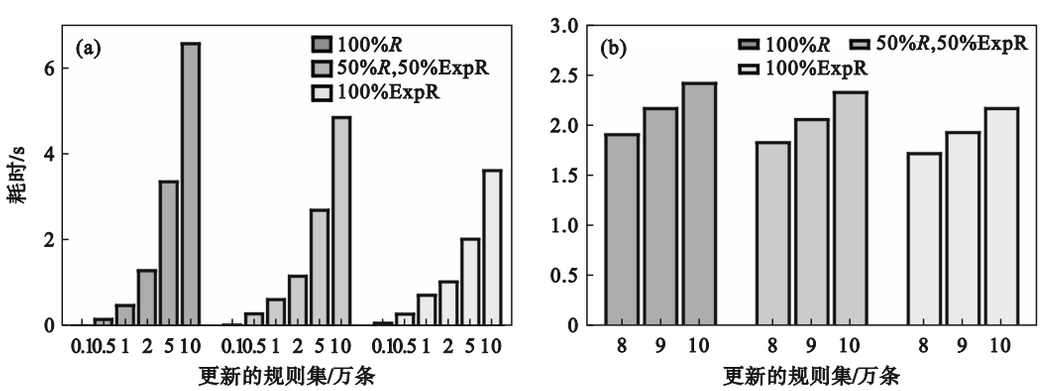

图7 FRC算法更新规则耗时

Fig. 7 Time consumption of FRC algorithm on updating rules

(a)—更新规则到交换机耗时;(b)—内存阈值告紧下重压缩耗时.

表3 FRC算法更新规则后交换机TCAM内存占用率对比

Table 3 Comparison of switch TCAM memory occupancy rate after rules updated by FRC %

规则占比	0.1条	0.5条	1万条	2万条	5万条	10万条
100% *R*	1.2	6.0	12.1	22.3	57.6	100
50%*R*,50%ExpR	1.1	5.7	11.5	21.5	55.3	100
100% ExpR	1.1	5.6	11.3	21.1	52.4	100

4 结　论

本文提出了一种基于在线计算的可计算表达式规则压缩算法(FRC).FRC算法通过流规则去重与并查压缩技术,有效解决了规则压缩过程中出现的区间重复问题以及最优压缩问题.此外,FRC通过利用可编程交换机对数据包进行重构,实现了规则的卸载,从而加速了数据包的匹配过程.实验结果表明,FRC能够显著减少流规则的数量,达到98%的流规则压缩率.在数据包匹配转发效率方面,FRC相较于其他方法提升了40%至60%.研究结果证明,FRC在云计算网络处理性能方面表现出色,有效应对了云计算环境下分布式应用的大规模流量处理效率低下以及网络交换机存储规模受限的问题.因此,FRC为流表压缩和高性能流量转发提供了在网计算加速的范式方案.

参考文献:

[1] Kang N X, Liu Z M, Rexford J, et al. Optimizing the "one big switch" abstraction in software-defined networks[C]// Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies. Santa Barbara, 2013: 13-24.

[2] Kanizo Y, Hay D, Keslassy I. Palette: distributing tables in software-defined networks [C]//2013 Proceedings IEEE INFOCOM. Turin, 2013: 545-549.

[3] Gouda M G, Liu X A. Firewall design: consistency, completeness, and compactness [C]//Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04). Tokyo, 2004: 320-327.

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

[5] Jiang W R, Prasanna V K. Energy-efficient multi-pipeline architecture for terabit packet classification [C]// GLOBECOM 2009-2009 IEEE Global Telecommunications Conference. Honolulu, 2010: 1-6.

[6] Curtis R, Mogul C, Tourrilhes, et al. DevoFlow: scaling flow management for high-performance networks [C]// ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. New York, 2011:1-10.

[7] 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.

[8] Ge J G, Chen Z, Wu Y L, et al. H-SOFT: a heuristic storage space optimisation algorithm for flow table of OpenFlow [J]. *Concurrency and Computation: Practice and Experience*, 2015, 27(13): 3497-3509.

[9] 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.

[10] 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.

[11] Lao C L, Le Y F, Mahajan K, et al. ATP: in-network aggregation for multi-tenant learning [C]//USENIX Symposium on Network Systems Design and Implementation. Boston, 2021:741-761.

[12] Sapio A, Canini M, Ho C Y, et al. Scaling distributed machine learning with in-network aggregation [C]// USENIX Symposium on Networked Systems Design and Implementation. Boston,2021:1-10.

[13] Jin X, Li X Z, Zhang H Y, et al. NetCache: balancing key-value stores with fast in-network caching[C]//Proceedings of the 26th Symposium on Operating Systems Principles. Shanghai,2017: 121-136.

[14] Zhang P H, Pan H, Li Z Y, et al. Accelerating LSH-based distributed search with in-network computation [C]//IEEE INFOCOM 2021—IEEE Conference on Computer Communications. Vancouver, 2021: 1-10.

[15] Matousek, Antichi J, Lucansky J, et al. ClassBench-NG: recasting ClassBench after a decade of network evolution [C]//2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems. Beijing, 2017: 204-216

[16] Mimidis A, Caba C, Soler J. Dynamic aggregation of traffic flows in SDN: applied to backhaul networks [C]// 2016 IEEE NetSoft Conference and Workshops. Seoul, 2016: 136-140.

[17] Chao T Y, Wang K C, Wang L C, et al. In-switch dynamic flow aggregation in software defined networks [C]//2017 IEEE International Conference on Communications (ICC). Paris, 2017: 1-6.