doi:10.3772/j.issn.1002-0470.2021.06.001

基于优先级替换的批量规则快速更新方法研究①

丁自旋②*** 于金萍* 李文斌* 毕经平③*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 当前,软件定义网络(SDN)交换机的三态内容寻址存储器(TCAM)基于优先级编码其存储规则的物理位置,因此,当发生规则更新时,TCAM 不可避免地会因为新优先级的出现或原有规则优先级的变化而移动大量已有规则的物理位置,产生不可忽视的高时延。本文提出了基于优先级替换的批量规则快速更新方法(BRUS),解决规则更新时因 TCAM 移动规则而产生的高时延。BRUS 引入了基于规则依赖的规则语义一致性,在规则语义一致性的基础上,通过替换插入规则优先级为删除规则的优先级来避免不必要的规则移动,实现快速的规则更新。实验结果表明,BRUS 能够有效找到91% 以上的替换规则对,从而大幅减少规则更新的移动次数。与最新方法相比,针对批量更新场景,BRUS 具有更好的稳定性和适用性。

关键词 软件定义网络(SDN);三态内容寻址存储器(TCAM)更新;规则更新;语义一致性;规则依赖

0 引 言

软件定义网络(software-defined networking, SDN)中,集中式的控制平面需要经常更新数据平面的规则以适应网络的动态变化。通常一次网络更新涉及一台交换机上的规则是多条的,发生在应用层的多个网络策略同时进行更新,控制器为了减少与交换机的通信时延会把可以并行更新的策略规则一起下发给交换机[1-2]。在三态内容寻址存储器(ternary content addressable memory, TCAM)作为缓存流表的场景中[3-4],发生在 TCAM 流表与软件流表之间的缓存规则替换也是批量进行的。然而,相关研究[5-7]指出,基于 TCAM 硬件的 SDN 交换机在批量规则更新(batch rule update scheme, BRUS)时存在高时延。TCAM 通常依据规则的优先级编排规则的物理位置,以便在并行查找数据包的匹配规则时快速返回位置最高的匹配规则。这种通过物理位置体现规则优先级的特性造成了 TCAM 更新规则时,需要移动大量规则来维持规则的优先级顺序,最终导致规则更新耗时长。

为了减少 TCAM 规则更新时不必要的规则移动,提高规则更新的速度,已有研究提出了设计流表结构和调度规则移动两种解决方法。其中,设计流表结构方法[8-10]将完整的大 TCAM 流表划分成若干较小的分区,使得每个分区上的最坏情况规则移动被限制在可接受的范围内,减少向紧凑排列的流表中插入高优先级规则而移动大量规则的情况。然而,这种方法并不适用于大量规则同时更新的场景,该类方法会产生大量冗余规则用来维持语义一致性。语义一致即指实现相同的数据包匹配功能,同时,其同步流表分区的操作也会加剧规则更新的延

① 国家自然科学基金(61702470,61472403)资助项目。

② 女,1990 年生,博士生;研究方向:软件定义网络;E-mail: dingzixuan@ict.ac.cn

③ 通信作者,E-mail: bjp@ict.ac.cn

(收稿日期:2020-06-17)

迟。调度规则移动[11-14]针对规则移动操作进行优化，直接在 TCAM 上调度规则的移动路径，对规则进行不按优先级的排列以减少规则的移动次数，同时保证其仍能实现正确的数据包匹配。

相比设计流表结构方法，调度规则移动方法更利于从根本上减少不必要的规则移动。已有调度规则移动方法的缺陷在于，只能获得单一规则更新的局部最优移动路径，而由于批量规则更新的调度规则移动问题不满足最优子结构性质，简单地采用贪心选择无法得到全局最优移动路径。

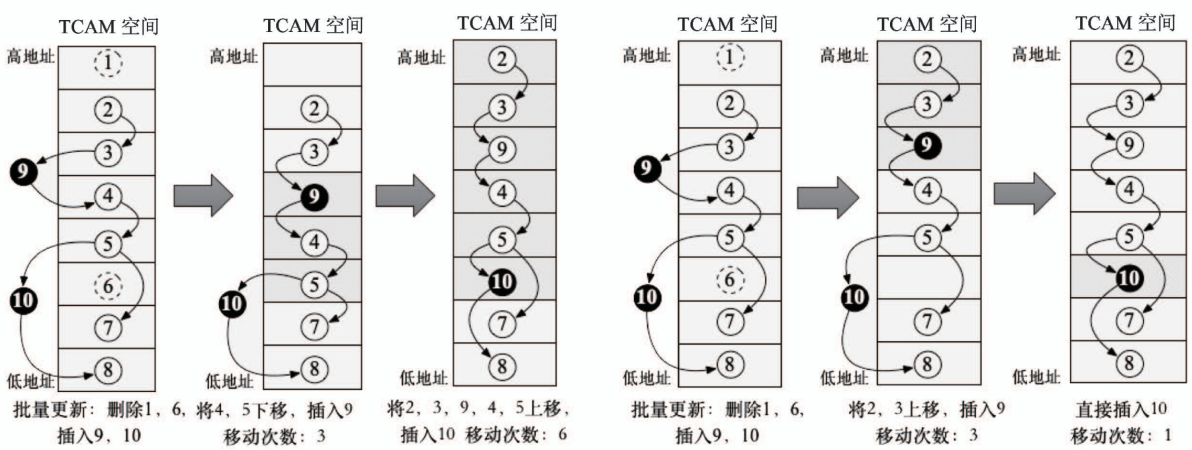
本文在已有的调度规则移动基础上，研究如何计算批量规则更新时的全局最优移动路径方法，减少批量规则更新的规则移动次数，进而提高批量规则更新的效率。为此，针对多规则更新中规则删除与插入并存的特点，本文创新地引入了维持语义一致性的“规则替换对”概念，并提出了基于规则优先级替换的更新机制，用规则替换对中的待删除规则优先级替换待插入规则优先级，使得 TCAM 中删除规则直接被替换为插入规则，从而避免其他规则移动。本文提出并证明了替换规则对的判断条件，并将批量规则更新上寻找规则替换对的问题建模为二分图最大匹配问题进行分析和解决，同时，提出了面向批量规则更新的最大匹配（batch rule update scheme-maximum match，BRUS-MM）和随机匹配（batch rule update scheme-random match，BRUS-RM）分别在优化效果和时间开销上各有优势的两种解决算法。实验表明，更新方法是对多规则更新进行规则移动优化的方法，相比针对单条规则更新的优化方法，在大规模、依赖复杂度高的规则集上，本文方法在优化效果和时间开销上都有明显优势，并且本文方法不依赖交换机硬件，具有更好的通用性和灵活性。

1　基于优先级替换的规则更新理论分析

1.1　动机

为确保规则优先级顺序而引发的规则移动是 TCAM 规则更新时延高的根本原因，已有研究工作针对这一问题提出的优化手段是直接操纵规则在 TCAM 上的移动路径。这种方法能够从根本上解决更新导致的大量规则移动问题，但对 TCAM 规则移动的操控必须实现为交换机上的固件程序，缺乏灵活性，无法在现有网络已部署的普通 SDN 交换机上直接使用。此外，目前已有的对 TCAM 上规则更新进行移动调度的方法，都仅面向单条规则的处理，当多条规则更新同时到达时，如果采用面向单条规则的方法逐条对规则调度移动路径，由于对先后更新的规则之间的关系不感知，容易产生冗余的规则移动。

例如，图 1（a）给出了逐条对规则调度移动路径，但仍产生大量规则移动的示例。多规则更新包含 4 条更新命令，删除规则 1、删除规则 6、插入规则 9 和插入规则 10。在删除规则 1 和 6 之后，为插入规则9寻找最短移动路径，此时根据空槽位的位置



(a) 逐条调度规则移动方法用于多规则更新效率低的示例　　(b) 逐条调度规则移动在多规则更新下最优规则移动示例

图 1　面向单条规则更新的调度规则移动方法在多规则更新场景下优化不足示例

有向上移动和向下移动两种选择且移动开销均为3,图1(a)选择了向下移动,将规则4、5下移后为规则9找到了合适的插入位置。但接下来为插入规则10寻找最短移动路径,就只能将规则2、3、9、4、5全部上移后插入规则10,在这个示例中为该批规则更新共调度了9次规则移动。作为对比,若在插入规则9时选择向上的移动路径如图1(b)所示,那么插入规则10时就可以直接插入无需移动其他规则,总共规则移动次数只需要4次。但目前已有方法均无法考虑当前规划的规则移动路径对后续更新规则的影响,无法保证实现图1(b)所示的最优移动,因此,需要进一步考虑优化多规则更新场景下的规则更新时延。

本文观察多规则更新的特点,其中既有删除规则又有插入规则,进而在维持规则语义的前提下,如果删除规则的位置能够直接被插入规则所使用,则可以通过优先级替换达到硬件层规则位置替换的效果,能够同时解决规则移动方法对硬件的紧耦合问题和在多规则更新下冗余规则移动的问题。在图1(b)的多规则更新例子中,删除规则6的TCAM位置就被插入规则10最终使用,形成规则替换的效果,直接避免了插入规则10需要的移动。

1.2 优先级替换的语义一致性条件

本文提出的更新方法的理论基础是修改规则优先级但维持语义的一致性。为此,本文借助规则依赖图描述规则之间由匹配域重叠和优先级大小导致的依赖关系。表1给出了相关术语和概念。

对于SDN流表规则之间由于匹配域重叠和优先级高低形成的位置排列依赖关系,本文用图结构来表示。对于一组给定的规则集 $R = \{r_1, r_2, \cdots, r_n\}$,其中每一条规则可以视为一个节点,对于其中一条规则 r,用 $r.p$ 表示规则 r 的优先级取值,用 $L(r)$ 表示规则 r 在TCAM上的物理位置。假设每条规则拥有唯一不重复的优先级值以保证规则语义的唯一性,同样,每条规则也拥有唯一不重复的TCAM物理位置。对于规则集 R 中的一条规则 r_i,若规则集 R 中存在另一条规则 r_j,满足 $r_i \cdot p > r_j \cdot p$ 且 $r_i \cdot m \cap r_j \cdot m \neq \varnothing$,即 r_i 的优先级高于 r_j 且二者匹配域存在重叠,则从 r_i 向 r_j 连一条有向边 (r_i, r_j),并称其为依赖边。依赖边 (r_i, r_j) 意味着 r_i 在TCAM中的物理位置必须高于 r_j,否则匹配域重叠区域的数据包就会被 r_j 误匹配;两条规则之间不存在依赖边,则说明它们没有重叠的匹配域,它们在TCAM中的物理位置关系不会影响数据包匹配。通过对规则集中所有规则两两检查依赖边存在的条件,即可构造出一张有向无环图 $G = (V, E)$,其中点集 V 与规则集 R 中的规则一一对应,边集 E 为所有规则之间形成的依赖边的集合,并称图 G 为规则依赖图。从规则依赖图可以得出,交换机TCAM上的规则排列能够实现规则集定义的语义,需保证所有有依赖关系的规则之间的相对顺序,即:

$$\forall (r_i, r_j) \in E, L(r_i) > L(r_j) \tag{1}$$

目前通用的交换机严格按照规则优先级排列规则,假如一条新插入规则的优先级可以修改为一条删除规则的优先级,那么交换机按修改后的优先级可以直接找到删除规则的位置插入规则,从TCAM上看相当于用插入规则替换了删除规则,从而实现不修改交换机固件程序,即可避免插入规则引起冗余规则移动。本文定义这样一对规则为“替换规则

表1 多规则更新重要概念

标记	描述
$R = \{r_1, r_2, \cdots, r_n\}$	TCAM初始规则集
$r.p$	规则 r 的优先级值
$r.m$	规则 r 的匹配域
$G = (V, E)$	规则依赖图 G,节点集合为 V,和规则集 R 中规则一一对应,依赖边集合为 E
(r_i, r_j)	规则依赖图上从规则 r_i 到规则 r_j 的一条有向边
$L(r)$	规则 r 在TCAM上的物理位置
$I = \{\cdots, r_p^{ins}, \cdots\}$	多更新规则中的插入规则集合
$D = \{\cdots, r_q^{del}, \cdots\}$	多更新规则中的删除规则集合
$\langle r_p^{ins}, r_q^{del} \rangle$	由一条插入规则 r_p^{ins} 和一条删除规则 r_q^{del} 组成的替换规则对
$H(r)$	拥有指向规则 r 的依赖边的规则集合
$W(r)$	拥有从规则指来的依赖边的规则集合
$MR(r^{ins})$	插入规则 r_p^{ins} 的优先级修改范围
$[r_p^{ins}, r_q^{del}]$	二分图上连接插入规则 r_p^{ins} 和删除规则 r_q^{del} 的一条无向边
$CE([r_p^{ins}, r_q^{del}])$	二分图上与边 $[r_p^{ins}, r_q^{del}]$ 冲突的边的集合

对”，记为 $\langle r_p^{ins}, r_q^{del}\rangle$，由一条插入规则 r_p^{ins} 和一条删除规则 r_q^{del} 组成。为了鉴别一个规则替换对能否成立，本文需要得到插入规则 r_p^{ins} 与 TCAM 中的剩余规则（初始规则集 R 除去删除规则集 D，即 $R-D$）之间的依赖关系。TCAM 的剩余规则中与 r_p^{ins} 有依赖关系的规则可以分为两部分：拥有指向 r_p^{ins} 依赖边的规则，记为 $H(r_p^{ins})$（$\forall r_u \in H(r_p^{ins})$，$\exists(r_u, r_p^{ins}) \in E$）；以及拥有从 r_p^{ins} 指来依赖边的规则，记为 $W(r_p^{ins})$（$\forall r_d \in W(r_p^{ins})$，$\exists(r_p^{ins}, r_d) \in E$）。根据式(1)，要使插入规则 r_p^{ins} 放置在 TCAM 上后能够维持规则集语义，则需使其位置高于 $W(r_p^{ins})$ 中所有规则的位置，且低于 $H(r_p^{ins})$ 中所有规则的位置。若删除规则 r_q^{del} 的位置能够满足此条件，则认为 r_p^{ins} 与 r_q^{del} 能够构成替换规则对。即一个替换规则对存在的条件为

$$\max_{r_d \in W(r_p^{ins})} L(r_d) < L(r_q^{del}) < \min_{r_u \in H(r_p^{ins})} L(r_u) \tag{2}$$

式(2)中的 $L(r_q^{del})$ 指代的是与 r_p^{ins} 有依赖关系的规则之间的位置约束，而与具体的 r_q^{del} 无关。为了使本文的方法能够应用在更为普适的软件层，本文需要从优先级的角度刻画规则替换对的存在条件。根据式(1)，有依赖关系的规则之间的优先级关系和 TCAM 相对位置需要维持一致，那么可以推得从规则优先级角度判断一个规则替换对的存在条件为

$$\max_{r_d \in W(r_p^{ins})} r_d \cdot p < r_q^{del} \cdot p < \min_{r_u \in H(r_p^{ins})} r_u \cdot p \tag{3}$$

式(3)也可以理解为一条规则的优先级修改范围，如图 2 所示。定义 $MR(r_p^{ins})$ 为插入规则 r_p^{ins} 的优先级修改范围，它的下界是 $\max_{r_d \in W(r_p^{ins})} r_d \cdot p$，上界是 $\min_{r_u \in H(r_p^{ins})} r_u \cdot p$。如果存在一条删除规则 r_q^{del}，其优先级在这个范围内，则可以和该插入规则构成一个替换规则对 $\langle r_p^{ins}, r_q^{del}\rangle$。优先级的替换可以映射到规则在 TCAM 上的替换，因为对于按优先级顺序排列规则的交换机来说，当它为 r_p^{ins} 寻找符合要求的空槽位时，由于 r_p^{ins} 的优先级已经修改为 $r_q^{del} \cdot p$，那么删除 r_q^{del} 后空出来的 TCAM 位置会被交换机的搜索算法判断符合要求而找到，因此可以直接在这个空槽位上执行写入 r_p^{del}，不再需要移动其他规则。

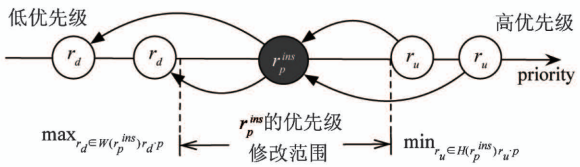


图 2 式(3)给出的优先级修改范围

2 基于优先级替换的多规则更新问题建模

基于式(3)，本文可以通过规则优先级判断一对替换规则对的存在条件。多更新规则中，通常包含一组插入规则和一组删除规则，本节对如何在多条更新规则上应用式(3)找到多对替换规则对、最小化多规则更新的总开销的问题展开讨论。

2.1 问题建模

本文假定初始 TCAM 上已经放置了规则集 R。在规则集 R 上的一个多规则更新包括一组待插入到 R 中的规则 $I = \{r_1^{ins}, r_2^{ins}, \cdots, r_P^{ins}\}$，以及一组待从 R 中删除的规则 $D = \{r_1^{ins}, r_2^{ins}, \cdots, r_P^{ins}\}$（$D \subseteq R$）。一个替换规则对意味着其中的插入规则可以被交换机通过一次写操作放置到删除规则的位置而不再需要其他规则移动，因此，本文的目标是找到一个最大的替换规则对集合 $S = \{\cdots, \langle r_p^{ins}, r_q^{del}\rangle, \cdots\}$，$r_p^{ins} \in I$，$r_q^{del} \in D$，其中的每一对替换规则都满足式(3)，并且每条规则最多在一个替换规则对出现。

插入规则集和删除规则集可以视为两类节点的集合，本文对每一条插入规则 r^{ins} 利用式(3)计算 $MR(r^{ins})$，然后在插入规则集中找到所有优先级在 $MR(r^{ins})$ 范围内的规则，为插入规则和每个符合条件的删除规则之间连一条无向边。这些符合条件的删除规则可以认为是 r^{ins} 的候选替换规则。通过为每一条插入规则寻找候选替换规则并连接边，可以得到一张二分图 $g = (v, \varepsilon)$。一个插入规则只能和一个删除规则形成替换规则对，因此，寻找最大替换规则对的问题可以建模为二分图上的最大匹配问题。

2.2 冲突边约束

上一小节中，$MR(r^{ins})$ 的计算只考虑了一条插入规则 r^{ins} 与 TCAM 中剩余规则 $R-D$ 的依赖关系，

并没有考虑插入规则集中的规则之间形成的依赖关系。对于一条插入规则 $r^{ins} \in I$,它的优先级修改范围应该由规则集 $R + I - D$ 共同决定。但是,I 中的规则的优先级本身就需要根据最大匹配的结果而定,它们之间会互相产生影响。一个具体的例子在图 3 中给出,只考察插入规则和 TCAM 中剩余规则之间的依赖关系构建了二分图,如图 3(a)所示,两条插入规则共享一些候选替换规则,在图中用灰色标出。注意到两条插入规则 r_i^{ins} 和 r_j^{ins} 之间存在一条依赖边 (r_i^{ins}, r_j^{ins}),若首先从 r_i^{ins} 的候选替换规则中选择了 r_4^{del} 与 r_i^{ins} 构成一个替换规则对 $\langle r_i^{ins}, r_4^{del} \rangle$,那么再看 r_j^{ins} 的优先级修改范围,因为 $r_j^{ins} \in H(r_i^{ins})$,所以 $MR(r_j^{ins})$ 的上界计算应该将修改优先级为 $r_4^{del} \cdot p$ 的 r_i^{ins} 一起考虑。如果不考虑 r_i^{ins} 修改后的优先级,原本的 $MR(r_j^{ins})$ 会覆盖更广的范围;修改 $r_i^{ins} \cdot p$ 为 $r_4^{del} \cdot p$ 后,如图 3(b)所示,$MR(r_j^{ins})$ 的上界受到影响覆盖范围变窄。体现在二分图的最大匹配问题上,选择 r_4^{del} 与 r_i^{ins} 匹配后,r_j^{ins} 的候选替换规则中那些优先级值高于 $r^{del} \cdot p$ 的规则不能再与 r_j^{ins} 构成替换规则对。也就是说,图中边 $[r_i^{ins}, r_4^{del}]$ 被选择后,会对 r_j^{ins} 的侯选边产生影响,除去边 $[r_j^{ins}, r_4^{del}]$ 因为不能多次匹配而不能被选择,边 $[r_j^{ins}, r_2^{del}]$ 和 $[r_i^{ins}, r_3^{del}]$ 为了维持语义一致性也不能再被选择。

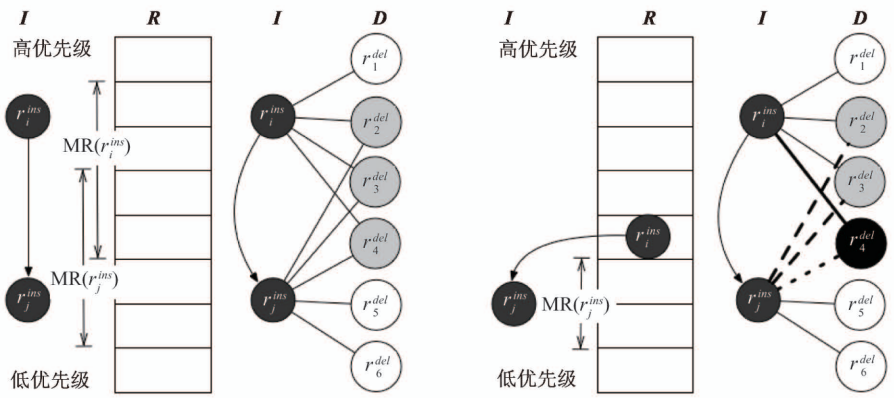


图 3　带冲突边约束的二分图最大匹配

根据以上分析,如果插入规则集的规则之间存在依赖关系,那么插入规则的优先级修改范围会在寻找最大匹配的过程中改变。本文在寻找最大匹配的过程中加入约束条件以维持插入规则之间依赖关系的正确。如果二分图上一条边 $[r_p^{ins}, r_q^{del}]$ 被选择加入最大匹配集,那么满足如下条件的边应该被移除。

(1) 如果插入规则之间存在依赖边 (r_h^{ins}, r_p^{ins}),那么对于二分图上所有满足 $r_k^{del} \cdot p < r_q^{del} \cdot p$ 的边 $[r_h^{ins}, r_k^{del}]$,都要移除。

(2) 如果插入规则之间存在依赖边 (r_p^{ins}, r_w^{ins}),那么对于二分图上所有满足 $r_k^{del} \cdot p > r_q^{del} \cdot p$ 的边 $[r_w^{ins}, r_k^{del}]$,都要移除。

由插入规则之间的依赖关系导致的二分图上一些匹配边无法和另一些匹配边共存,本文称其为冲突边(conflict edges),用 $CE([r_p^{ins}, r_q^{del}])$ 标记所有与边 $[r_p^{ins}, r_q^{del}]$ 相冲突的边,在图 3(b)的例子中,$CE([r_i^{ins}, r_4^{del}]) = \{[r_j^{ins}, r_2^{del}], [r_j^{ins}, r_3^{del}]\}$。算法 1 描述了创建二分图及寻找所有冲突边的过程。首先,根据式(3)计算每个插入规则的优先级修改范围,与每个优先级在范围内的删除规则连接二分图的边(第 2 ~ 10 行)。注意到这里计算优先级修改范围时仍然是只考虑单条插入规则与 $R - D$ 中规则的依赖关系。由于插入规则的最终优先级是本文的求解目标且在求解过程中是动态变化的,本文用冲突边记录插入规则之间的依赖关系。在前文中本文总结了插入规则之间的依赖关系产生于两个方向,在伪码搜索过程中,本文可以通过沿依赖边方向单向搜索但是双向添加冲突边降低复杂度同时保证查

找的完整性,第 11 ~25 行描述了这一过程。

为一条插入规则计算优先级修改范围的最坏情况时间复杂度为 $O(|I|\cdot|E'|)$,E' 为从规则依赖图中删除 D 再插入 I 后的依赖边集合,构建二分图的复杂度为 $O(|I|\cdot|D|)$。由于插入规则之间的形成依赖边是 E' 的子集,算法的搜索过程中也会跳过优先级修改范围不产生重叠的情况,因此识别所有冲突边的最坏情况复杂度不会超过 $O(|E'|\cdot|D|^2)$。

算法 1 创建二分图并识别冲突边

1: 在依赖图 G 上移除规则 D, 插入规则 I, 得到新的依赖图 $G'=(V',E')$
2: **for** $r^{ins}\in I$ **do**
3: $MR(r^{ins})\leftarrow(\max_{(r^{ins},\,r)\in E',r\in R\setminus D}r.p,\ \min_{(r,r^{ins})\in E',r\in R\setminus D}r.p)$
4: $v\leftarrow I\cup D$
5: **end for**
6: **for** $r^{ins}\in I$ **do**
7: **for** $r^{del}\in D$ **do**
8: **if** $r^{del}\cdot p$ 在 r^{ins} 的优先级修改范围 $MR(r^{ins})$ 内 **then** 为二分图 g 添加边 $[r^{ins},\ r^{del}]$
9: **end for**
10: **end for**
11: **for** $(r_i^{ins},\ r_j^{ins})\in E'$ **do**
12: **if** $MR(r_i^{ins})$ 的下界大于 $MR(r_j^{ins})$ 的上界 **then**
13: $replace\_i\leftarrow\{r^{del}\mid[r_i^{ins},r^{del}]\in\varepsilon\}$,并按优先级升序排列
14: $replace\_j\leftarrow\{r^{del}\mid[r_j^{ins},r^{del}]\in\varepsilon\}$,并按优先级升序排列
15: $m\leftarrow0$
16: **while** $m<size(replace\_i)$ **do**
17: $n\leftarrow replace\_j$ 中第一个满足 $replace\_j[n].p>replace\_i[m].p$ 的规则的下标
18: **while** $n<size(replace\_j)$ **do**
19: 添加边 $[r_j^{ins},\ replace\_j[n]]$ 到冲突边集合 $CE([r_i^{ins},\ replace\_i[m]])$
20: 添加边 $[r_i^{ins},\ replace\_i[n]]$ 到冲突边集合 $CE([r_i^{ins},\ replace\_i[m]])$
21: $n\leftarrow n+1$
22: **end while**
23: $m\leftarrow m+1$
24: **end while**
25: **end for**
26: **return** $g=(v,\ \varepsilon)$ 和 $CE(\varepsilon)$

2.3 多规则更新调度算法

二分图最大匹配问题可以转化成有向无权重图上的最大流问题并采用经典的 Ford-Fulkerson 算法或 Edmonds-Karp 算法求解,时间复杂度为 $O(|v|\cdot|\varepsilon|)$。在本文的实现代码(算法 2)中,递归地寻找增广路径并扩大匹配集合,在算法 2 中通过 MaximumMatch 和 FindPath 两个函数实现。对于每一条插入规则 r_i^{ins},查看它在二分图上连接的每一个删除规则并尝试将这两条规则加入到匹配集合。如果删除规则 r_j^{del} 还未被指派到其他匹配,那么将匹配 $\langle r_i^{ins},\ r_j^{del}\rangle$ 加入到匹配集合;否则,如果 r_j^{del} 已被指派给另一条插入规则 r_k^{ins},本文则递归地检查 r_k^{ins} 能否和其他规则构成一个匹配(第 15 行)。为了保证 r_k^{ins} 不被匹配回 r_j^{del},在递归调用前将 r_j^{del} 标记为“已访问”(第 14 行)。以上过程可有效地解决原始的最大匹配问题,但是多规则更新建模的最大匹配问题是有冲突边约束的,需要在指派一个匹配之后,将与这个匹配相冲突的匹配边从二分图中删除(第 17 ~18 行)。这样才能保证插入规则之间的依赖关系不会产生动摇。

本文提出的最大匹配算法为每一条插入规则都需要递归地遍历所有匹配边,这个过程的基本的时间复杂度和原始最大匹配问题解决方法的时间复杂度一致,为 $O(|v|\cdot|\varepsilon|)$。但由于引入冲突边约束,本文需要动态地删除冲突边,因此引入额外的 $O(|\varepsilon|)$ 开销,算法的总时间复杂度为 $O(|v|\cdot|\varepsilon|^2)$。

由于大量的边 ε 会加大计算时间开销,本文还提出了另一种随机指定匹配的方法作为降低算法时间开销的选择,分别称这两种优化方法为 BRUS-MM 和 BRUS-RM。不同于最大匹配方法 BRUS-MM 为每个插入规则都遍历所有匹配边,随机匹配方法 BRUS-RM 只对插入规则进行一次遍历,在它的可匹配规则中随机选一个作为匹配,之后即便出现无法匹配的情况也不再回溯寻找别的可能,因此 BRUS-RM 方法的时间复杂度仅为 $O(|v|\cdot|\varepsilon|)$。两种方法在优化效果和时间开销的比较将在第 3.2 节进行讨论。

算法 2 同时给出了多规则更新调度的完整过程。首先,利用算法 1 创建二分图并找到所有冲突规则,接下来采用 MaximumMatch 方法或 RandomMatch 方法计算一个最大替换规则对集合。之后对交换机下发更新命令,先将所有删除规则的命令下

算法 2 多规则更新调度方法 BRUS

```
创建二分图 g = (v, ε),并记录所有冲突边 CE(ε)
S ← ∅
调用 MaximumMatch 或 RandomMatch 计算最大匹配规则对集合 S
删除规则 D
for ⟨r_i^ins, r_j^del⟩ ∈ S do
r_i^ins · p ← r_j^del · p
end for
M ← {r_i^ins | ⟨r_i^ins, r_j^del⟩ ∈ S}
将 M 中的新增规则更新到 TCAM
将 I-M 中的新增规则更新到 TCAM

procedure FindPath(r_i^ins)
for [r_i^ins, r_j^del] ∈ ε 中的每个 r_j^del do
if 未访问 r_j^del then 标记 r_j^del 为已访问
if ∄⟨r_k^ins, r_j^del⟩ ∈ S 或 FindPath(r_k^ins) then
添加匹配对 ⟨r_i^ins, r_j^del⟩ 到匹配集合 S
for [r_u^ins, r_v^del] ∈ CE([r_i^ins, r_j^del]) do
从二分图边 ε 中删除边 [r_u^ins, r_v^del]
return true
end for
end for
end procedure
procedure MaximumMatch
    for r_i^ins ∈ I do
∀ r_j^del ∈ D, 标记 r_i^del 为未访问
FindPath(r_i^ins)
    end for
end procedure
procedure RandomMatch
∀ r_j^del ∈ D, 标记 r_j^del 为未访问
for r_i^ins ∈ I do
从 [r_i^ins, r_j^del] ∈ ε 中随机选择一个未访问规则 r_j^del,并标记为已访问
添加匹配对 ⟨r_i^ins, r_j^del⟩ 到匹配集合 S
for [r_u^ins, r_v^del] ∈ CE([r_i^ins, r_j^del]) do
从二分图边 ε 中删除边 [r_u^ins, r_v^del]
end for
end for
end procedure
```

发给交换机,清空其槽位;再对 S 中每一个替换规则对,将插入规则的优先级修改为对应的删除规则的优先级,之后将修改了优先级后的插入规则的命令下发给交换机;最后,如果除去 S 中的插入规则后还剩余未能匹配的插入规则,本文将这部分插入规则的命令不做修改下发给交换机。这个更新计算及调度的过程可以实现在控制器上或者交换机的软件层。因为优化仅仅是对规则优先级的修改,并且输入和输出的都是从控制器下发的标准的规则更新命令(如 OpenFlow 协议),这个过程不需要交换机固件的支持,对硬件完全解耦。由于本节内容专注于保证优先级修改不动摇语义一致以及这种优先级的修改应用在普通交换机上可以减少 TCAM 上的规则移动,因此并不对具体实现和应用方法展开详细讨论。

3 性能评估

本文比较了批量规则更新场景下,BRUS 和已有的规则移动调度方法的执行效果。本文所有实验运行在单台 Linux 主机上,采用 Intel Core i7 2.5 GHz CPU 和 16 GB DDR3 内存。

3.1 仿真实验设置

本文采用了 ClassBench 提供的 3 个种子规则数据 ACL1、FW1 和 IPC1,分别生成含有 1k、5k、10k 条规则的规则集。然后,利用文献[15]提供的方法将这些规则翻译成 SDN 的通配符形式流表规则,并为这些规则创建规则依赖图。表 2 列出了这些规则集最终包含的规则数量、规则依赖图的边数和最大的依赖链的深度。每次实验随机抽取一部分作为待更新的多规则,并设置为删除规则或插入规则,剩余规则被认为初始存在于 TCAM 上。

表 2 合成规则集

种子文件	规则数量	依赖边数量	依赖链最大深度
ACL1 1k	1168	1228	7
FW1 1k	2551	7420	24
IPC1 1k	1234	2466	9
ACL1 5k	5766	6497	13
FW1 5k	13 739	83 030	43
IPC1 5k	6055	17 021	55
ACL1 10k	12 765	13 966	23
FW1 10k	40 998	311 206	64
IPC1 10k	11 953	43 310	71

本文模拟了普通交换机上的规则更新行为，即按照优先级顺序排列和移动规则，并且统计规则移动次数，记为 priority-based。本文的优化方法将一对匹配规则记为一次规则移动，由于本文方法是面向不需定制修改固件的普通交换机的，因此未匹配规则需要被交换机按优先级顺序排列和移动，用 priority-based 模拟方法统计这部分规则更新的移动次数。对于一批更新规则，分别用本文提出的 BRUS-MM 方法和 BRUS-RM 方法统计规则移动次数，并与单纯的 priority-based 方法和目前面向单规则调度规则移动的方法[13]进行对比。本文的模拟方法假设 TCAM 初始状态为按规则优先级顺序紧凑排列的。由于删除规则相比插入规则只需花费非常小的常量时间[5,8]，参考其他工作[12-14]，这部分开销本文中忽略不计。

3.2　实验结果分析

首先将本文所提方法与直接在普通交换机上更新 priority-based 进行对比，计算节省的规则移动次数以检验本文方法的有效性。在实践 BRUS-MM 和 BRUS-RM 方法的过程中发现，只要多规则更新中有足够数量的删除规则，大部分插入规则都能成功找到规则匹配对，这也与实验中观察到二分图的边数量多的现象能够互相解释。大部分插入规则都有比较广的优先级修改范围，落在范围内的删除规则数量也很多，这种高密度的二分图能够很大概率提升最大匹配的数量。最大匹配的理论上限是插入规则数量和删除规则数量中的最小值，即 $\min(|I|, |D|)$。与理论上限对比，无论插入规则和删除规则的组成比例如何，BRUS-MM 和 BRUS-RM 分别能够稳定地找到超过 94% 和 91% 的理论上限的替换规则对。

本文方法将未能找到匹配的插入规则交给 priority-based 方法处理，而未能找到匹配的主要原因来自删除规则的数量少于插入规则数量，因此本文方法的优化效果受 $|I|$ 和 $|D|$ 组成比例的影响显著。图 4 给出了 BRUS-MM 和 BRUS-RM 方法在不同的 $|I|$ 和 $|D|$ 组成比例下相比 priority-based 方法节省的规则移动数量的百分比。首先，由于衡量的是节省的总移动次数的比例，BRUS-MM 和 BRUS-RM 方法在这个量纲上的优化效果差别不是很明显，但也可以看出 BRUS-MM 略微优于 BRUS-RM 方法。接下来分析不同 $|I|$ 和 $|D|$ 组成比例的影响，插入规则数量少于删除规则数量时，插入规则基本都能找到替换规则对，可以节省 94% ~98% 的规则移动，优化效果显著；插入规则多于删除规则时，优化效果随着 $|I|:|D|$ 的增长而下降，因为越来越多的插入规则需要被 priority-based 方法处理，但是即便 $|I|:|D|$ 达到 9:1，仍能节省 15% 以上的规则移动，因此本文方法在各种组成比例下都是有效的。

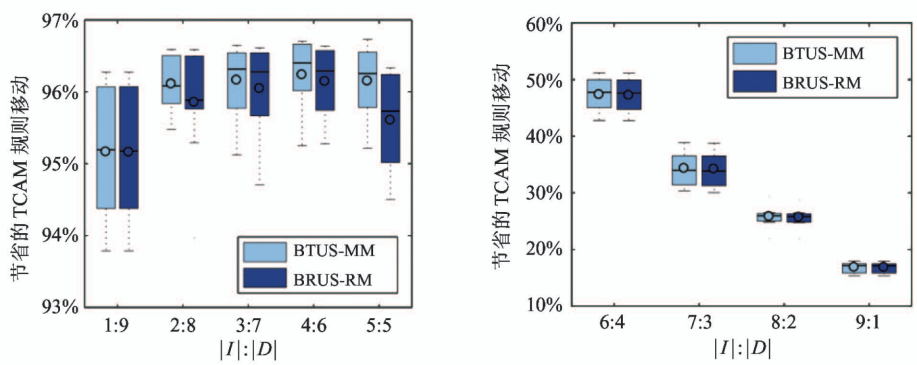


图 4　不同 $|I|:|D|$ 下节省的 TCAM 规则移动次数

接下来固定 $|I|:|D| = 5:5$，检验本文方法在不同更新数量、不同 TCAM 初始规则大小、不同策略集规则下的通用性。固定 $|I|:|D| = 5:5$ 的原因是当 $|I| > |D|$ 时，$|I|$ 和 $|D|$ 的组成比例成为影响总移动次数的主导因素，不适合作为不变量去衡量其他变量的影响；当 $|I| \leqslant |D|$ 时，$|I| = |D|$ 的情况对于本文的优化方法来说是最坏情况，因为更多的删除规则会给每个插入规则带来更多的替换

规则对的选择可能,因此本文采用最坏情况作为以下实验的设置。

图 5 给出了 BRUS-MM、BRUS-RM 和 priority-based 方法在表 2 列出的各种策略规则集和规则集大小,以及不同更新数量的情况下需要的规则移动次数。BRUS-MM 和 BRUS-RM 比起 priority-based 方法能够减少两个数量级的规则移动(纵坐标为指数跨度)。并且随着多规则更新数量的增长、规则集大小的增长,BRUS-MM 和 BRUS-RM 方法的增长明显比 priority-based 方法平滑。对于不同策略规则集,规则依赖的复杂度会很大程度上影响规则移动次数的量级。从表 2 可以看出,FW 策略比 ACL 和 IPC 策略有更大的依赖图密度和更长的依赖链长度,表现在更新需要的规则移动次数上,FW 策略比 ACL 和 IPC 需要更多的规则移动。对比 BRUS-MM 和 BRUS-RM,BRUS-MM 比 BRUS-RM 需要更少的规则移动,将 BRUS-RM 作为基准,BRUS-MM 平均可以节省 32.58% 的规则移动次数。

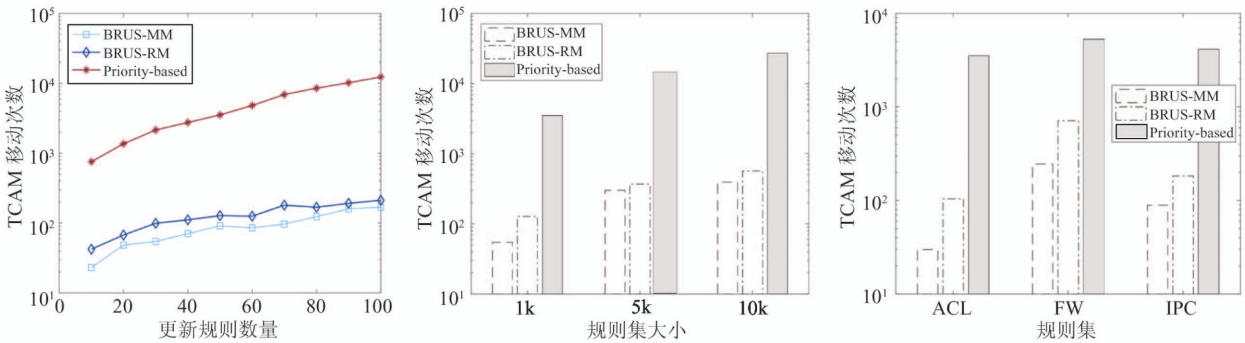


图 5　更新大小、规则集大小、策略集不同对规则移动次数的影响

图 6 给出了对应的图 5 的实验下 BRUS-MM 和 BRUS-RM 所花费的计算时间。计算时间可以分为构建二分图和寻找最大匹配两部分。其中构建二分图的时间开销是 BRUS-MM 和 BRUS-RM 共享的。对于寻找最大匹配的时间开销,随着更新大小的增长和规则集大小的增长,BRUS-MM 的计算时间增长比 BRUS-RM 更陡峭,这个趋势符合 2.4 节的复杂度分析。平均来说,BRUS-MM 需要 1.61 倍于 BRUS-RM 的计算时间。

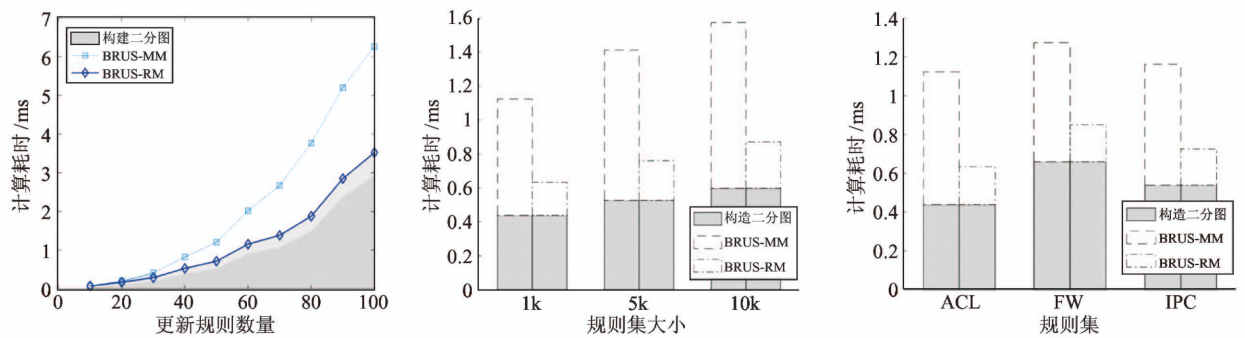


图 6　更新大小、规则集大小、策略集不同对计算时间开销的影响

进一步对 BRUS-MM 和 BRUS-RM 的对比展开讨论,尽管 BRUS-RM 方法采用随机选择匹配的策略节省了计算时间开销,但规则移动次数比 BRUS-MM 多。从计算时间开销来看,平均每条规则的计算时间只有几微秒,但 TCAM 交换机平均每条规则更新时延是在毫秒级[5-7],这使得计算时间开销相比之下微不足道。根据目前大部分交换机的 TCAM 更新速度,采用 BRUS-MM 用相对很小的计算开销节省更多的规则移动时间更为推荐。由于不同厂商的不同型号交换机的 TCAM 规则更新时延也不尽

相同,这两种方法的抉择最终取决于实际使用时具体的交换机硬件的规则移动速度和软件层运行本文方法的速度。网络管理员可以根据这两方面的性能综合决定,采用 BRUS-MM 的方法花 1.6 倍于 BRUS-RM 方法的计算时间,换取节省 32.58% 的 TCAM 规则移动是否值得。

此外,将本文的方法与当前针对单条规则更新在 TCAM 上的移动调度的 Γ down 方法[13] 进行对比。Γ down 方法面对多规则更新,需要对更新规则逐条调用寻找规则移动路径的算法,本文将平均每规则需要的移动次数和计算时间进行对比。图 7 的结果表明,在规则移动次数方面,BRUS-MM 总是优于 Γ down,平均能够减少 17% 的规则移动次数,尤其在大规模、依赖复杂度高的规则集上优势显著,BRUS-RM 方法为降低计算开销而牺牲了优化效果,并且随机匹配策略带来的不稳定性导致优化效果存在波动,因此相比 Γ down 方法时优时差;在计算时间开销方面,BRUS-MM 和 BRUS-RM 方法在各种策略、各种大小规则集上的计算开销都比 Γ down 方法更稳定。在小规模规则集和 ACL 5k 的规则集上,Γ down 方法的优势比较明显。Γ down 方法的原理是在更新规则所在的依赖链上调度移动路径,从表 2 可以看出,这些规则集构成的规则依赖图密度低、依赖链短、结构简单,因此有利于 Γ down 方法计算。本文方法需要先对全规则集构建二分图,相比 Γ down 方法仅关心单个规则所涉及的依赖链要花费更多时间。但随着规则集规模增大,规则依赖结构更加复杂,单个规则涉及的依赖链变得更大,Γ down 方法的计算时间开销也急剧增长,此时 BRUS-MM 对多规则同时优化的优势开始凸显,构建二分图和寻找最大匹配的时间均摊到每一条规则上的增量不再明显。综上,在规则集规模小、规则之间的依赖复杂度低的场景下,BRUS-MM 的优化效果略优于 Γ down 方法,但计算时间开销大,在此情况下建议根据交换机实际的 TCAM 速度衡量 BRUS-MM 移动较少规则节省的时间能否抵消其计算多花的时间,据此选择 Γ down 方法或 BRUS-MM 方法;在规则集规模大、规则之间依赖复杂度高的场景下,BRUS-MM 无论在优化效果或时间开销上都显著优于 Γ down 方法;BRUS-RM 方法在优化效果上存在不稳定因素,但整体上优于 Γ down 方法,并且在时间开销上稳定优于 BRUS-MM 方法。若追求优化效果的稳定则建议选择 BRUS-MM 方法,若交换机实际 TCAM 速度较高,则可根据前文对 BRUS-RM 和 BRUS-MM 的对比数据,以 1.6 倍的计算时间节省 32.58% 的 TCAM 规则移动是否值得来选择 BRUS-MM 方法或 BRUS-RM 方法。

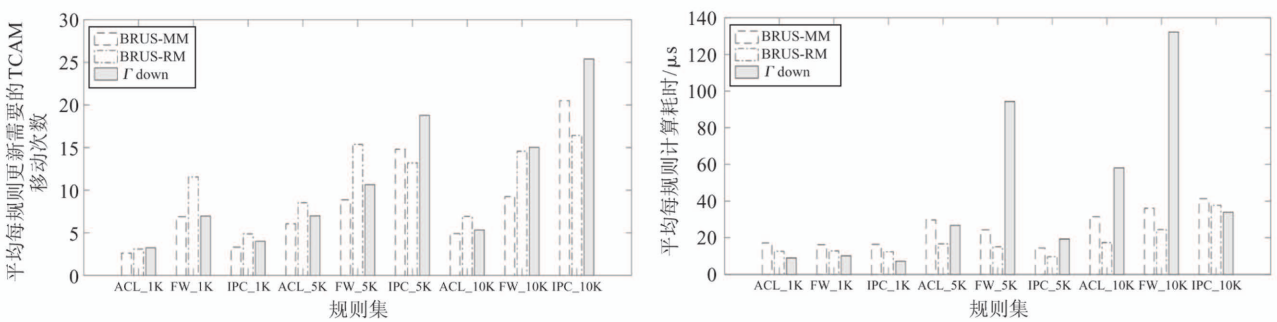


图 7　与 Γ down 方法的平均每规则移动/计算时间对比

4　相关工作

针对基于 TCAM 实现流表的 SDN 交换机存在规则更新时延高的问题,目前国内外的相关研究工作主要通过两种途径对规则更新速度进行优化,即设计流表结构和调度规则移动。

设计流表结构的方法采用了“化整为零”的思想,将完整的大 TCAM 流表进行划分,使得最坏情况规则移动次数被限制在一个有限的可接受的范围内。这种流表划分的方法具体可以细分为两种,即流表内部分区和多流表设计。流表内部分区是利用

硬件层的 TCAM 分区(chunk)功能的支持,将完整的 TCAM 划分为若干个较小的分区[9, 13]。多流表设计是在流表架构上采用 TCAM 流表和其他流表配合的方式实现规则的存储、查找和更新[8,10,16]。

调度规则移动目的是从根本上解决 TCAM 规则更新速度慢的问题,即在保持规则按优先级顺序排列的基础上,减少插入规则导致的规则移动次数。其方法是直接在 TCAM 上对规则更新调度的移动路径,使新的位置排列能保证规则语义一致性的同时最小化规则的移动次数[11,12,14,17,18]。

然而,目前针对 SDN 规则更新调度移动的工作都是针对单条插入规则的最短移动展开,在多规则更新场景下存在优化效率低的问题。若对多规则更新逐条应用目前的单规则移动调度方法,为每条更新规则单独计算移动路径,会因为视野的局限导致每条规则的最优移动路径并不一定能保证多规则下的整体最优规则移动,上一条更新规则产生的位置排列可能会使下一条更新规则需要数倍的移动次数,从而造成在多规则更新场景下的冗余移动,产生高更新时延。目前尚未有针对多规则更新场景下的快速规则更新方法。

此外,调度规则移动的方法由于对规则排列位置进行了持续性的更改,在更新持续到达的场景下,会导致之前到达的更新引起的位置排列,在后到达的更新规则引入新的依赖关系时产生依赖冲突。目前针对这种依赖冲突问题,仅文献[13]提出了这个问题的存在,并采用对冲突规则进行反复重排列的方法解决冲突,这种原始的方法虽然最终能够得到语义一致的规则排列,但代价是大量冗余的规则移动,最终导致规则更新时延高。

5 结 论

本文针对当前 TCAM 批量规则更新由于维持优先级顺序而导致的高时延问题,提出了基于规则优先级替换的批量规则快速更新方法(BRUS)。通过计算维持规则语义一致性的替换规则对,在 TCAM 硬件更新之前,将替换规则对中插入规则的优先级替换为删除规则的优先级,避免了 TCAM 硬件层上不必要的规则移动。本文分别从理论层面和实验层面证明了该方法的可行性和有效性。仿真实验结果表明,BRUS 能够有效地找到 91% 以上的替换规则对,从而大幅减少规则更新的移动次数,并且可以有效地适用于各种更新规模、规则集规模和策略规则集的场景。与针对单条规则移动优化的方法相比,通过模拟普通的按优先级排列规则的交换机上的规则更新,能够平均减少 17% 的规则移动,且在面向大规模、依赖复杂度高的规则集的更新时,BRUS 的优化效果和时间开销更稳定。此外,BRUS 方法不依赖硬件结构,具有更广阔的应用前景。

参考文献

[1] Dudycz S, Ludwig A, Schmid S, et al. Can't touch this: consistent network updates for multiple policies[C]// IEEE/IFIP International Conference on Dependable Systems and Networks, Toulouse, France, 2016: 133-143

[2] Nguyen T D, Chiesa M, Canini M, et al. Decentralized consistent updates in SDN[C]//Proceedings of the Symposium on SDN Research, New York, USA, 2017: 21-33

[3] Katta N, Alipourfard O, Rexford J, et al. CacheFlow: dependency-aware rule-caching for software-defined networks[C]//Proceedings of the Symposium on SDN Research, Santa Clara, USA, 2016: 1-12

[4] Sheu J P, Chuo Y C. Wildcard rules caching and cache replacement algorithms in software-defined networking [J]. *IEEE Transactions on Network and Service Management*, 2016, 13(1):19-29

[5] Kuźniar M, Perešíni P, Kostić D. What you need to know about SDN flow tables[C]//Passive and Active Measurement, New York, USA, 2015: 347-359

[6] He K, Khalid J, Gember-Jacobson A, et al. Measuring control plane latency in SDN-enabled switches[C]//Proceedings of the Symposium on SDN Research, New York, USA, 2015: 1-6

[7] Lazaris A, Taharad, Huang X, et al. Tango: simplifying SDN control with automatic switch property inference, abstraction, and optimization[C]//Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, New York, USA, 2014: 199-212

[8] Bifulco R, Matsiuk A. Towards scalable SDN switches: enabling faster flow table entries installation[C]//ACM Special Interest Group on Data Communication, London, UK, 2015: 343-344

[9] Chen H, Benson T. The case for making tight control plane latency guarantees in SDN switches[C]//Proceedings of the Symposium on SDN Research, New York, USA, 2017: 150-156

[10] Wang Y, Tai D Z, Zhang T, et al. Flow shadow: keeping update consistency in software-based OpenFlow switches[C]//The 24th International Symposium on Quality of Service, Beijing, China, 2016: 1-10

[11] Jeong H J, Song I S, Kwon T G, et al. A multi-dimension rule update in a TCAM-based high-performance network security system[C]//Proceedings of the 20th International Conference on Advanced Information Networking and Applications, Washington DC, USA, 2006: 62-66

[12] Wen X T, Yang B, Chen Y, et al. RuleTris: minimizing rule update latency for TCAM-based SDN switches[C]// The 36th International Conference on Distributed Computing Systems, Nara, Japan, 2016: 179-188

[13] He P, Zhang W, Guan H, et al. Partial order theory for fast TCAM updates[J]. *IEEE/ACM Transactions on Networking*, 2018, 26(1):217-230

[14] Qiu K, Yuan J, Zhao J, et al. Fast lookup is not enough: towards efficient and scalable flow entry updates for TCAM-based OpenFlow switches[C]//IEEE 38th International Conference on Distributed Computing Systems, Vienna, Austria, 2018: 918-928

[15] Kazemian P, Varghese G, Mckeown N. Header space analysis: static checking for networks[C]//The 9th USENIX Symposium on Networked Systems Design and Implementation, San Jose, USA, 2012: 113-126

[16] Banerjee T, Sahni S, Seetharaman G. PC-DUOS +: a TCAM architecture for packet classifiers [J]. *IEEE Transactions on Computers*, 2014, 63(6):1527-1540

[17] Reddy P M. Fast updating algorithm for TCAMs using prefix distribution prediction[C]//International Conference on Electronics and Information Engineering, Kyoto, Japan, 2010: 400-404

[18] Wang Z, Che H, Kumar M, et al. CoPTUA: consistent policy table update algorithm for TCAM without locking [J]. *IEEE Transactions on Computers*, 2004, 53(12): 1602-1614

Accelerating batch rule update scheme based on priority replacement

Ding Zixuan[*,**], Yu Jinping[*], Li Wenbin[*], Bi Jingping[*]

([*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

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

Abstract

Currently, the physical storage locations of rules in ternary content addressable memory(TCAM) at software-defined networking(SDN) switches depend on the priorities of the rules. Therefore, when updating rules in TCAM, a great quantity of rules will be moved to new locations due to the emergences of new priorities and the modifications of old priorities, resulting in a significantly high latency. To solve the problem, a fast method of batch rule updating scheme (BRUS) based on priority replacement is proposed, which reduces the latency while updating rules in TCAM. BRUS introduces the definition of rule dependency-based semantic consistency. Based on this, BRUS replaces the priorities of inserted rules with the priorities of deleted rules to avoid unnecessary rule movements, and therefore achieving fast rule updating. Experiment results show that BRUS is capable of finding more than 91% replace pairs and significantly reduces rule moves when updating. Compared with the state-of-the-art methods, BRUS is clearly superior in terms of stability and suitability in batch rule-update scenarios.

Key words: software-defined networking(SDN), ternary content addressable memory(TCAM) update, rule update, semantic consistency, rule dependence