|  |
| --- |
| 清华大学 |
| SDN技术调查报告 |
|  |

|  |
| --- |
|  |

小组成员及分工

**张豪 2017213815**

《Fast Filter Updates for Packet Classification using TCAM》

《Fast Updating Algorithms for TCAMs》

《PC-DUOS：Fast TCAM Lookup and Update for Packet Classifiers》

**崔浩 2018214160**

《CoVisor：A Compositional Hypervisor for Software-Defined Networks》

《Fast Lookup Is Not Enough：Towards Efficient and Scalable Flow Entry Updates for TCAM-based OpenFlow Switches》

《FastUp：Fast TCAM Flow Table Updates for SDN Switches》

**钟春蒙 2018214147**

《Fast Filter Updates for Packet Classification using TCAM》

《Partial Order Theory for Fast TCAM Updates》

《Fast Lookup Is Not Enough：Towards Efficient and Scalable Flow Entry Updates for TCAM-based OpenFlow Switches》

**张苏坤 2018214135**

《CoPTUA：Consistent Policy Table Update Algorithm for TCAM without Locking》

《RuleTris：Minimizing Rule Update Latency for TCAM-based SDN Switches》

《Fast incremental updates on Ternary-CAMs for routing lookups and packet classification》

目录

[CoVisor 3](#_Toc533172622)

[一、Covisor简介 3](#_Toc533172623)

[二、增量组合策略 5](#_Toc533172624)

[三、编译拓扑转换 7](#_Toc533172625)

[四、效率优化 9](#_Toc533172626)

[TCAM 10](#_Toc533172627)

[一、TCAM简介 10](#_Toc533172628)

[二、Fast Update Algorithms for TCAMs 10](#_Toc533172629)

[1. 插入新的路由表项 13](#_Toc533172630)

[2. 删除旧的路由表项 15](#_Toc533172631)

[三、Fast Filter Updates for Packet Classification using TCAM 15](#_Toc533172632)

[1. 使用闲置位存储优先级信息 17](#_Toc533172633)

[2. 使用GMR位剪枝 18](#_Toc533172634)

[四、Fast TCAM Lookup and Update for Packet Classifiers 18](#_Toc533172635)

[五、Partial order theory for fast tcam updates 21](#_Toc533172636)

[1. 普适TCAM更新流程 22](#_Toc533172637)

[2. TCAM更新的时间复杂度度量 24](#_Toc533172638)

[3. TCAM表项分割 27](#_Toc533172639)

[六、Towards Efficient and Scalable Flow Entry Updates for TCAM-based OpenFlow Switches 29](#_Toc533172640)

[1. TCAM规则更新框架 30](#_Toc533172641)

[2. TCAM更新流程 31](#_Toc533172642)

[3. BIT数据结构的应用 32](#_Toc533172643)

[4. 值的定义与计算 35](#_Toc533172644)

[5. TCAM中表项的不同位置分布 36](#_Toc533172645)

[6. 算法效果 38](#_Toc533172646)

[七、CoPTUA:Consistent Policy Table Update Algorithm for TCAM without Locking 39](#_Toc533172647)

[1. 一些有用的概念和数学表达式 40](#_Toc533172648)

[2. 问题的提出 41](#_Toc533172649)

[3. CoPTUA算法 42](#_Toc533172650)

[4. CoPTUA算法效果 46](#_Toc533172651)

[八、RuleTris: Minimizing Rule Update Latency for TCAM-based SDN Switches（最小化依赖） 47](#_Toc533172652)

[1. 问题的提出 48](#_Toc533172653)

[2. RuleTris 框架 48](#_Toc533172654)

[3. 生成DAG图的三种策略 50](#_Toc533172655)

[4. 后端优化器 54](#_Toc533172656)

[5. 算法效果 56](#_Toc533172657)

# CoVisor

## 一、Covisor简介

Covisor是一种网络管理程序。在一个网络里，使用不同语言编写的、部署在不同控制平台的控制器管理网络不同的区域，Covisor组合了这些控制器，使得它们能够协同工作，管理整个网络。Covisor还对网络拓扑进行抽象化，每个控制器看到的都是一个虚拟的网络拓扑，网络管理员可以指定每个控制器的权限。文章最主要的贡献是建立了一套高效率的算法来整合不同的控制器、把虚拟的网络拓扑编译为实际的物理网络拓扑，以及高效率地处理规则更新。

SDN的一个重要原则是解耦，管理员可以部署最适合需求的硬件和软件，而不是受各种兼容性限制，因此需要一个网络管理程序来组合这些孤立的控制器。这样的管理程序需要做到：

1. 集成多种不同的控制器
2. 定义抽象的网络拓扑，每个控制器都可以看到分配给自己的拓扑结构
3. 防止失效控制器对系统的影响，控制器有一定的访问权限控制，不能访问或修改某些字段

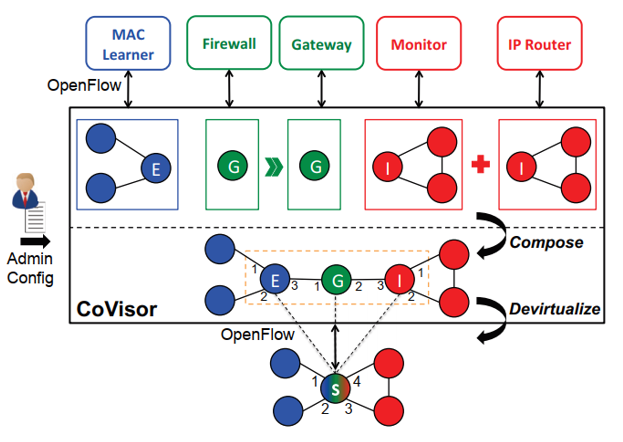
现有技术的主要挑战是效率问题，文章定义了一系列算法来解决这些问题，文章的主要贡献包括：

1. 定义了组合多种控制器的管理程序的体系架构
2. 设计新的并行、串行和覆盖算法来组合控制器的操作
3. 设计新的增量算法来把为虚拟拓扑编写的规则编译为实际物理交换机的规则
4. 使用自定义的数据结构来进行访问控制

为了实现以上特征，Covisor主要分为两个阶段：

1. 整合孤立控制器的策略，变成全局的组合策略
2. 把这个组合策略编译为物理网络能理解的策略

Covior的架构图如下图所示：



Covisor接收管理者的网络配置，这些配置定义了如何集成不同控制器的策略（是先运行A，再运行B，还是A与B共同执行，还是A的规则覆盖B的规则），还定义了每个控制器的虚拟拓扑的创建方式，以及每个控制器的访问权限。

对控制器组合策略的配置包括：

Actions：最基本的包组合策略就是原子的包处理操作Action，如丢包操作drop，指定从某个端口转发的操作fwd(3)等等。

Parallel operator(+)：并行操作符，相当于把一个包复制两份，同时给控制器T1和T2执行，再将计算的结果进行综合。

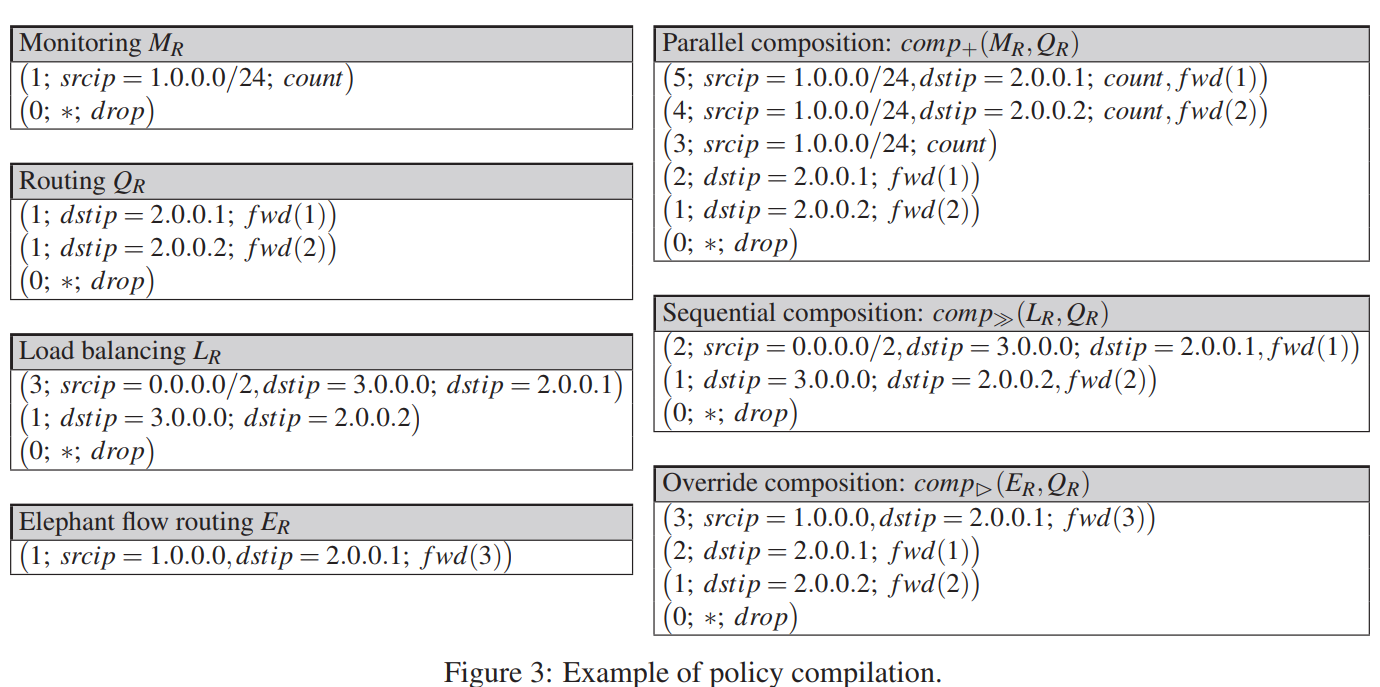
Sequential operator (>>)：串行操作符，一个控制器在处理完以后再给下一个控制器进行处理。

Override operator ()：优先级更高的规则优先执行

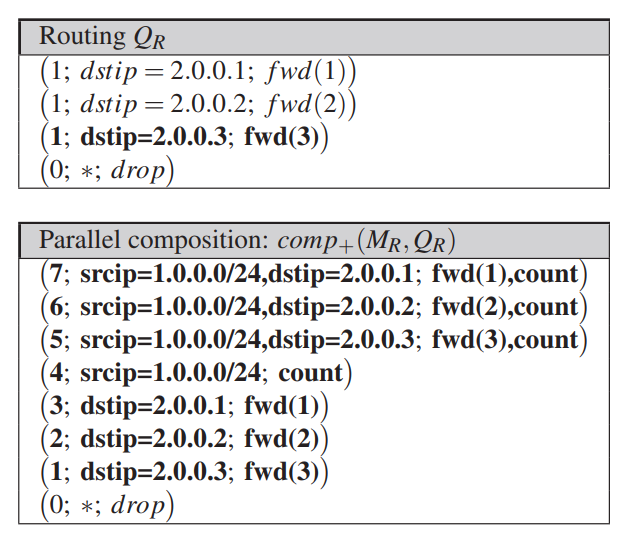
对控制器的限制包括：

1. 每个控制器只能看到虚拟的网络拓扑，不同控制器看到的网络拓扑是不一样的
2. 每个控制器只有权限访问部分包信息

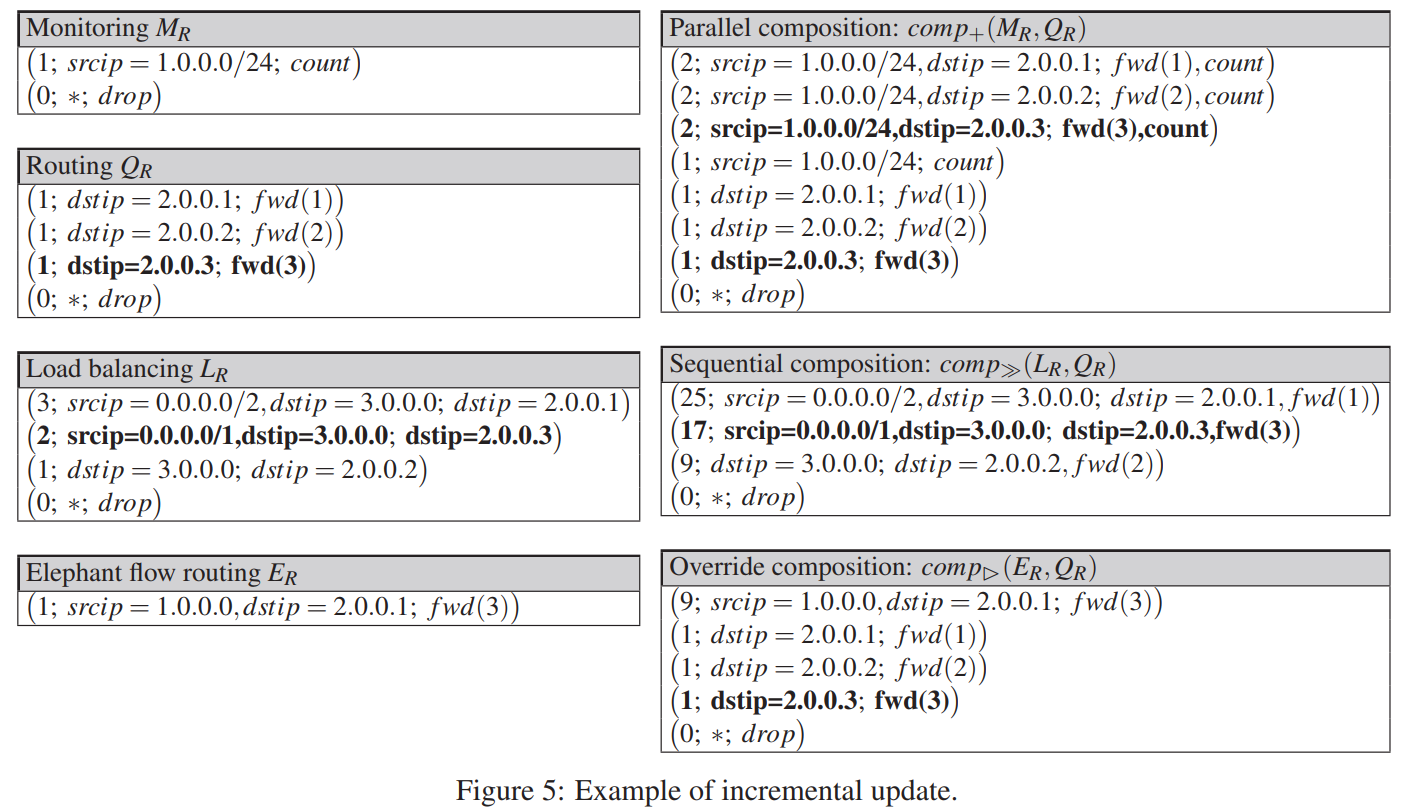
## 二、增量组合策略



对于并行操作，需要两两组合两个控制器规则T1和T2中的所有规则，将这些组合的规则放在更高的优先级上。对于串行操作，需要检查第一个控制器规则T1中的规则，将对应的规则与T2中的规则进行合并。对于覆盖操作，需要将第一个控制器规则T1的所有规则放在更高优先级的位置。

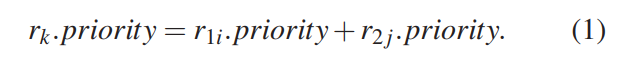


如果只是顺序从0开始到高位依次排列优先级，在规则更新时会造成很大的负担。如下图所示，在QR增加一条新规则时，集成规则集MR+QR中大部分的组合规则的优先级都需要更新，这种方法需要的计算量太大。因此需要一种新的计算优先级的方法。



如图所示，文章这样定义组合规则的优先级：

如果是并行操作，那么优先级定义为：



如果是串行操作，那么优先级定义为：

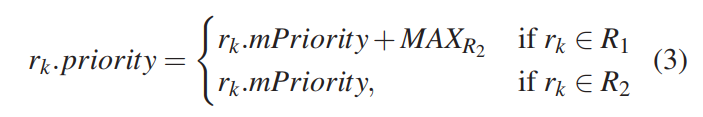


其中小圆圈代表了拼接操作，也就是直接把两个优先级的数字当成字符串拼接在一起。但是这样的拼接是在一定的进制下进行的，在10进制下，1和2拼接为12，在8进制下，1和2拼接为1\*8+2=10。因此优先级的实际定义为：



其中MAXR2指的是第二个规则集T2的最大规则数量

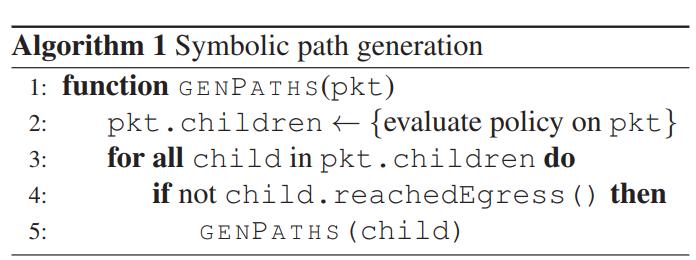
如果是覆盖操作，那么优先级的定义为



使用这种方法定义优先级，在更新到来是只需要更改几个响应的规则，其他规则的内容和优先级都不会有影响.

## 三、编译拓扑转换

文章重点讨论了一对多的转换，也就是一个物理交换机对应多个虚拟交换机的转换。算法需要从多组虚拟规则中编译出实际物理设备需要的表项。



首先从一个入口开始发送符号包如果这个包没有到达出口，则递归寻找子结点。



例如，对于A、B、C三个规则集，首先发送两个符号包：P1：2.0.0.0/16 ，P2：1.0.0.0/8，其中P2没有到达出口，所以生成P21和P22， P21在B中匹配第一个规则，从2离开，P22在C中匹配第一个规则，从3离开。

由此生成三条路径：

P1： A1

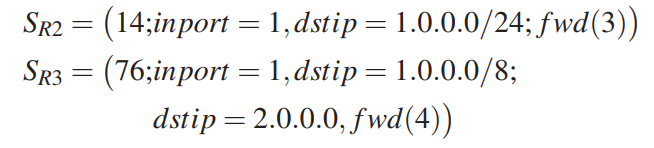
P21 ：A2 → B1

P22 ：A2 → B2 → C1

通过这些路径来进行规则集的合并，如A2 → B1合并算法为comp>>(AR2,BR1)，因为路径是串行操作，类似的，A2 → B2 → C1合并算法为comp>>(comp>>(AR2,BR2),CR1)。

最终得到的物理设备的规则集为：





需要注意的是优先级的重新确定，在进行串行操作合并时，优先级的分配方式为comp>>的分配方式，也是就是，这种分配方式是错误的。SR3因为进行了三次合并，所以优先级更高，而实际上应该是SR2优先级更高，因此需要使用虚拟拓扑中虚拟交换机的数量作为指定长度，不足这个长度的优先级拼接需要补0，如。

## 四、效率优化

在组合两个规则集时，通常是一个n\*n的问题，方法复杂度为O(n2)，但是如果知道R1、R2的结构，就可以忽略一些结构不同的规则的组合，提高组合的速度。



在组合两个规则集时，可以对规则中的每个字段建立索引：

* 如果是精确匹配：建立hash索引
* 如果是前缀匹配：建立字典树索引
* 如果是通配符匹配：建立链表索引

# TCAM

## 一、TCAM简介

当收到一个数据包后，路由器需要根据其目的地址所匹配的路由表项来决定将该数据包送往哪个端口。无类别域间路由技术（CIDR，classless interdomain routing）将这一做法改为寻找与数据包目的地址拥有相同前缀的路由表项，但这样做可能会出现一个数据包同时与多条路由表项相匹配的情况，此时我们认为拥有最长前缀匹配的路由表项具有最高的优先级。随着网络带宽的不断提升，路由器处理数据包的延时便成为了提升网络整体性能的瓶颈。为提高路由器处理数据包的效率，人们开始尝试将简单的线性搜索、关联搜索等算法在硬件上实现，其中，三态内容寻址存储器（TCAM，ternary content-addressable memory）得到了广泛的商业应用。

## 二、Fast Update Algorithms for TCAMs

TCAM中的每个晶元都拥有三种状态：0,1，或者 “不关心”。利用第三种状态，TCAM可以灵活设置每条路由表项待匹配前缀的长度，或者直接将其设置为通配表项。TCAM将所有的路由表项依照优先级的顺序存储，这样首次匹配成功的路由表项即是数据包的最终匹配表项。采用这种方法，TCAM很好地解决了CIDR匹配中的优先级问题，这使得TCAM在进行数据包的路由表查询时非常快，但也带来了新的问题，当路由表发生改变，如增加或者删除了某些路由表项时，TCAM需要维护路由表项之间的优先级。关于这个问题，最简单的解决办法是在找到新表项的正确位置后移动所有的后续表项：

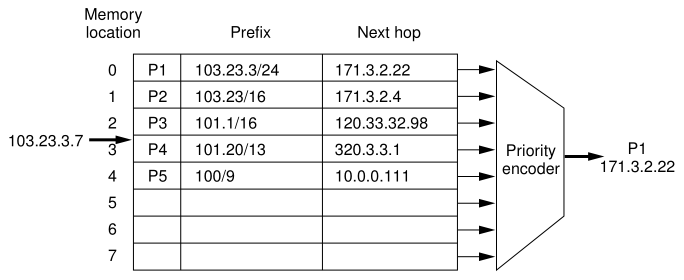


图 2-1

如图 2-1，当新的路由表项插入3号内存区后，P4、P5两条表项均需要向下移动，显然，这种方法需要次（N为路由表的总条数）内存拷贝操作。在此基础上的一种改进措施是将预留空间分散化，如图2-2

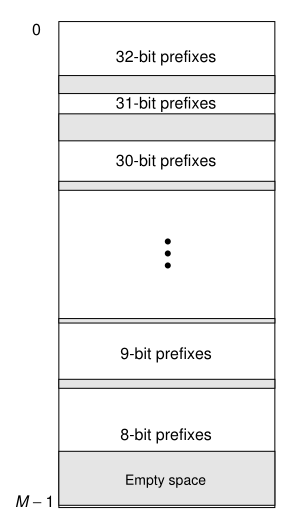


图2-2

但分散地预留空间会浪费大量的内存空间，而且这种做法在最坏情况下也需要移动次内存。一个更好的做法是，我们可以将预留的内存空间放置在中部，如图2-3

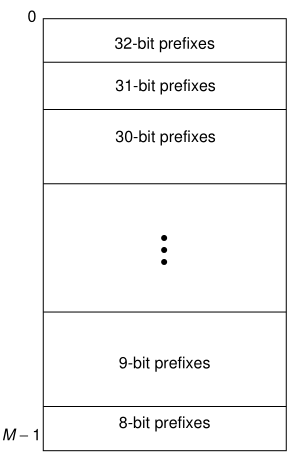


图2-3

这样一来我们需要的内存移动次数仅为图2-1中的一半，将集中化的预留空间也避免了内存空间的大量浪费。这或许对TCAM提升维护路由表的有序性的性能有所帮助，但还不够。其实我们只需要维护具有相同前缀的路由表项间的优先级。基于这一发现，D Shan等人提出了CAO\_OPT算法。该算法利用里字典树（Trie）来组织路由表项。而且仍旧将预留内容置于中部，如图2-4

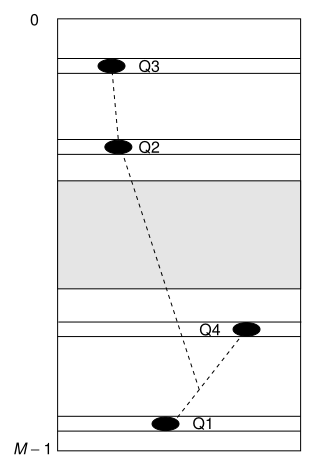


图2-4

鉴于字典树结构的特性，具有相同前缀的结点均在同一条路径上，因此当发生路由表项更改时，我们只需要维护字典树中一条上的结点顺序。下面来看一下具体的操作过程：

### 插入新的路由表项

当插入的新表项位于预留区上方（较小地址区）时

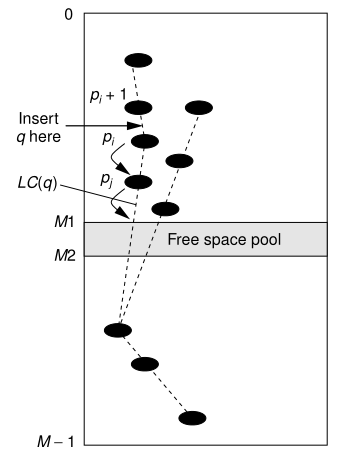


图2-5

我们只需要将从该位置原结点到其根路径上最邻近预留内存区的结点范围内的路由表项进行移动（如图2-5），便可以完成路由表项优先级的维护工作。

当插入的新表项位于预留区下方（较大地址区）时

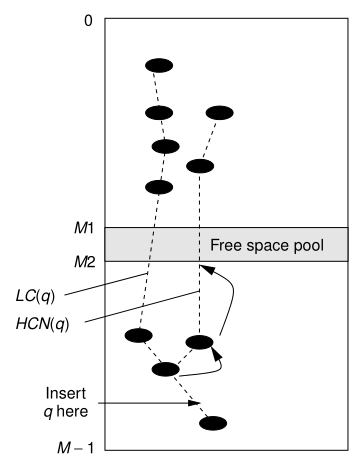


图2-6

在找到新结点应该插入的位置后，我们每次选取一个离此结点最近的子结点，以此后推，直至遇到预留的内存区域，然后调整这些结点的位置以维护结点的有序性（如图2-6）。

### 删除旧的路由表项

删除路由表项可以看做在对应的位置插入一条“空白”的路由表项，再将该“空白”路由表项依照上述插入过程移动至中间的预留内存区域。

可以看到，利用字典树结构组织起来的路由表项在维护有序性时只需要移动次内存（D是字典树中根到叶节点的最大路径长度）。

## 三、Fast Filter Updates for Packet Classification using TCAM

使用字典树来应对以最长前缀匹配为优先规则的路由表项的更新问题看起来是一个不错的办法。但这种优先规则并不通用，在应对具有一般性的优先级的路由表更新问题上，Haoyu Song等给出了一些解决方案[2]。同样的，这里也将 路由表项在TCAM中存储的位置作为其优先级，存储在地址编号更小的路由表项具有更高的优先级，在进行路由查找时会被先检索到。注意，即使是考虑一般化的优先级，优先规则仍旧只会在具有相同前缀的路由表项间有意义，只不过相比于最长前缀匹配，此时较短的前缀匹配可能具有更高的优先级。Haoyu Song等给出的方法是将路由表项依照优先级划为若干集合，统一集合内部的路由表项不存在匹配冲突，可以随意调换位置，而不同的集合之间存在匹配冲突，不能调换位置。图3-1所示的情况是一个具体的例子：

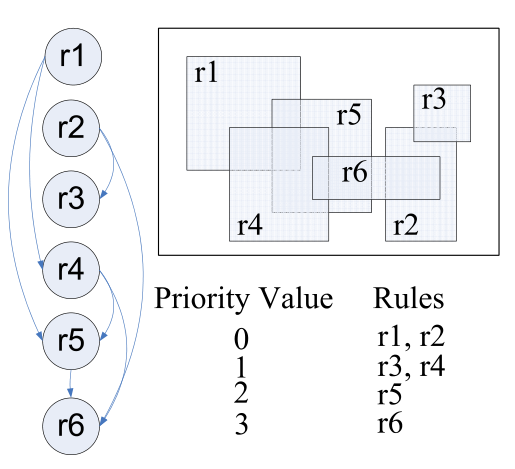


图3-1

图3-1中每个结点代表一条路由表项，用带箭头的线连接的两个结点具有匹配冲突，其箭头指向优先级更低的结点，如r1与r4有相同的前缀，存在匹配冲突，且r1的优先级高于r4，r1存储在地址编号更小的内存段。标识出路由表中所有的冲突关系后，我们可以根据这幅有向无环图给所有的路由表项设置一个具体的优先级的值。在图7中，先给没有任何入度的结点赋值，即将r1、r2的优先级赋值为0，然后将r3、r4的优先级赋值为1，依次类推。给所有的路由表项的优先级都赋值后，具有相同优先级值的路由表项便划分为一个集合。实践发现，优先级不同的值的个数远远小于路由规则的条数，如图3-2：

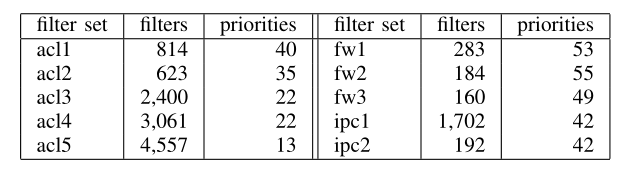


图3-2

因此，将处理不同的路由表项转变为处理不同的优先级将带来很大的方便。Haoyu Song等还给出了针对优先级的搜索方案，图3-3描述了如何搜索一个具有32个不同的优先级值的路由表：

### 使用闲置位存储优先级信息

TCAM可以提供144比特来与报文头部进行匹配，但是并不是全部144比特都用到了，还有一些位没有用到。匹配项的优先级关系只存在于有包含关系的匹配项之间，而且这些匹配项的有效长度一定不相同，在一般的匹配项集中，优先级不会超过64种，因此事实上优先级信息可以用较少的几个比特位来进行存储。

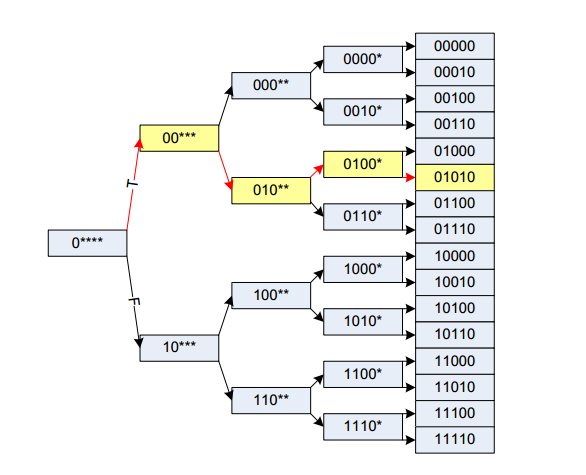


图3-3

先行搜索优先级范围（0b0xxxx）。若发现匹配的路由表项，则继续搜索（0b00xxx）；否则搜索（0b10xxx），依次类推。

使用TCAM中没有用到的位来存储优先级信息，因此优先级信息不再与表项的物理存储位置相关，表项的删除、添加和更新操作都可以在任意物理位置进行，且不需要对已有的表项进行移动。结合GMR的特性，每次查询时，将可能的优先级范围进行二分，递归进行，可以在O(log n)时间内完成查询，相比于原本的线性查找O(n)有了较大提升。

### 使用GMR位剪枝

TCAM中有一系列可以重复配置的Global Mask Registers (GMR)，这些寄存器可以按照位信息忽略一下匹配项。multi-match output 信号指示是否有多个匹配表项，如果这个信号无效，那么表示只有一个匹配表项，那么在找到一个匹配项之后就不用继续查询了，因为不可能有更高优先级的匹配项。如果multi-match output信号无效，那么表示只有一个匹配表项，那么在找到一个匹配项之后，就不需要进行进一步的递归查询。

## 四、Fast TCAM Lookup and Update for Packet Classifiers

在处理一般化的优先级问题上，Tania Mishra等提出了基于DUOS改进的PC-DUOS[3]方法。这里同时使用了两块TCAM，如图4-1

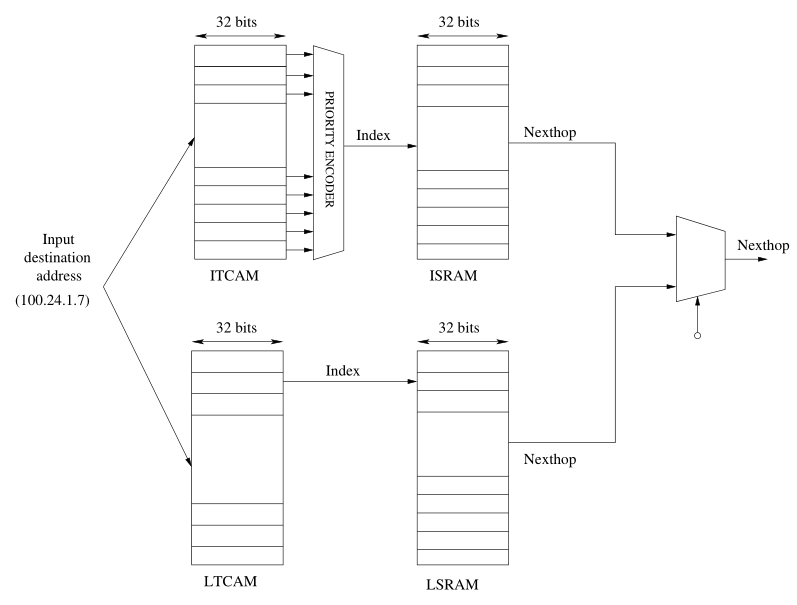


图4-1

每块TCAM均有一块SRAM与之相连，TCAM中存贮的是路由表项中待匹配的前缀，而SRAM中存储是与之相对应的动作。两块TCAM分别为ITCAM（Interior TCAM）和LTCAM（Leaf TCAM）。DUOS也使用字典树来管理路由表项的前缀，将字典树中的叶节点对应的路由表项存储在LTCAM中，将其余的叶节点对应的路由表项存储在ITCAM中。因此，LTCAM中存储的路由表项间不存在前缀匹配冲突，所以LTCAM中的路由表现也不需要优先级。在处理数据包时，同时查询ITCAM和LTCAM，如果在LTCAM中查找到了相匹配的路由表项，则直接将该路由表项最为最终匹配结果，终止ITCAM查询。

一旦我们在LTCAM中找到了相匹配的路由表箱，则立即将其作为最终匹配结果，立即终止在ITCAM中的查找。为了保证这种查找策略的正确性，我们必须保证如下两点：

1. 每个数据包在LTCAM中多有只能有一条相匹配的路由表项。
2. 若一个数据包在LTCAM中匹配到一条路由表项，这条路由表项必须是该数据包在路由表中能匹配到的最高优先级路由表项。

对于一条由以下域组成的路由表项

PC-DUOS按照每个域对应一颗字典树，将路由表组织成一棵多级字典树。为了使查找性能达到最优，应该将路由表中的最大前缀不相交集放入LTCAM中，但寻找这一集合并不容易，权衡之下，PC-DUOS改为寻找一个较大的前缀不相交集——“叶叶集”，如图4-2

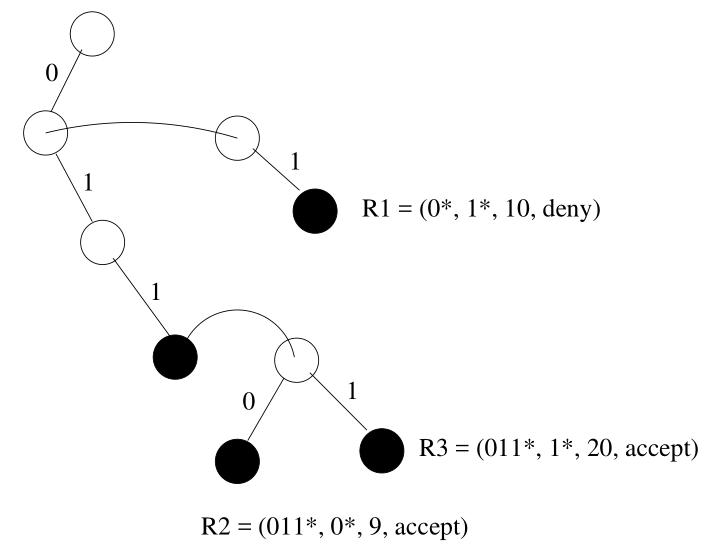


图4-2

图4-2中描述了路由表项的两个域，目的地址和源地址。R1、R2、R3在源地址字典树中位于叶子节点，但为了保证条件1的满足，PC-DUOS将R1对应的路由表项置于ITCAM中，由于其所在字典树的根节点不是从上一级字典树的叶节点派生出来的。另外，为了保证条件2的满足，PC-DUOS将R3对应的路由表项也置于ITCAM中，尽管其所在的字典树的根节点是从上一级字典树的叶节点派生出来的，但由于其优先级小于已经置于ITCAM中的R1节点，且与R1有前缀匹配冲突。故最终只有R2被置于LTCAM中。

LTCAM中的路由表项无需优先级编号，对ITCAM中的表项，PC-DUOS采取了和[2]中同样的方法。

## 五、Partial order theory for fast tcam updates

本文的主要贡献是

1.基于部分排序理论，提出了TCAM中增加表项的一般性过程，弄清楚了TCAM表项更新的理论下界，为今后的所有研究确立的目标；

2.提出了使用规则集分裂的方法来进一步加速TCAM更新

本文针对的问题场景是

TCAM更新算法的时间复杂度没有理论指导；TCAM表项更新时间代价大。

### 普适TCAM更新流程

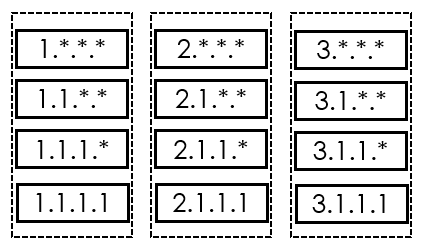


图5-1

TCAM中的规则集可以分为若干组，组内的规则之间具有优先级关系，组间的规则没有优先级关系

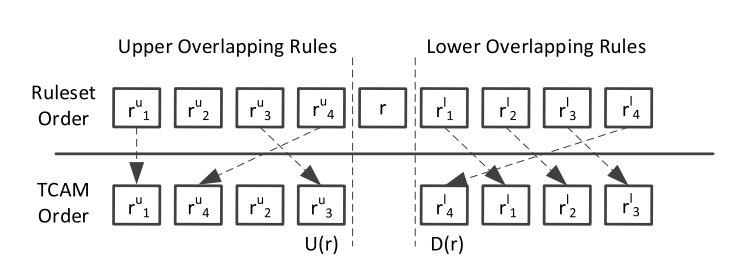


图5-2

插入任何一个规则r时，都只能找到唯一一个可能与r具有优先级关系的组。

在这个组当中，按照优先级排序后，紧挨着r的两个规则分别是U(r)和D(R)

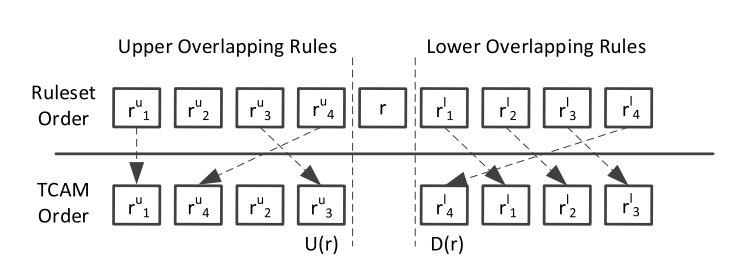


图5-3

在找到U(r)和D(r)之后，这两个规则之间的所有物理位置，都可以用来存储r，将这个合适的物理位置范围成为交换区域。

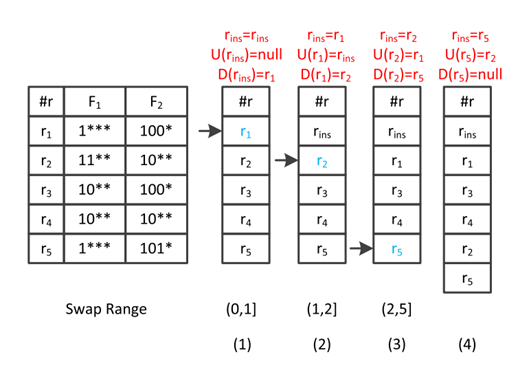


图5-4

如果交换区域中有空闲的位置，那么可以直接将r放在这个空闲位置上；否则需要用r替换这个区域之间的某一个规则r’，然后对r’执行插入程序。

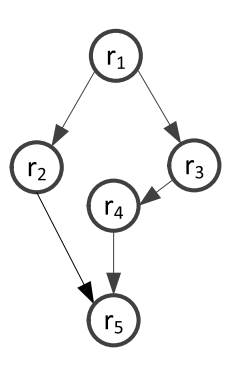


图5-5

如果交换范围中没有空闲位置，那么需要替换其中的一个规则r’，然后对r’进行插入，直到r’找到一个空闲位置。一般是不断往后找，也就是替换优先级较高的规则，最终一定会变成一个组中优先级最高的规则，然后直接放在TCAM的末尾。

### 2. TCAM更新的时间复杂度度量

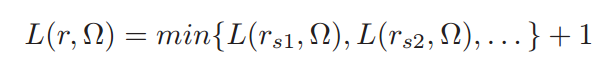
#### 1）插入规则的交换链

从以上分析可以看出，TCAM更新的主要操作，就是表项的替换，而替换的次数与表项插入循环的次数有关。

插入规则r，替换规则r’；插入r’时，可能还会继续替换并插入r’’，如此循环，最终插入过程会形成一个插入链，即r——r’——r’’——r’’’……交换链的长度就是TCAM表项替换操作的次数。也就是说，要减少TCAM的更新时延，就是要尽量使得交换链的长度减小。

因此，在交换范围中选择r’的原则，就是尽量减少替换的次数，也就是交换链的长度。如图r1——r2——r5是 插入r1的最短交换链。要选出交换链长度最短的r’，就要首先计算出每一个规则的交换链长度，然后遍历选出最短的。

#### 2）交换链长度的计算



计算方法每一个规则的交换链长度，可以使用动态规划算法：

先给最末尾的（所有优先级最高的）规则赋予cost 1；其他规则，找它们交换范围中代价最小的规则+1；

#### 3）最优替换规则的选择

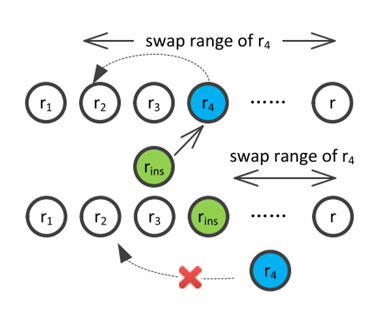


图5-6

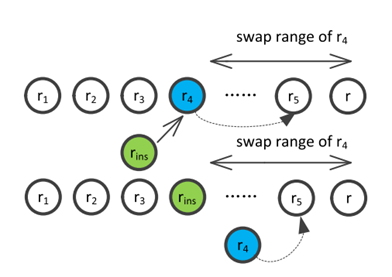


图5-7

在交换范围中找具有最短交换链长度的r’，一共有三种策略，分别是full，bh，down：

Full策略：在整个交换范围中找合适的r’

Bh策略：将交换范围分成两半，在优先级较高的一半中找r’

Down策略：直接使用D(r)，也就是交换范围中优先级最高的规则

Full策略实际上是不可行，因为会使得原有的规则顺序不稳定，规则的替换会改变现有规则的交换链长度，也就无法预先计算所有节点的交换链长度。Bh策略比目前（2018年）认为的最优算法还要高效，但是计算代价大，也无法实际运用。目前（2018年）认为最优的TreeCAM算法，就是down策略的一种实现。

### 3. TCAM表项分割

#### 1）表项分割思想

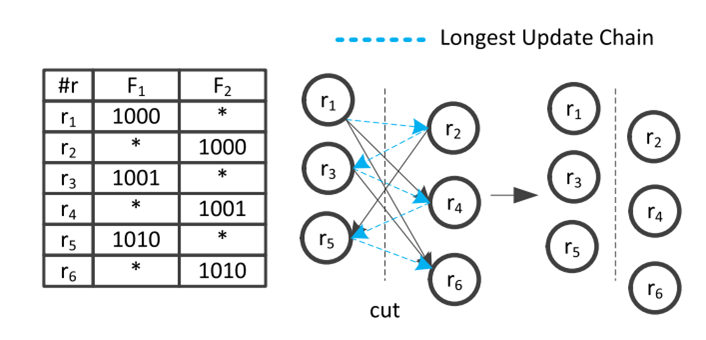


图5-8

TCAM更新代价高的原因之一，是优先级约束太多，减少优先级约束的一个方法，是划分规则集。

划分规则集的主要思想是，将一个完整的规则集，划分成若干份，分别存储在不同的TCAM上，然后并行的进行查找，最后在所有查找结果中选择优先级最高的。这样可以加快表项查找时间，并且可以减少每一个TCAM中的更新时延，。

如图，如果将r1，r3，r5分为一组，r2，r4，r6分为一组，那么组内的所有规则之间就完全没有优先级关系，可以在O(1)时间内完成插入。

#### 2）表项划分算法：BW split

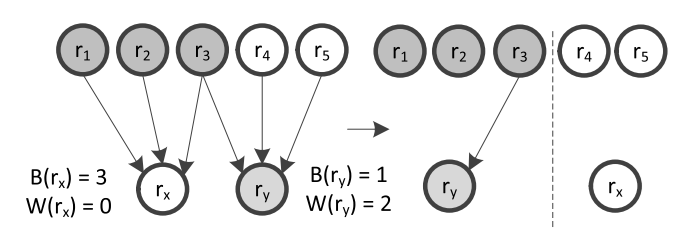


图5-9

将规则集进行二分时，可以使用黑白划分方法：

制作一个Hasse 图，在这个图中，节点代表规则，边从低优先级节点指向紧邻的高优先级节点。首先给根节点涂黑色；对于其他节点，统计其所有父节点（指向它的边的源节点）颜色，它的颜色与大多数父亲节点（多数投票）的颜色相反。

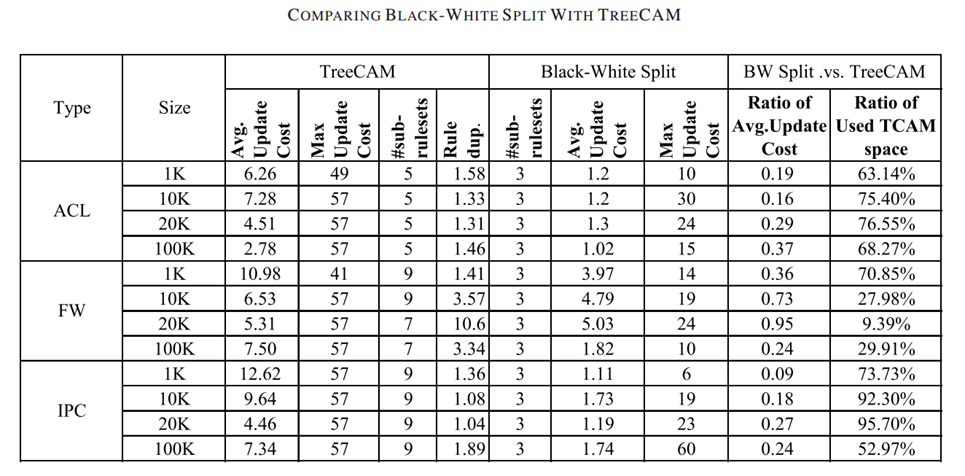


图5-10

从实际验证来看，BW split在平均更新代价和TCAM空间使用上，都比TreeCAM要好。

## 六、Towards Efficient and Scalable Flow Entry Updates for TCAM-based OpenFlow Switches

本文提出了一种使用贪心算法，BIT数据结构，有向无环图的，适用于各种部署情景的更新算法。

本文的问题场景是，当前TCAM更新算法时间复杂度高。

### TCAM规则更新框架

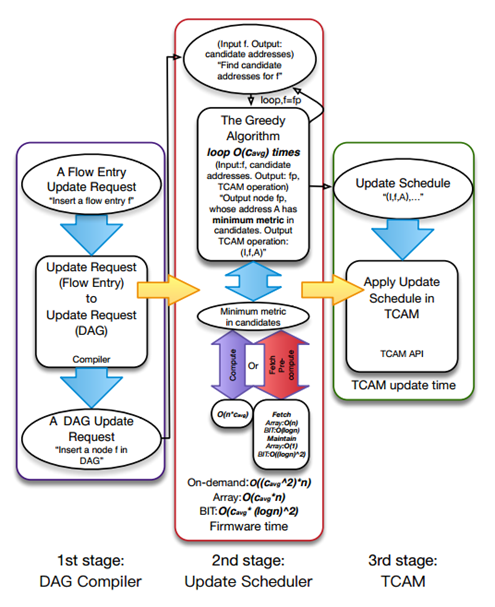


图6-1

目前的TCAM更新算法的框架大体如下：

1.将更新命令转化为有向无环图中的插入操作，完成这一部分功能的模块被称为DAG编译器。

2.将有向无环图中的插入操作翻译为TCAM交换命令，完成这一部分功能的模块被称为更新调度器。

3.在TCAM中执行2中的命令序列，命令序列决定了更新时延。

### 2. TCAM更新流程

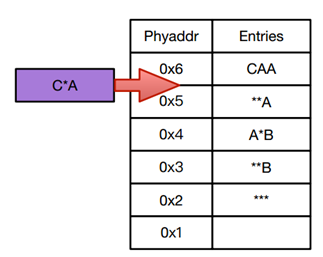


图6-2

TCAM中的部分表项有优先级顺序，并且表现为物理位置上的先后关系，在执行插入f的操作时，必须要维护这种优先级关系。

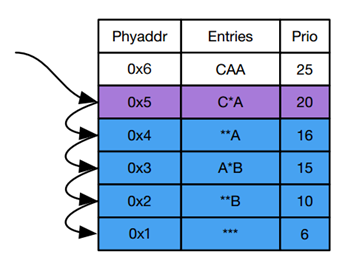


图6-3

具体方法是：

1.找到fa和fb，这是在将所有相关的规则排序之后，f两侧紧邻的规则

2.在fa和fb对应的物理位置之间的区域里，选择一个合适规则f’，和f替换

3.循环插入f’

### 3. BIT数据结构的应用

为了减少交换链的长度，在每一次选择替换规则的时候，我们都要选择交换链最短的规则。

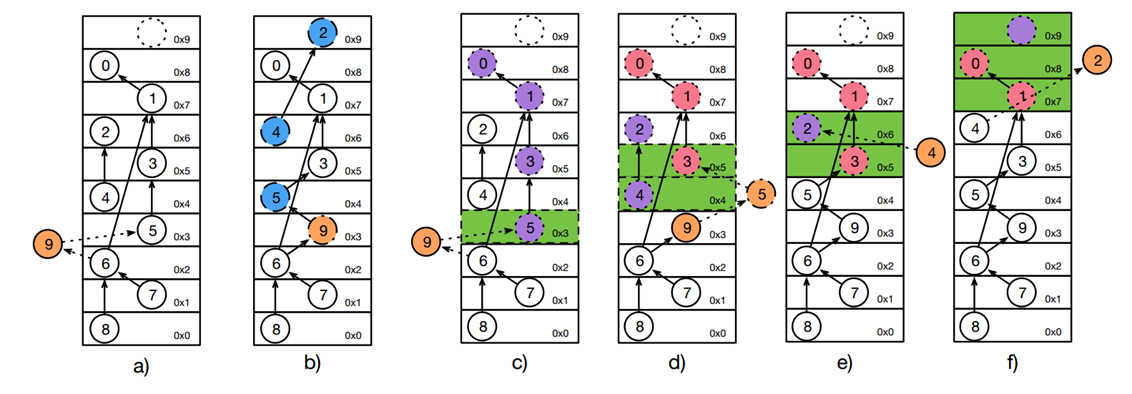


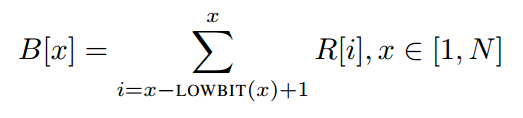
图6-4

绿色区域是交换区域，蓝色节点是交换链上的节点

在交换区域中选择节点时，使用了贪心算法，即选择交换链长度（M）最短的节点



在交换区域内选择合适的规则的时候，可以使用BIT数据结构减少这一过程的时间负责度。

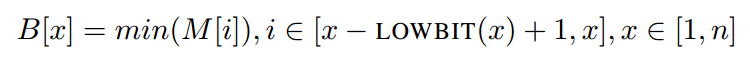


选择交换区域中M值最小的节点，如果使用朴素遍历算法，是一个耗时为O(N)的过程，如果采用BIT数据结构可以降低为O(log N)。

BIT数据结构可以用来在O(log N)时间内计算下标某一个范围[1,b]内所有元素的和。思想是将每一个下标按照二进制特点进行划分，使得每一个元素存储的不是自身的值，而是某一个下标范文内所有元素的和。具体来讲，B[x]存储索引在[x，(x-LOWBIT(x) + 1)]范围内的值的和。

举例：

所以：



在算法的实际应用中，B[x]不再存储范围内元素的和，而是存储最小值。如下图

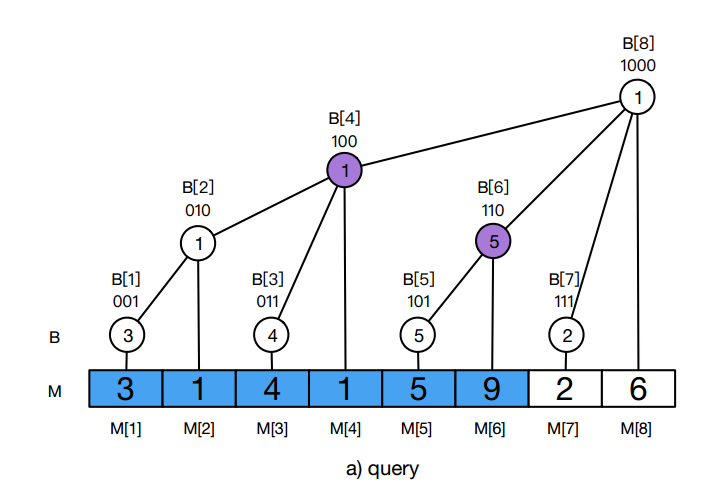


图6-5

### 4. 值的定义与计算

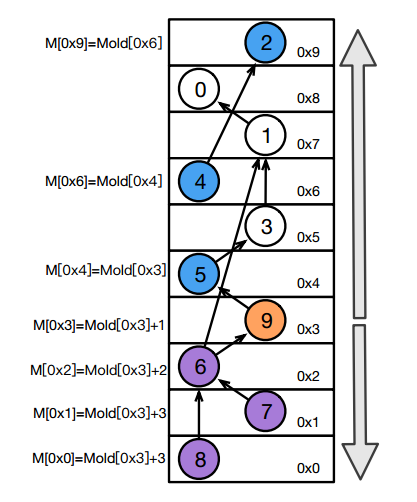


图6-6

在fa和fb之间选择合适的f’时，用到了度量值M，M(A)是插入A所需的循环次数。

1.根据优先级关系制作有向无环图，有向边代表优先级关系；

2.首次计算M值时，使用DFS算法，从每一个节点出发，搜索所有邻居；

3.在之后的所有操作中，只对M值进行更新，而不进行重新计算。更新的范围是交换链上的所有节点，以及直接或者间接与交换链上节点有优先级关系的节点

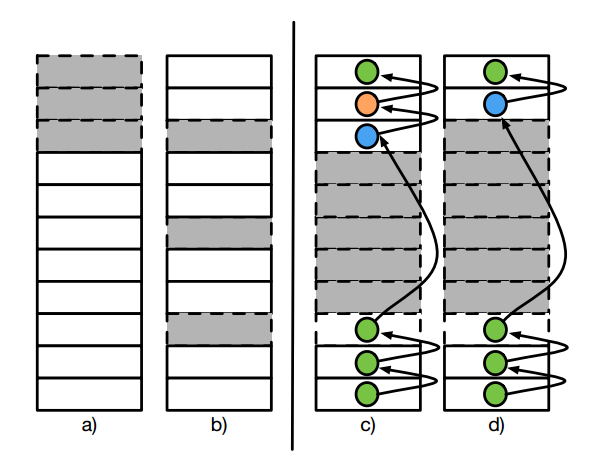


图6-7

### 5. TCAM中表项的不同位置分布

TCAM中表项的物理位置分布大致有三种：

1.全部在顶部或者全部在底部；

2.每隔一段距离留若干空位置；

3.分布在顶部和底部，且数量均衡；

1是最普通的情形，算法直接可以应用；2情形很难一直保持下去，也就是说，经过若干次插入操作，中间的空位置可能被填满了；3中的情形可以降低插入复杂度，但是增加表项删除的复杂度。

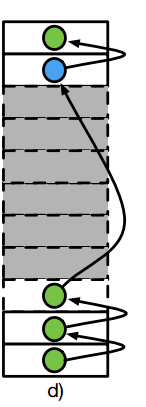


图6-8

第三种物理位置分布，即表项均衡的分布在顶部和底部，增添和删除表项的操作如下：

增添表项：

如果f两边（优先级顺序）的两个规则fa，fb，都在顶部，或者都在底部，那么就按照之前的方法插入顶部或者底部，如果fa，fb分别在顶部和底部，那么f就直接插入中间的空位置，如果顶部的规则较多，就插入靠下的位置，反之即反

删除表项：

脏删除：删除之后不进行任何处理

平衡删除：删除之后用其他表项填补空缺，维持所有表项都存储在顶部和底部的性

### 6. 算法效果

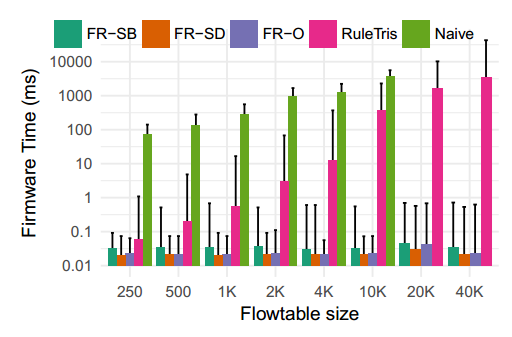


图6-9

FR-SB：两侧分布 + FastRule + 平衡删除策略

FR-SD：两侧分布 + FastRule + 脏删除

FR-O：一般分布

RuleTris：当前广泛使用的算法

Naïve：朴素算法

## 七、CoPTUA:Consistent Policy Table Update Algorithm for TCAM without Locking

由于确定性和快速查找性能，TCAM最近在用于高速网络中的分组分类的一般策略过滤（PF）中越来越流行。然而，PF表更新对有效使用TCAM提出了重大挑战。为避免错误和不一致的规则匹配，传统方法是在规则更新期间锁定PF表，但表锁定会对数据路径处理产生负面影响。

在本文中，我们提出了一种新的方案，称为TCAM的一致性策略表更新算法（CoPTUA）。CoPTUA不是最小化规则移动次数以减少锁定时间，而是在整个更新过程中维护一致的PF表，从而无需锁定PF表，同时确保规则匹配的正确性。我们的分析和仿真表明，即使对于具有100,000个规则的PF表，如果只有2%的PF数目为空，在最坏的情况下，可以在1秒内同时更新任意数量的规则。因此，CoPTUA可以在不到1秒的时间内强制执行任何新规则，同时具有高内存利用率且不影响数据路径处理。

### 一些有用的概念和数学表达式

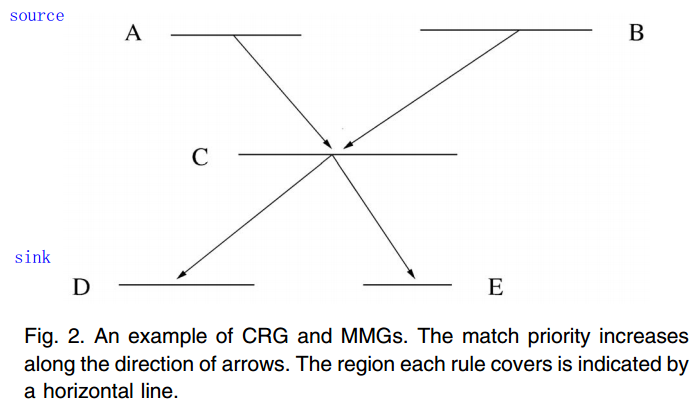


图7-1

* 规则空间(Rule space): b bits被称为b维规则空间。 Eg：IPV4 32bit
* 规则重叠（Rule overlapping）。 Eg：A和C拥有规则重叠。
* 超集和子集规则（Superset and Subset rules）。Eg：C是E的超集。
* 局部重叠（Partially overlapping）:

Eg:规则1x and x0被10局部重叠.

* A -> C:A比C的匹配优先级低。
* A < E: A比E的匹配优先级地址低。
* 连通规则（Connected rules）: Eg: A&C&D , B&C&E
* Connected Rule Graph (CRG):
* 源(宿)叶规则（Source (sink) leaf rule）:

Eg : source A 、B Eg : sink D 、E

* Multiple Match Group(MMG): 任何源叶规则到任何宿叶规则的一条路。
* 无关规则（Independent rules）: Eg： A&B ，D&E

### 问题的提出

一个规则可以出现在多个MMG中，但是只会出现在一个CRG。我们可以看出，MMG中的规则必须按顺序安放在TCAM中，而独立规则可以安插在TCAM中的任意位置。

所有MMG中的规则都有一个**超集和子集规则**（Superset and Subset rules）的关系。超集规则的前缀索引一定比子集短。所以，一条MMG中最大的数量是**规则空间**的长度b。

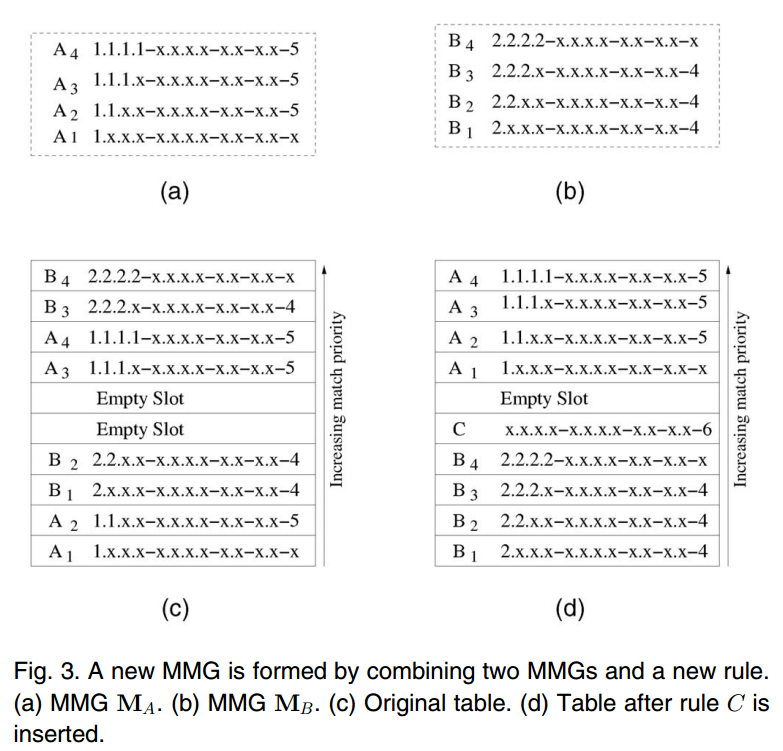


图7-2

在之前的算法中，我们在TCAM的表中间维护一个empty slot，这使得移动数目最坏得情况不超过MMG.length/2。1）在最坏情况下，MMG.length=表长度。2）有时候加入一条规则，可能会使两个MMG合并成一个更大的MMG，且依赖顺序改变，如上图。这使得在表空间中放置空位，在最坏得情况下，不会减少移动的规则数量，最高可以达到Nr/2(Nr为非空的规则)。**在这种情况下，如果继续采用上锁的方法，对TCAM的性能有巨大的影响。**

### 3. CoPTUA算法

【目的】在整个更新过程中维护一致的PF表，从而无需锁定PF表，同时确保一致的(consistent)和无错的（error-free）的规则匹配。

* **Consistency rule matching：**对于每个规则移动、添加、删除，搜索关键字匹配的规则与操作之前匹配的规则相同。
* **Error-free rule matching：** 匹配前后的规则对应的action没有更新。

【硬件能力要求】

1. 每个TCAM规则条目都有一个与之关联的有效位。 要激活规则条目，需要设置此有效位。否则，规则条目永远不会被匹配。
2. 匹配规则后，重置有效位对操作返回过程没有影响。 换句话说，如果在删除操作之前发生该规则的匹配，则删除规则不能停止将该规则的动作返回。
3. 在两个连续的部分密钥匹配的规则，将优先较高的规则有效位设为无效，会匹配到第二个规则。
4. TCAM是双端口，可同时从本地CPU和网络处理器访问。

【对一条规则更新】

* Step1：删除过程。将原来存在的规则的有效位设为非法。
* Step2：写入过程。将要添加的新规则写入其对应的位置。
* Step3：将新规则的有效的设为合法。

【添加一条新规则】

* Step1：删除过程。将原来存在的规则的有效位设为非法。
* Step2：写入过程。将要添加的新规则写入其对应的位置。

【对一条规则移动】

* Step1：添加过程。按照上述的【添加规则】将该规则写入新的位置。
* Step2：删除过程。将旧规则从原有位置上删除。

我们可以证明，在PF表中按上述步骤进行移动时，不会影响PF表的一致性。（证明详见论文4.3）

【算法具体操作】

（a）具有Nr规则和Ne空托管的原始表，对一批规则进行更新。

（b）删除所有未出现在最终配置中的规则，所有相关规则都标记为“o”。

（c）Ne个具有最低匹配优先级的**相关规则**被移动到底部的空条目中。

（d）其余的**相关规则**被移至顶部。

（e）迭代重新排列top的相关规则，将它们放在所有Ne空条目下面（不包括分配给新规则的条目）。所有相关规则都是有序的。

（f）将**相关规则**移至顶部（底部）以填充所有可用的空条目并添加所有新的传入规则。

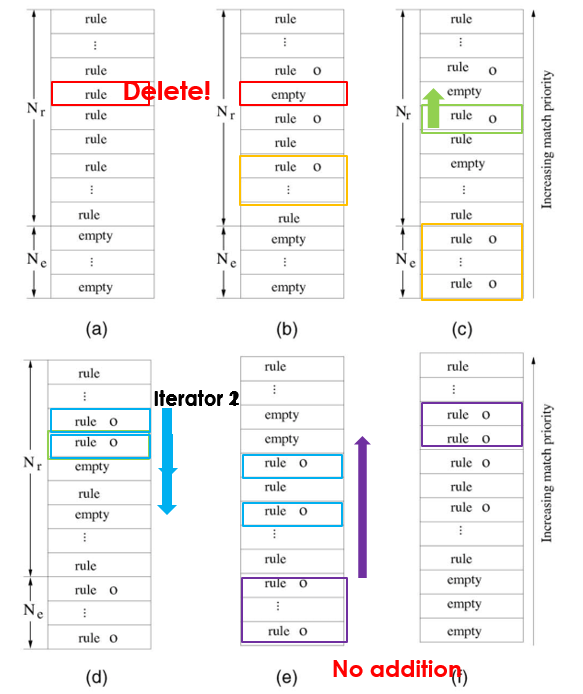


图7-3

【算法举例说明】







删除E2，加入G,H使得：



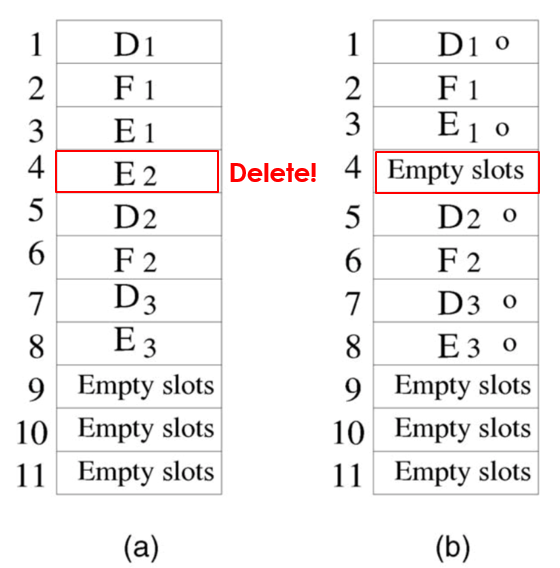
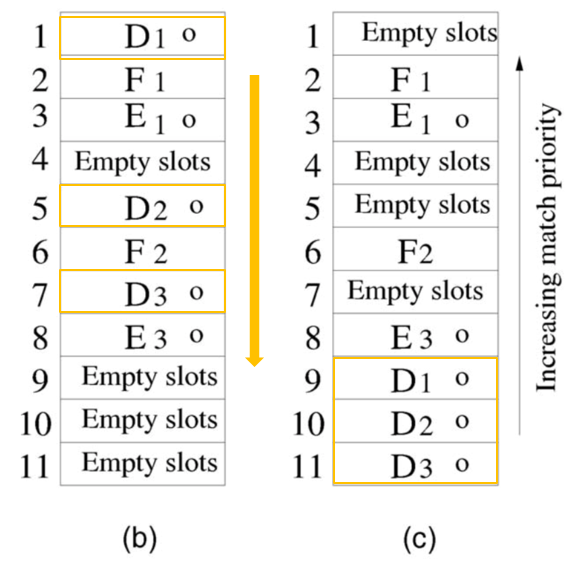
 

图7-4

第一步：删除E2规则。

第二步：empty slot为3，将新的规则联中规则优先级较低的3个规则移动到empty。为d1，d2，d3，并按最终的顺序放置。

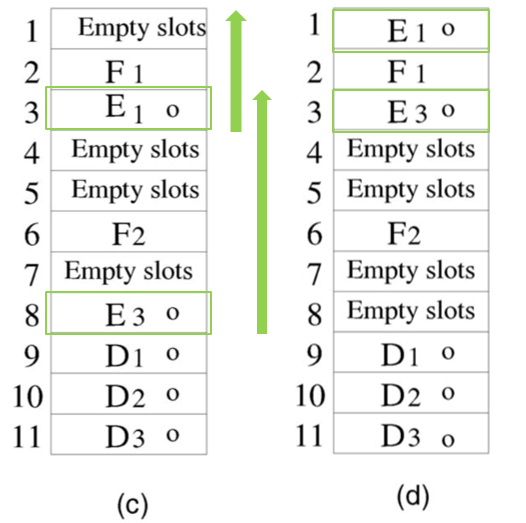
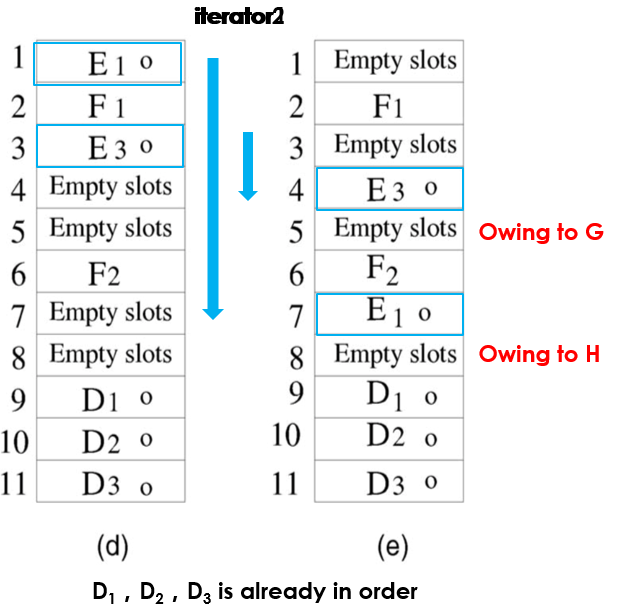
 

图7-5

第三步：将剩余的规则移动到顶部，按最终的顺序放置。E1，e3。

第四步：迭代重新排列top的相关规则，将它们放在所有Ne空条目下面，1位的e1移动到7位，3位的e3移动到4位。并预留出位置给G,H。

第五步：将**相关规则**移至顶部（底部，以移动代价最小为标准）以填充所有可用的空条目并添加所有新的传入规则。

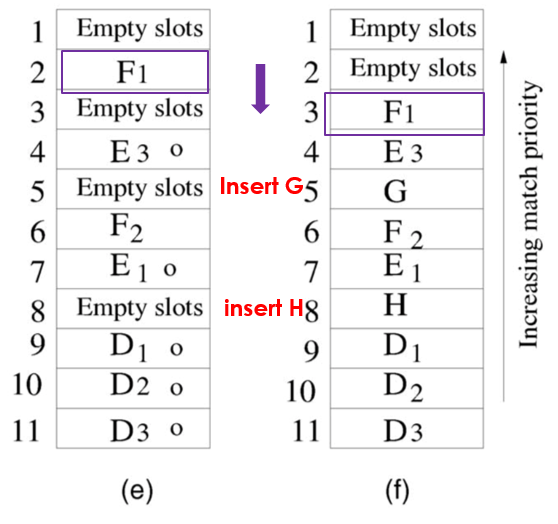


图7-6

### 4. CoPTUA算法效果

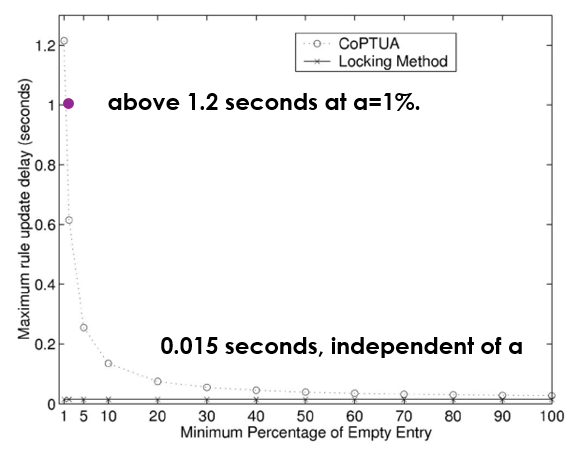


图7-7

使用CoPTUA的每个规则更新的最大延迟是1.2秒（在empty slot占比1%时）。这虽然在性能上来说和上锁差不多，但是在这段时间，上锁的方法将导致有0.1875million个包会丢失。

使用CoPTUA的平均时延为0.015s。

## 八、RuleTris: Minimizing Rule Update Latency for TCAM-based SDN Switches（最小化依赖）

软件定义网络（SDN）被认为能够对数据中心网络进行更加动态的管理，以便随着网络策略的变化迅速响应网络事件。尽管越来越多的SDN控制器体系结构针对快速策略更新进行了优化，但数据平面，尤其是物理SDN交换机上主流的基于TCAM的流表，对于快速规则更新仍未得到优化。这逐渐成为策略更新管道的主要瓶颈。

在本文中，我们介绍RuleTris，这是第一个**SDN更新优化框架**，它可最大限度地减少基于TCAM的交换机的规则更新延迟。 RuleTris使用**依赖图（DAG）**作为关键抽象来最小化更新延迟。RuleTris使用新的依赖保留算法（novel dependency preserving algorithms）有效地获得DAG，该算法随着编译过程递增地构建规则依赖性。然后，在DAG的指导下，RuleTris优化TCAM中的规则更新，以避免不必要的规则移动。

我们证明RuleTris使用最少数量的TCAM条目移动生成TCAM更新。在评估中，RuleTris在我们的硬件原型上实现了<12ms和90％的中位数<15ms的端到端规则更新延迟，优于最先进的合成编译器CoVisor 20倍。

### 问题的提出

现在对TCAM的优化大多都停留在最小化移动规则条目上，尽管有各种各样的算法被提出，由于在物理上的更新机制并未改变，它们在性能上都存在着瓶颈。换一个角度思考，我们观察到，为了维护依赖关系，一个规则的移动可能引入很多不必要的规则移动。如果我们能**最小化这个依赖关系**，那么可以大大的提升性能，以突破这个性能瓶颈。我们的主要挑战为：

* 如何高效的生成DAG图；
* 如何根据DAG图高效的进行优化跟新。

### RuleTris 框架



图8-1

RuleTris用于最小化规则更新延迟的方法是**通过维持最小的DAG来消除不必要的TCAM条目移动**。如上图所示，该框架分为三个部分。

* Front-end in controller：

前端主要由一个增量的SDN协议编译器组成，用来组合多张流表生成DAG图。

RuleTris允许administrators通过合成运算符组合多个控制器应用程序或控制器。这种能力由构成RuleTris前端的通用组合编译器提供。RuleTris组合编译器与应用程序或控制器链接，接受它们对网络策略的主动（proactive）或被动（reactive）修改。

除了编译流表外，RuleTris还生成DAG以解决匹配的歧义，这取代了其他组合编译器中使用的整数优先级值。在proactive网络策略安装到来后，RuleTris批量编译策略，并为后端提供包含整个DAG的新流表。 在reactive响应策略更新到来后，RuleTris以**增量方式**编译策略更新，并为后端提供增量规则插入，删除和修改以及对DAG的更新。

* Back-end in switch firmware

后端主要由一些列硬件优化器组成，将DAG图映射到一系列TCAM的条目上。

RuleTris后端优化器利用DAG的优势并优化物理交换机中的实际规则安装/更新过程。目前，RuleTris提供了三个后端优化器。

1. **更新调度程序**（update scheduler）使用DAG执行特定于硬件的优化，并生成可证明的最小的更新调度以在TCAM表中实现规则更新。
2. **冗余消除器**(redundancy eliminator)删除了两种类型的冗余规则。
3. **CacheFlow管理器**（cacheflow manager）管理多级规则缓存并执行由DAG引导的规则驱逐。

RuleTris后端直接生成TCAM条目移动序列。

* F/B Interface

DAG图在前端生成后，传递给后端用来更新优化。

### 3. 生成DAG图的三种策略

* **平行组合（Parallel Composition）**



图8-2

平行组合的思路为，首先将两个流表进行交叉相乘，有两种情况需要特殊处理，

1. 删掉交集为空的规则。如上图所示：D和N在表空间上不想交，则删除该结点。
2. 第二种情况是两个结果顶点相邻但相应的规则具有相同的匹配（冗余）。我们设计了一个**两级嵌套图**来消除这种冗余。如下图所示：CM,AN会被CN覆盖。但是为了以后规则更新的正确性，不能将CM,AN删除，而是将这三个结点放在一个两级嵌套图种，用CN作为**索引结点(key vertices)**。

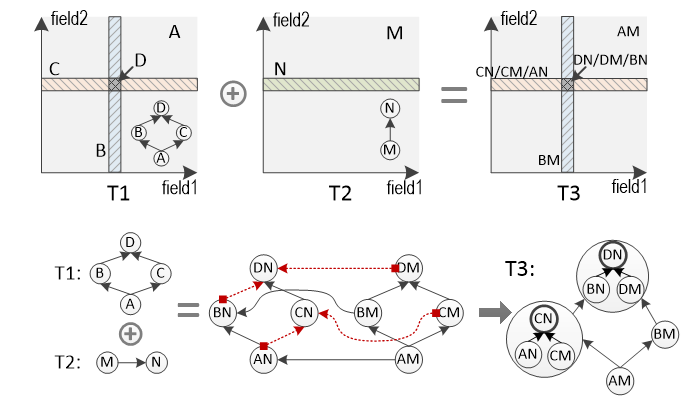


图8-3

* **顺序组合（Sequence Composition）**

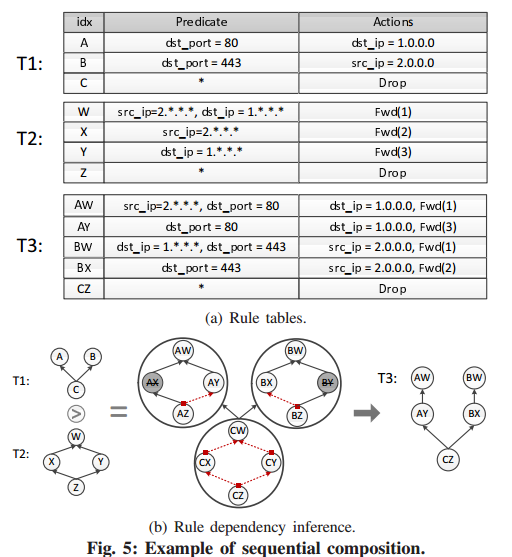


图8-4

顺序组合其实是一个两层循环，外层循环遍历T1表，内层循环将T1表种的i规则与T2表规则组合。如上图所示。同样的，要消除不匹配的空规则如AX。

* **优先级组合（Priority Composition）**

通过优先级堆叠流表来导出T1和T2的优先级组成。因此，可以通过用从T2到T1的mega依赖关系缝合操作数DAG来计算DAG的优先级组成。如下图所示。X->B的边将两个流表缝合起来。

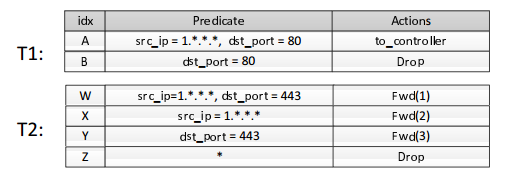


图8-5

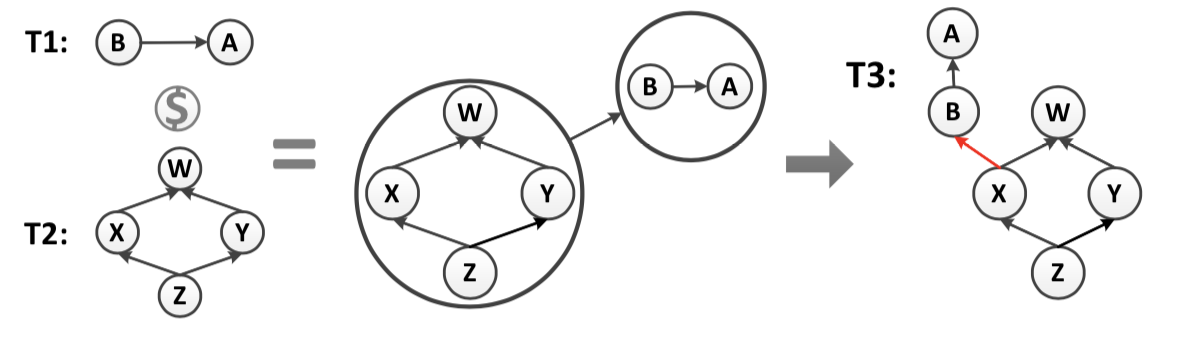


图8-6

找mega依赖的过程如下：

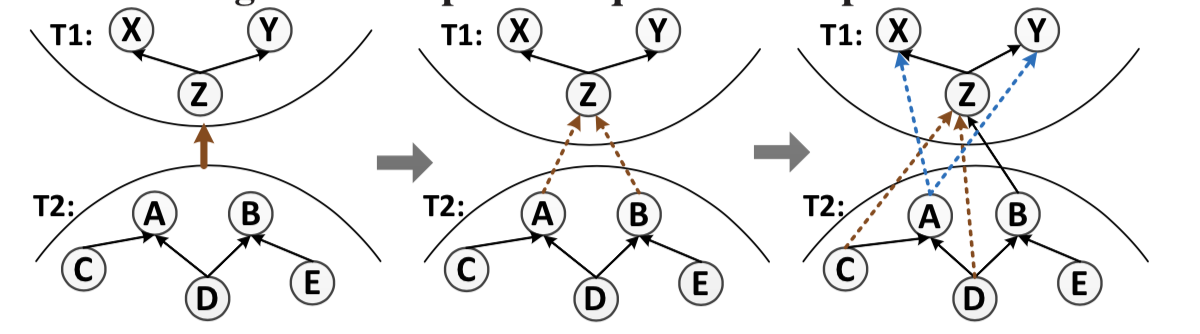


图8-7

1. 找T2中的每个sink结点到T1种的每个source结点，作为试验边。如A->Z,B->Z;
2. 对于每个试验边r2->r1，检查两个规则r1和r2的匹配是否重叠。如果是，则将边r2->r1放入结果DAG中。 否则，递归地生成试验边：

* 看每个r2结点的前任r3，如A的前任C,D，如果，r3比r2更通配，且与r1重叠，将r3->r1加入DAG中并设为试验边。（C->Z, D->Z）；
* 看r1的后任结点r4，如X,Y。如果r1不必r4严格通配。（）。则把r2->r4加入试验边。

1. 递归验证每条试验边。

### 4. 后端优化器

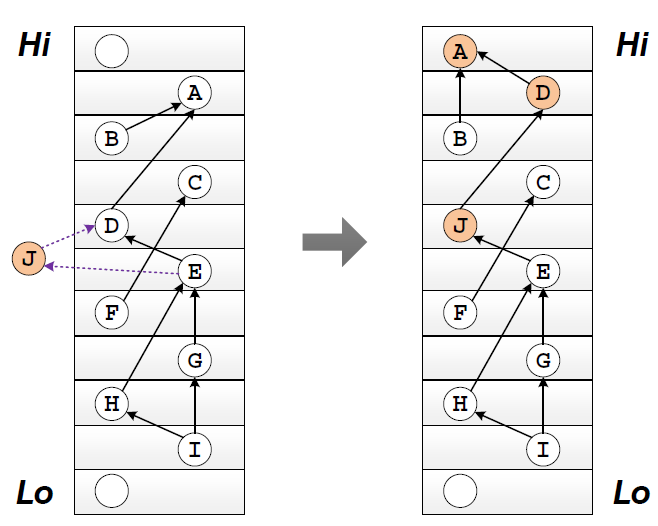


图8-8

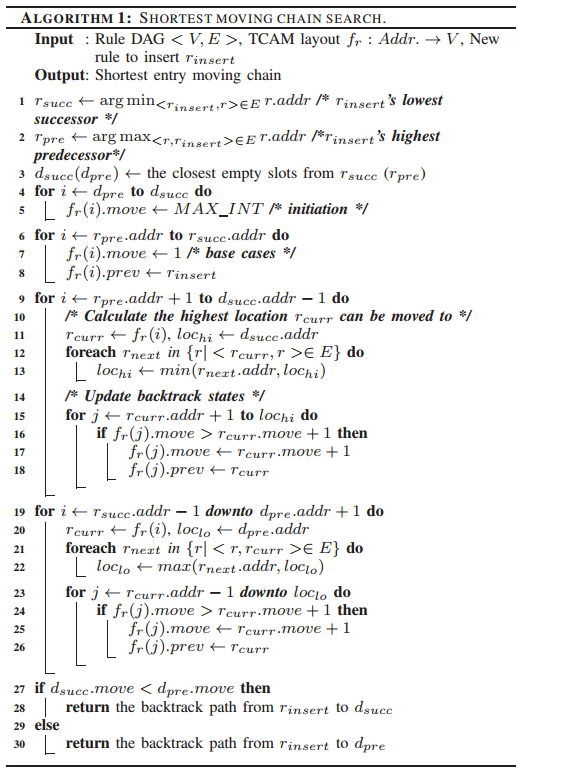
RuleTris可以提供保证，以最少的TCAM移动进行规则更新，编译没有冗余规则的最小大小的流表，并为效率规则缓存层次结构提供支持。步骤如下：

第1步：查找插入规则的TCAM位置范围（例如：d＆e）

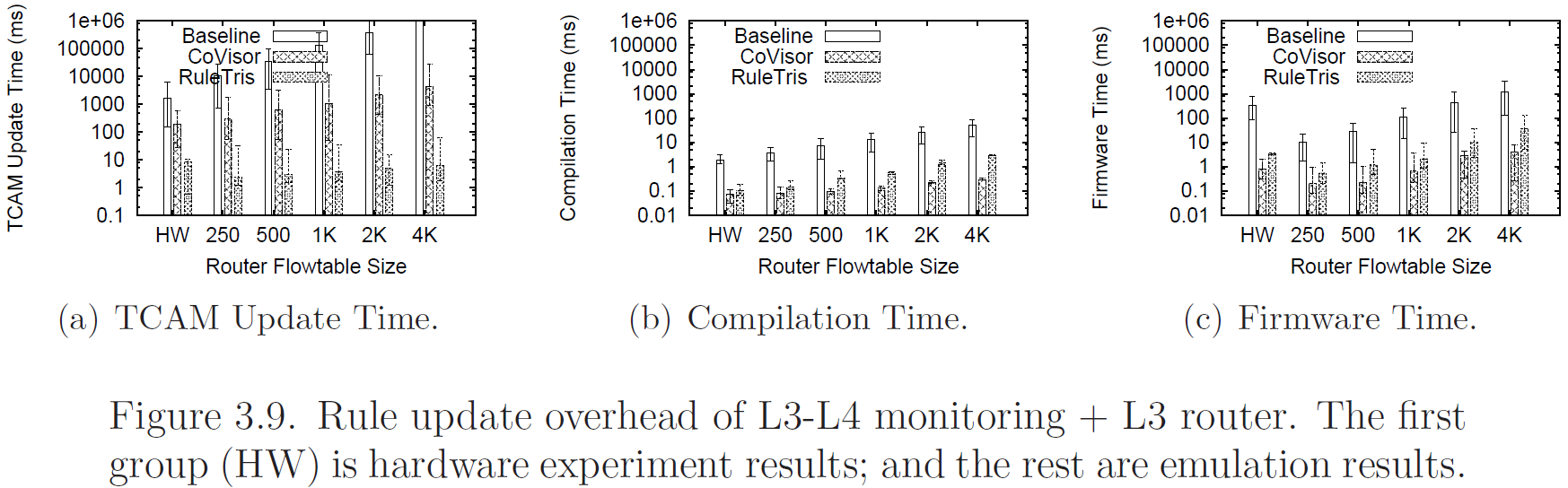
第2步：找到最近的上部插槽和下部插槽

第3步：使用动态规划算法向上和向下搜索最短的移动链。见下图算法1。

第4步：使用较短的路径进行实际更新。



### 5. 算法效果



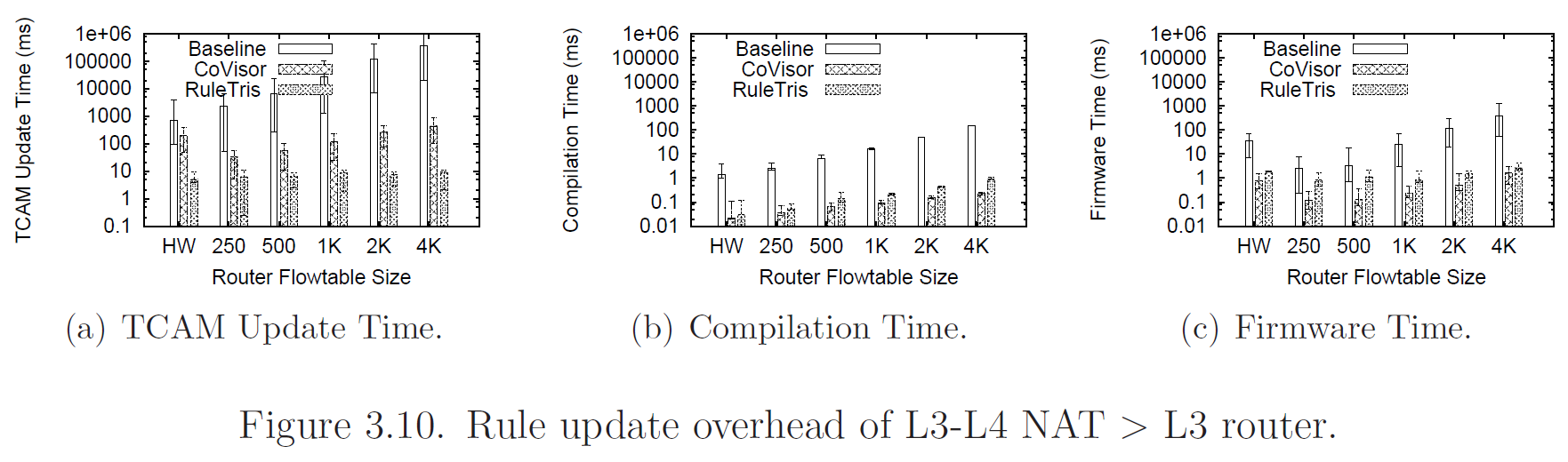


图8-9

对于每规则更新延迟，RuleTris的中位数<12毫秒且90%的结果小于<15毫秒。甚至超过在同一硬件交换机上部署的CoVisor近20倍。