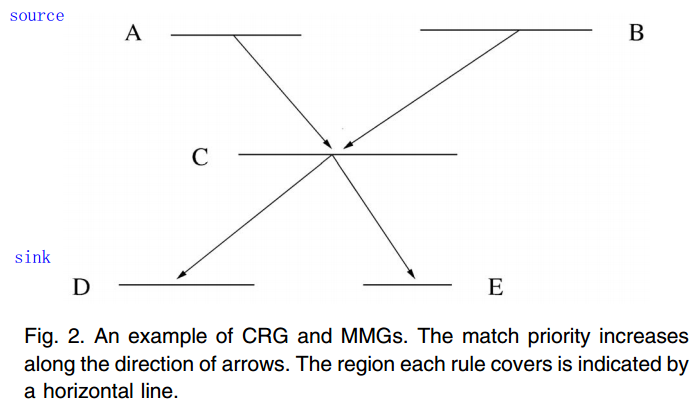
## 一．CoPTUA:Consistent Policy Table Update Algorithm for TCAM without Locking

### 1.摘要

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。

### 2.一些有用的概念和数学表达式



* 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）:

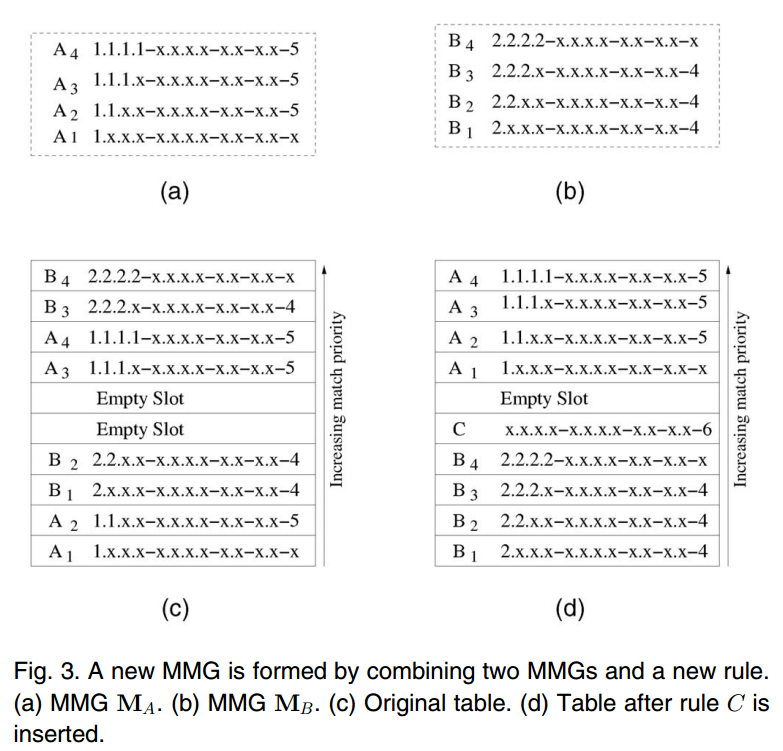
Eg : source A 、B Eg : sink D 、E

* 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

### 3.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。



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。**

### 4. 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。

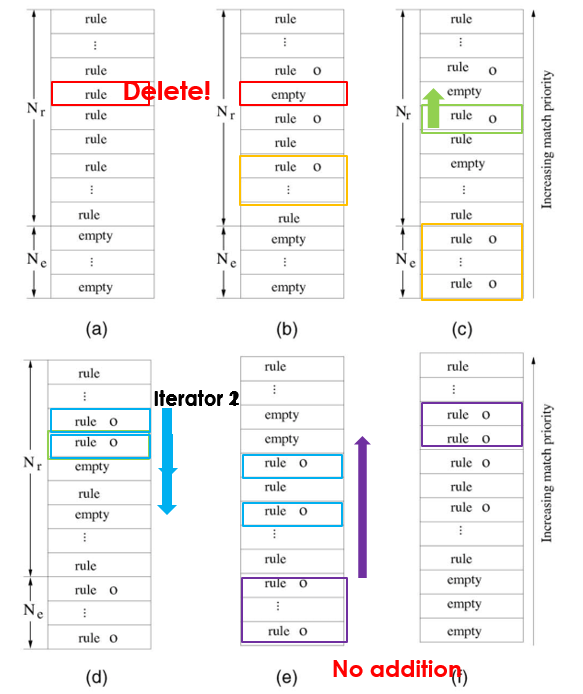
（b）删除所有未出现在最终配置中的规则，所有相关规则都标记为“o”。

（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。



【算法举例说明】

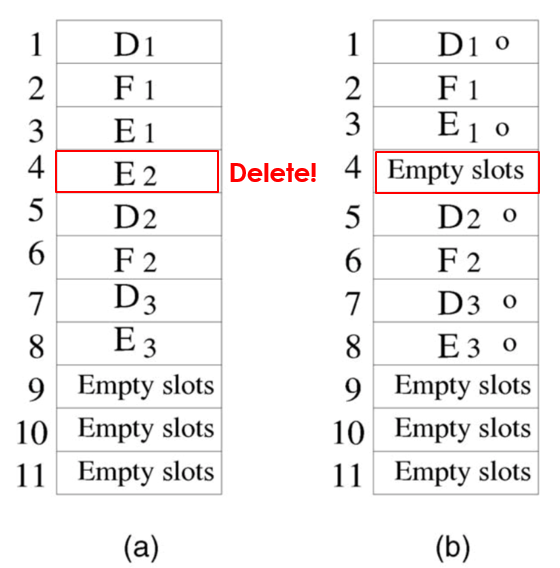
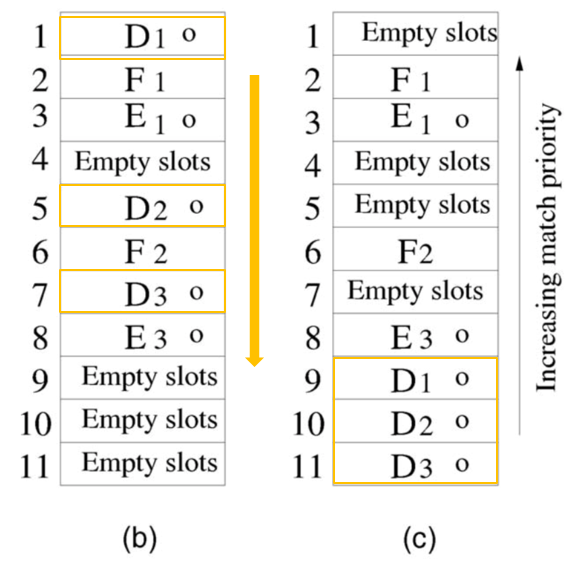






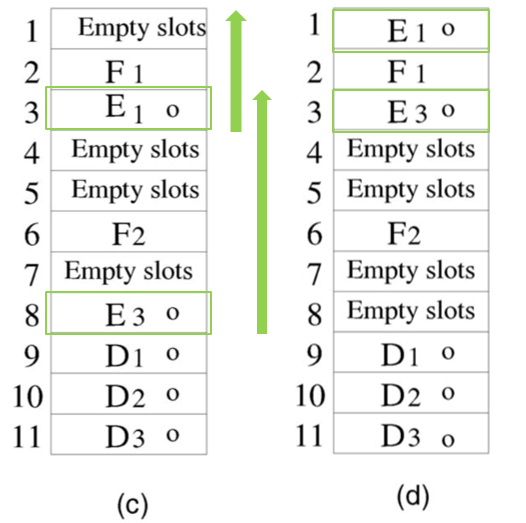
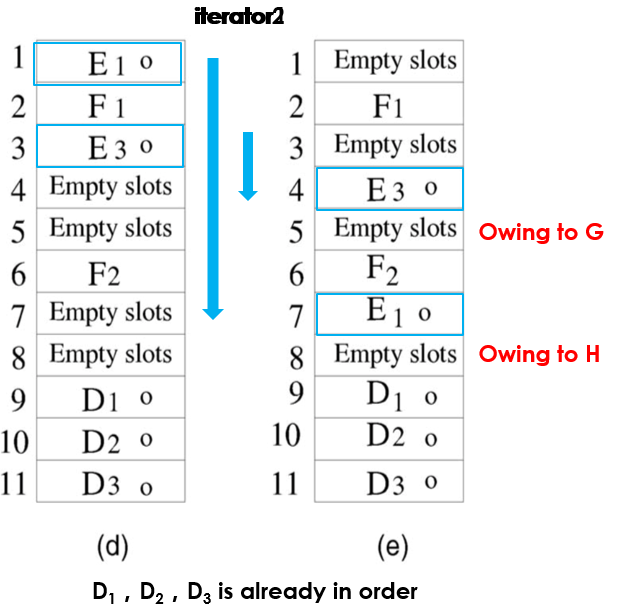
删除E2，加入G,H使得：



第一步：删除E2规则。

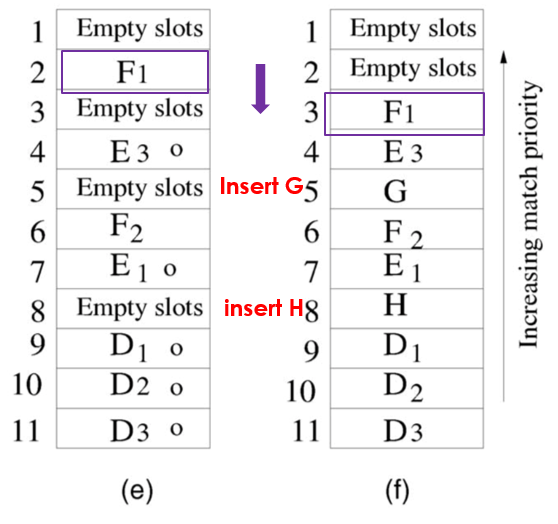
第二步：empty slot为3，将新的规则联中规则优先级较低的3个规则移动到empty。为d1，d2，d3，并按最终的顺序放置。

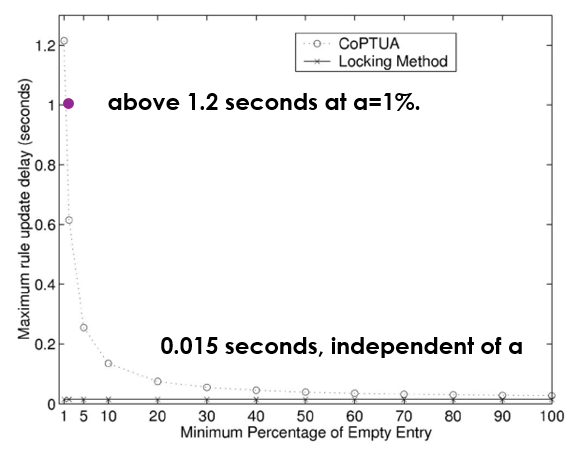
第三步：将剩余的规则移动到顶部，按最终的顺序放置。E1，e3。

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。



### 4. CoPTUA算法效果



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。

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

### 1.摘要

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。

### 2.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。

### 3.RuleTris 框架



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。

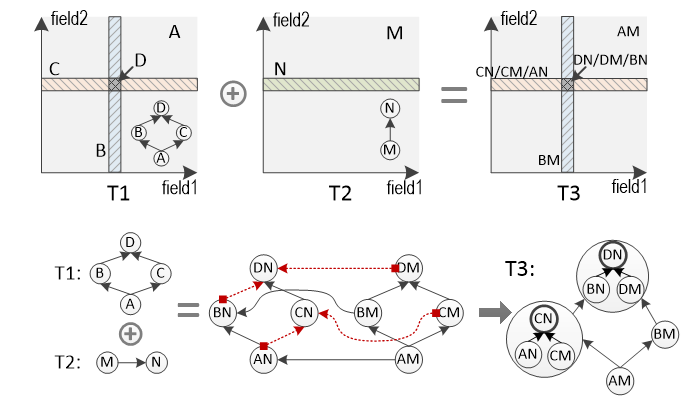
### 4.生成DAG图的三种策略

* **平行组合（Parallel Composition）**

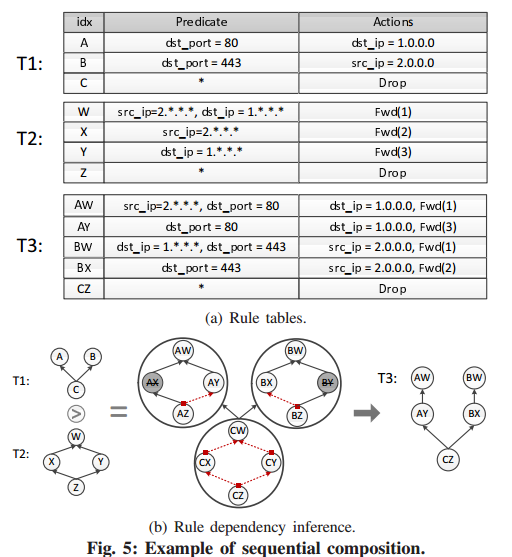


平行组合的思路为，首先将两个流表进行交叉相乘，有两种情况需要特殊处理，

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)**。



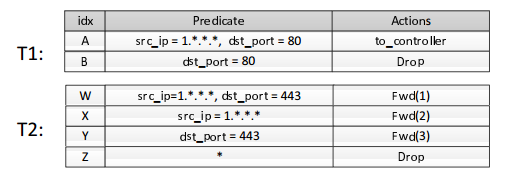
* **顺序组合（Sequence Composition）**

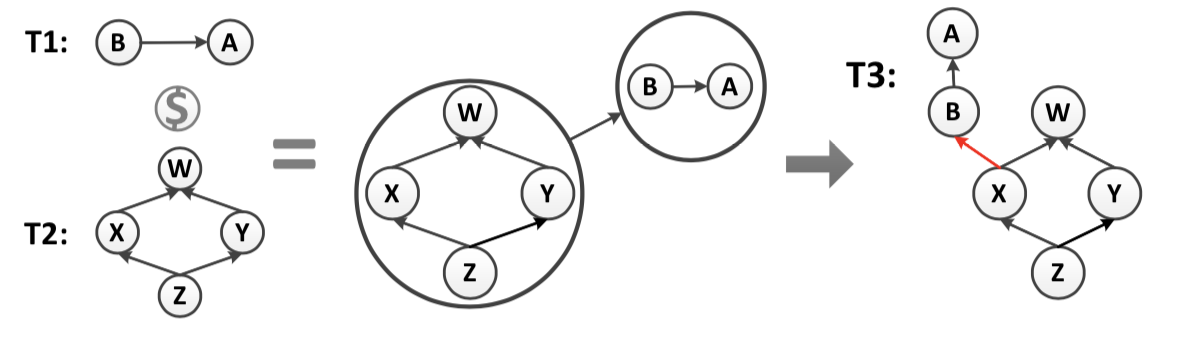


顺序组合其实是一个两层循环，外层循环遍历T1表，内层循环将T1表种的i规则与T2表规则组合。如上图所示。同样的，要消除不匹配的空规则如AX。

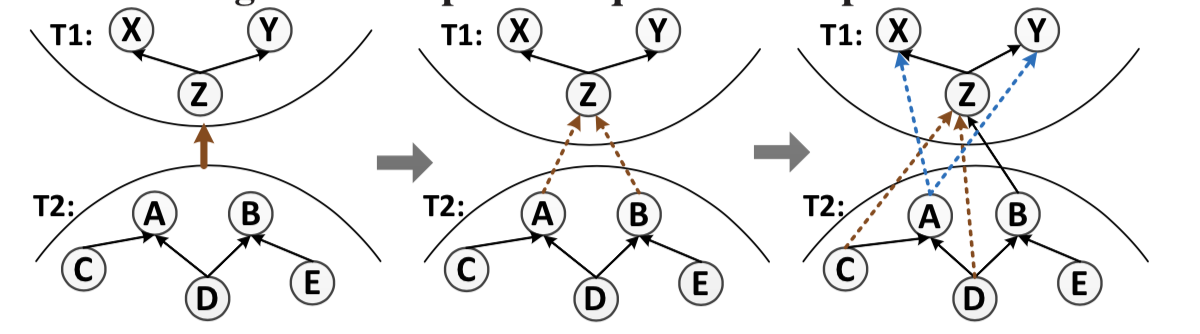
* **优先级组合（Priority Composition）**

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。





找mega依赖的过程如下：

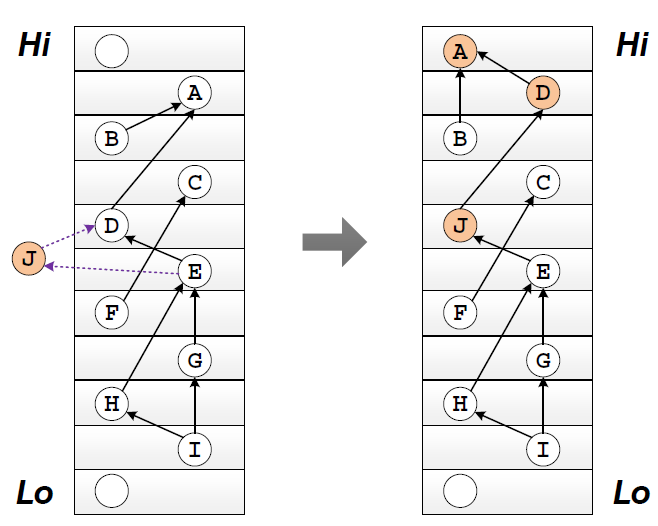


1. 找T2中的每个sink结点到T1种的每个source结点，作为试验边。如A->Z,B->Z;
2. 对于每个试验边r2->r1，检查两个规则r1和r2的匹配是否重叠。如果是，则将边r2->r1放入结果DAG中。 否则，递归地生成试验边：

* 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。

### 5.后端优化器



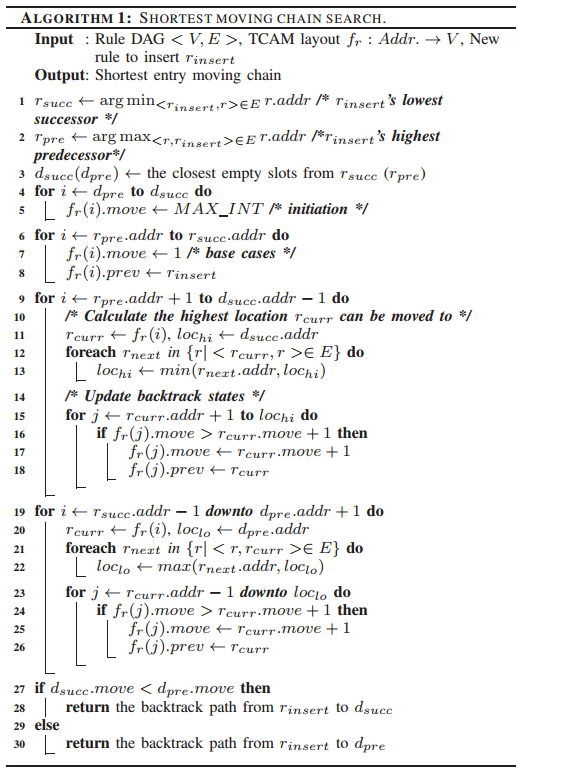
RuleTris可以提供保证，以最少的TCAM移动进行规则更新，编译没有冗余规则的最小大小的流表，并为效率规则缓存层次结构提供支持。步骤如下：

第1步：查找插入规则的TCAM位置范围（例如：d＆e）

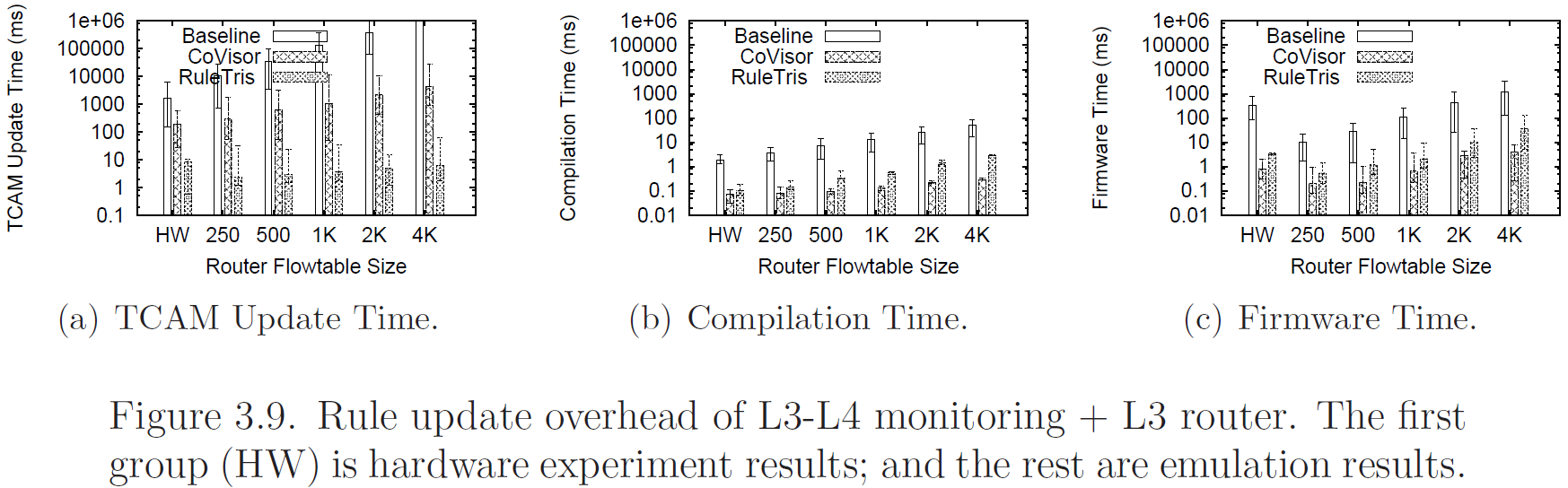
第2步：找到最近的上部插槽和下部插槽

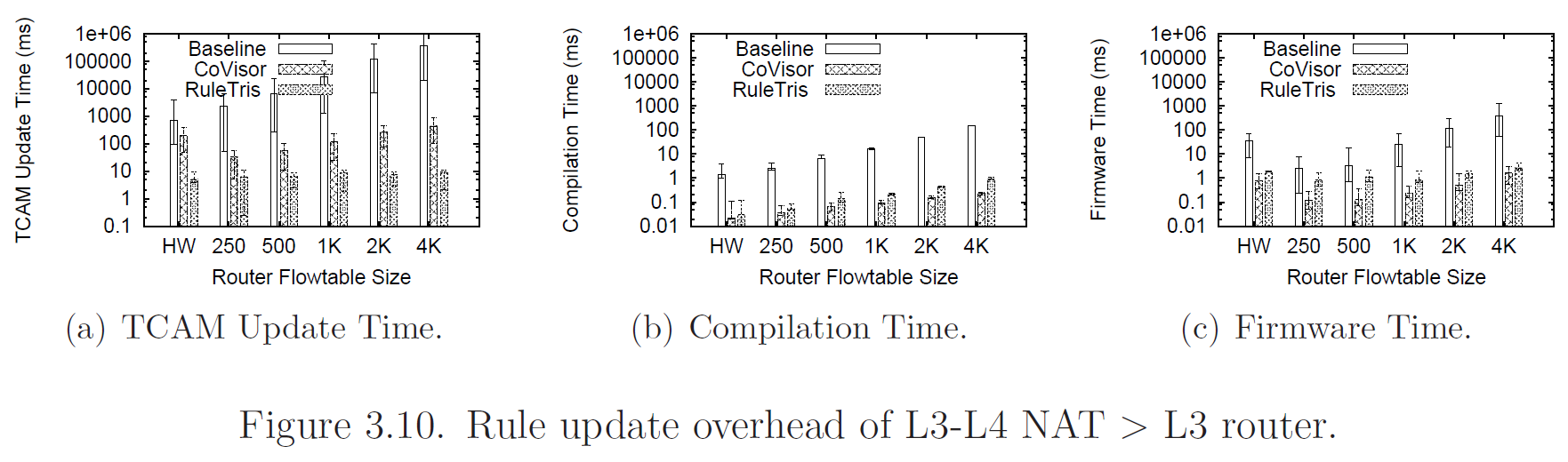
第3步：使用动态规划算法向上和向下搜索最短的移动链。见下图算法1。

第4步：使用较短的路径进行实际更新。



### 6.算法效果





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。