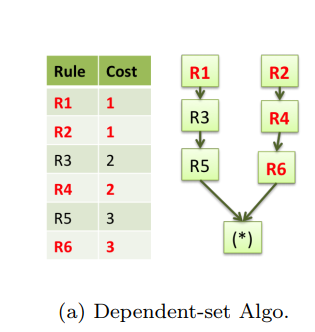
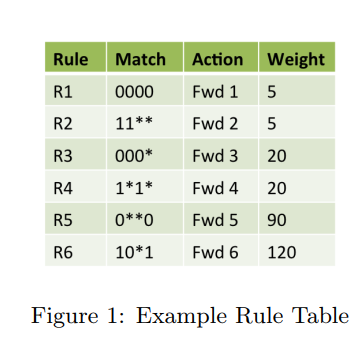
SDN 流表查询中的cache加速技术调查报告

小组成员及分工

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 |  |
| 张豪 | 2017213815 |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. SDN流表查询技术背景介绍

本文的背景是在硬件交换机上使用TCAM作为cache处理尽可能多的包，如果cache没有命中就把包发送给软件交换机，这是因为硬件查询更快，而软件能存储更多的流表规则。然而在cache中存储规则时经常会遇到依赖的问题。



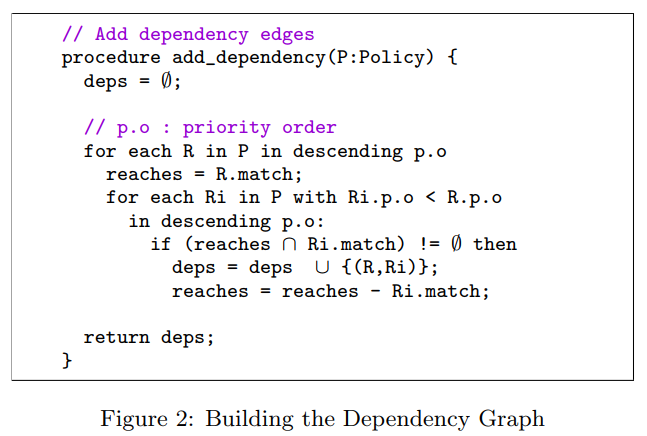
一般情况下，我们总希望把最经常使用的规则存放在cache中，上图的weight就代表每条规则对应的流量。如果只将R5，R6存放在cache中，那0000就会在cache中命中R5，可实际上0000应该对应的是R1，因为R1的优先级更高。为了避免这种错误，现有的方案都是把整个依赖关系全部存入缓存，根据优先级来判断命中哪一个。可是许多服务可能会生成很长的依赖链，其中大部分都是很少出现的，但由于优先级更高只能一并被存入cache。例如防火墙服务，可能在设置允许通过的规则之上还定义了很多禁止的IP，这些禁止规则很少会遇到，而为了保持cache的正确性不得不放入cache中，占用了大量空间，影响其他规则放入。在cache内和cache外交换整个依赖关系链是低效的。

1. cache替换策略

对应上述问题，文章提出的解决方案主要是：

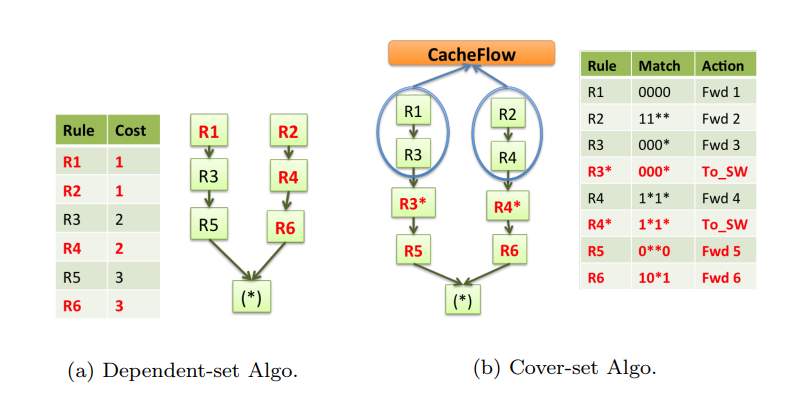
1. 首先生成规则依赖关系的关系图
2. 把较长的规则链切割为小的规则集，创建一些新的规则来覆盖不常用但优先级更高的规则，如果命中了这个新的规则重新发给上层软件交换机来处理，只把常用规则和新创建的规则存储在cache中，减少了需要存储的规则的数量
3. 处理规则链的变化

**1. 生成依赖关系的关系图**



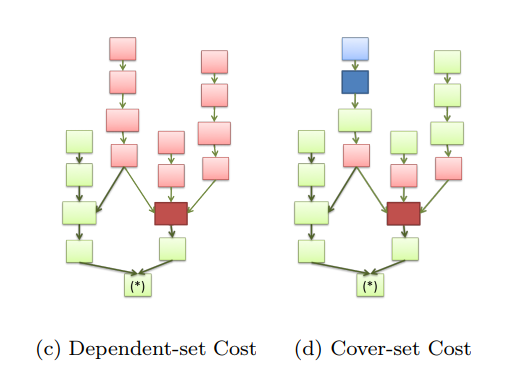
需要注意的地方是它没有找所有的规则之间的关系，而是根据优先级生成规则图，在找到依赖保存关系后就移除当前的reach，防止更低优先级的规则也建立依赖。

**2. 重新生成依赖规则集**



在生成规则依赖图(a)之后，每一条规则对应一个cost，cost就是指这条规则本身和依赖的其他规则数量，计算最优解是NP问题，所以构造启发式算法，可以使用贪心算法选出weight（流量）/cost（数量）最大的规则集放入cache中。不过为了解决长依赖链的问题，本文把weight较小优先级又较高的规则合并为一个规则R\*，如图(b)所示，R\*定义的操作就是转发给上层软件交换机To\_SW，相当于是miss，这样减少了依赖链的长度，重新计算weight和cost，使用贪心算法选取合并后的规则集放入cache中。

**3. 处理规则链的变化**



规则集之间是独立的，移除旧的规则集不会对新的规则集造成影响。如图（d），移除红色的规则集并不会对新加的蓝色规则集产生影响，作者团队设计了一个算法来维持依赖图的更新，有更新发生时不需要重新构建整个依赖图，但在文章里没有说明具体的算法。

1. 对生命周期短、负载量少的流的处理方法

这种方法缩减了规则集，在遇到小流时不会将整个规则链放入cache中，而是选取缩减规则链选取weight/cost最大的规则集放入cache中，避免了cache的大规模更新，在遇到大量小流时也能避免抖动，提高了cache命中率。

四、cache的更新问题