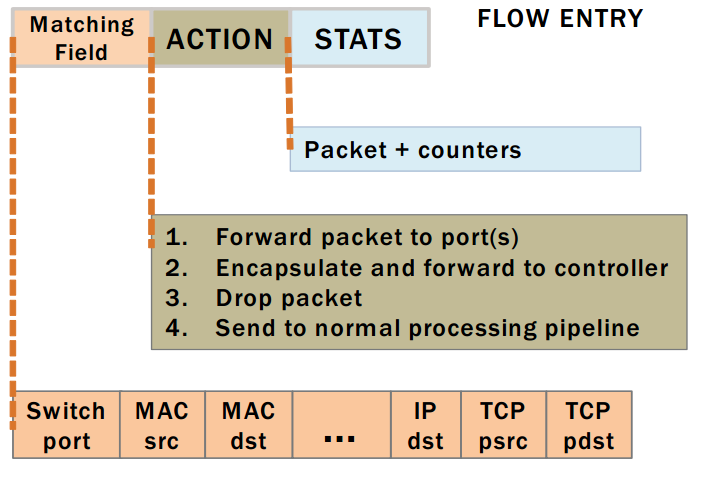
# 论文阅读总结

## 一、Efficient Flow Table Management Scheme in SDN-Based Cloud Computing Networks

这篇文章主要讲的就是microflow table和megaflow table的早期实现思想

1.流表的主要结构如下



首先是匹配域，匹配报文包头中的一个或多个关键字，记录进入交换机的端口，源MAC地址，目的MAC地址，源IP地址，目的IP地址，源TCP端口号，目的TCP端口号。由于OpenFlow协议需要匹配网络中所有类型的报文，所以匹配域才会这么的复杂

然后是动作域，表示对报文的转发动作。包括转发到制定端口，打包并发给控制器，丢弃报文，发送给处理管道

最后是状态域，在这篇文章中，状态域表示某一个表项匹配的次数，但是我认为这个域也可以有其他的意义，例如tag

2.流表存储结构：

流表项的存储结构一共有两种，一种叫做TCAM，一种叫做SRAM，TCAM开销太大不予讨论。SRAM有效利用资源，这个存储结构的实现简单来说，是一个基于hash的流表，和一个基于通配符匹配的流表

基于hash 的流表项，为一个报文的全部的需要匹配的包头域取了一个hash值，因此只要某一个域有一点点不一样，就会miss，就会重新存储一个新的hash表项

基于通配符的流表，如果所有以W/24为目的IP的报文都要被转发到1端口，那么W的后8位和其他域就不用匹配了 这种方式减小了空间开销，但是在最差情况下，一个报文可能需要遍历整个流表，在表项逐渐增加时，查询时间可能会线性增加

3.混合流表结构

基于以上对SRAM存储结构中基于hash的流表和基于通配符的流表的分析，提出了混合使用两种流表的存储结构，即同时使用基于hash的流表和基于通配符的表，首先查hash 表，没查到再查通配符表，然后将这一个包信息添加到hash 表中。

这样做的好处是，因为有了基于通配符的缓存表，就尽量减少了返回用户态查询多级流表的次数；因为有了基于hash的缓存表，就将查询基于通配符的缓存表的时间复杂度降为O(1)了。在没有hash流表的时候，可能需要遍历整个通配符流表才能找到匹配

但是混合流表结构还是存在问题：首先，流信息变多之后性能会下降，这里是还是说的通配符表的O(n)查询复杂度问题；其次，需要将通配符表中的信息写入hash表，这个写入过程会耗时。同时，hash表还是会很大，很耗空间。这里说的是hash流表中存储的仍然是一个流的全部信息，就像仅仅使用hash流表一样耗费空间

4.hash索引通配符流表存储方案

类似于上述混合流表结构，内核态存在两级流表缓存，一级是基于hash的流表，一级是基于通配符的流表，报文首先在hash流表中进行匹配，如果匹配到了，就根据索引以O(1)复杂度在通配符流表中取出转发动作；如果没有匹配到，那么就以O(n)复杂度遍历通配符流表找到转发动作，然后将索引写回hash流表

与混合流表结构的区别在于：hash流表中存储的内容

前者：【Src port | Dest port | Src IP | 。。。 | Src MAC | Dest MAC】 + 【Action】

后者：【hash value】 + 【Action】

hash 流表中值存储匹配域的hash值，而不是存储具体的匹配信息

5.一致性问题

hash索引通配符流表存储方案的执行顺粗是，报文首先在hash流表中查询，如果在hash表中查到了，还是要继续匹配通配符表项，不能直接取出通配符表的转发动作。如果在hash表中取出的动作与通配符表中取出的动作不一致的话，那么遍历整个通配符表，找到正确的转发动作，并更新hash表。

这里隐含两件事情，第一，hash流表和通配符流表可能存在不一致，第二，通配符表一定是正确的，而hash表不一定正确

5.与OvS的联系

上述基于hash的流表就是microflow table的实现机制

基于通配符的流表就是megaflow table的实现机制

OvS中的流表存储结构就是hash索引通配符流表存储方案

## 二、The Design and Implementation of Open vSwitch

1.OvS是一个通用网络应用，需要支持所有类型的需求，例如L2 MAC地址的匹配，L3 IP地址（DIP和SIP）的匹配，L4 端口匹配，生存时间限制，包头域修改等；

2.OvS强调的是对一般情况的支持，而不是对最坏情况的支持，也就是说，OvS只需要在平均情况下达到良好性能；

3.OvS布置在网络的边缘，随时会进行交换状态（流表）的更新；

4.OvS里有两个主要构件用来进行包转发，一个是ivs-vswitchd，这个各平台通用，另一个是datapath kernel module，这个平台独立；前者在用户态，后者在内核态，当一个报文到达的时候，首先在内核态中查找匹配项，如果没有找到，就转发到用户态的ivs-vswitchd，然后将这一条表项添加到内核态中

【重要】5.micorflow cache是针对一条流建立的缓存表项，megaflow是针对一系列流的集合建立一个缓存表项，这个集合中的所有流都有相同的转发动作，而且都可以通过一个前缀匹配到；在有了megaflow cache之后，microflow cache存储的实际上不再是转发表项，而是megaflow cache的索引；

6.元组分类搜索：元组指的是报文头部中每一个需要匹配的域，例如SIP，DIP，MAC等等，每一个元组都会建立一个哈希表，新来的报文通过匹配每一个元组，得到转发动作。元组分类搜索支持O(1)时间内的更新操作，支持任意形式的报文的匹配而不需要改变匹配算法本身，使用O(1)内存空间

7.OvS提出了“resubmit”方法，解决了“交叉相乘问题”：如果元组空间搜索要匹配两个域A和B，A中有n1条表项，B中n2条表项，如果要用一个表为所有的流建立缓存，那么缓存中一定会有n1\*n2条表项。如果A和B是无关的，那么一定有一些A与B的叉乘项是永远不会被匹配到的，因此可以采用两个表来分别缓存A和B，然后将A和B的匹配结果结合起来得到转发动作

8.microflow的来历：

OvS创建之初，因为发现内核态处理报文能够得到非常的快，因此将所有的报文处理全部放到内核态进行，也就是全部列表项都放在内核态中。但是因为在内核态开发十分的麻烦，而且每次更新到涉及到内核的部署和更新，于是这一方法被放弃了。

当前（2.0）版本的OvS使用了一种新的方法，用microflow cache实现内核模块，缓存中的表项是报文全部的域的精确匹配项。但是使用这种方法有很多问题，其中一个是流的建立非常的慢，因此采用了攒多个流一起建立缓存，和使用多线程利用多核优势的方法

9.mageflow cache：支持通项匹配；所有表项没有优先级；匹配只在一个表中进行，而不是一系列的表连续匹配。

【重要】10.因为OvS要支持所有类型报文的匹配，因此megaflow cache中必须包含所有需要匹配的包头域，因此，megaflow cache中可能会有无用的匹配项，无用表项多了之后，就会使得比配速度变慢。为了优化，采用了四种方法来尽量减少元组数量：

10.1元组优先排序：先匹配优先级高的字段，匹配到了就可以停止匹配

10.2阶梯查询：megaflow只匹配包头域的子集，而不需要匹配tuple中的所有字段。确定子集的方法如下：

将所有的域划分为四组：元数据，L2，L3，L4，然后划分四个检查等级： 一、检查元数据 二、检查元数据和L2域 三、检查元数据、L2、L3域 四、检查所有域 四个等级依次检查，只有匹配成功的等级里面的域会被添加到megaflow table中，其他域则不用检查

10.3前缀查询：使用字典树匹配IP字段，避免匹配IP地址的全部32位

10.4分类器分割：为每一个域创建一个典型元数据，这个元数据匹配所有的合格流，如果一个流不符合元数据，那么将不会符合下一步分类的任何结果

【重要】11.缓存有效性检查：因为流表更新等原因，缓存可能会需要更新。

优先级：当有高优先级的路由表项加进来的时候，原先存储在缓存里面的所有低优先级表项都要修改

大范围更新：每次在缓存中命中之后，都返回用户态查流表，比较两次匹配的结果，如果不一样，则更新缓存；

小范围更新：为每一个域的每一项都给定一个标签tag，将所有改变了的域的项的标签加入一个tag集合，周期性的遍历缓存和tag集合，如果发现缓存表项中有tag集合中对应的域，那么更新这一个表项。因为OvS控制器的应用会不断变复杂，而且OvS中会增加更多会随着网络状态改变的转发动作，每一个周期都会更新更多的表项，因此这一方法被OvS放弃了。

12.大范围更新的问题：

在更新缓存表项的时候，不能为新来的流建立表项，这些流所包含的报文都会被阻塞。解决办法是采用多线程，使得更新缓存的时候还能建立新的表项。

为了提高大范围更新的速度，采用多线程建立缓存表项。但是流的删除还是单线程，因此可能会出现流建立速度超过流删除速度，导致内核空间被迅速充满。OvS2.1中对流的删除也采用了多线程，使得流的删除的速度域流的建立速度大致相等。

13.流的老化：

缓存的流表项不能一直存储在内核中，一些很少用到的缓存表项应该被删除。目前采用的是生存时间，如果一个表项在一定时间内没被使用，那么就删除这个表项。