


ovs源码阅读--元组空间搜索算法

 **yearsj** 发布于 2018-08-22

关于TTS（元组空间搜索算法）的详细介绍可以参考[OVS+DPDK Datapath 包分类技术](#)这篇文章，本文只对该篇博客进行简单的介绍，其中案例和部分图片来自于[OVS+DPDK Datapath 包分类技术](#)

TTS算法主要组成部分

Rule : 单条的包过滤规则+动作

以下为具体的例子：

```
1 Rule #1: ip_src=192.168.0.0/16 ip_dst=0/0 protocol=0/0 port_src=0/0 port_dst=0/0
2 Rule #2: ip_src=0/0 ip_dst=23.23.233.0/24 protocol=6/8(TCP) port_src=0/0 port_dst=23/16
3 Rule #3: ip_src=0/0 ip_dst=11.11.233.0/24 protocol=17/8(UDP) port_dst=0/0 port_dst=4789/16
4 Rule #4: ip_src=10.10.0.0/16 ip_dst=0/0 protocol=0/0 port_src=0/0 port_dst=0/0
```

可以看到一个rule中有多个字段，每个字段的形式为：**字段值/掩码前缀**

Tuple : 使用相同的匹配字段+每个匹配字段都使用相同的掩码长度

以下为具体的例子：

```
1 Tuple #1: ip_src_mask=16 ip_dst_mask=0 protocol_mask=0 port_src_mask=0 port_dst_mask=0
2 Tuple #2: ip_src_mask=0 ip_dst_mask=24 protocol_mask=8 port_src_mask=0 port_dst_mask=16
```

tuple是将有**相同规则**的rule进行合并，例如上述rule #1和rule #4可以看成是同一个tuple #1，因为其每个字段的掩码都相同，所以tuple有如下特点：

- 1. 使用相同的匹配字段
- 2. 每个匹配字段都使用相同的掩码长度

Key：用于hash

以Tuple #2中的Rule #2为例说明一下，首先用tuple的掩码去与rule中的各个**字段值**，丢弃tuple不关心的位，得到：

```
ip_src=_ ip_dst=23.23.233 protocol=6 port_src=_ port_dst=23
```

然后把这些位拼接起来，就是哈希表的key，转换为二进制如下：

```
key = 0001 0111(23) 0001 0111(23) 1110 1001(233) 0000 0110(6) 0000 0000 0001 0111(23)
```

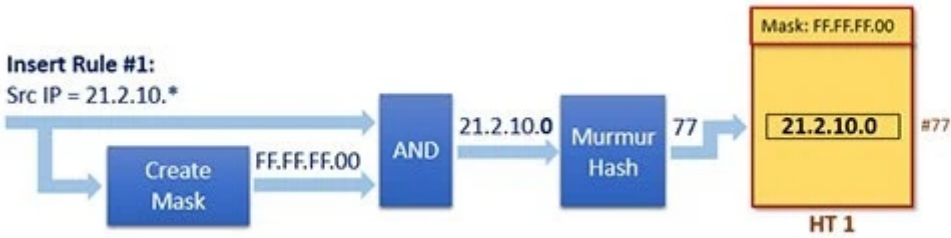
最后，用这个key去做散列，即是哈希表的索引

匹配过程

- 所有的rule都被分成了多个tuple，并存储在相应tuple下的哈希表中
- 当要对一个包进行匹配时，将遍历这多个tuple下的哈希表，一个一个查过去，查出所有匹配成功的结果，然后按一定策略在匹配结果中选出最优的一个。

下面以ovs中具体的事例进行说明：

1. 首先添加一个rule #1，该rule创建的过程中会创建对应的掩码（mask FF.FF.FF.00），也就是TTS中的Tuple，然后rule与mask进行与操作生成key，通过key进行散列得到一个索引值，最终将该rule #1加入到hash表HT 1对应的索引中

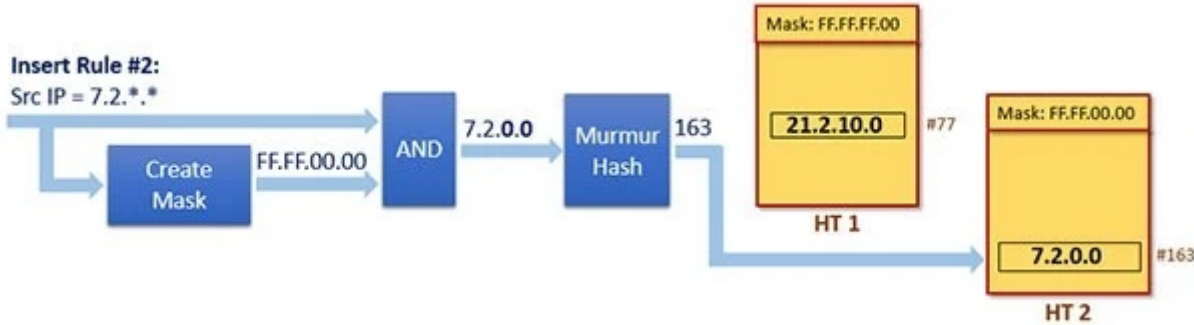


可以看到，同一个哈希表中的mask都是相同的，也就是说每一个tuple对应一个表

1. 接下来收到一个包packet #1，如下图所示，该包查找的过程中，会与所有的hash表进行匹配，由于目前只有一个表HT 1，所以该包会与HT 1对应的mask进行与运算，对其结果进行散列后查到对应表中的结果



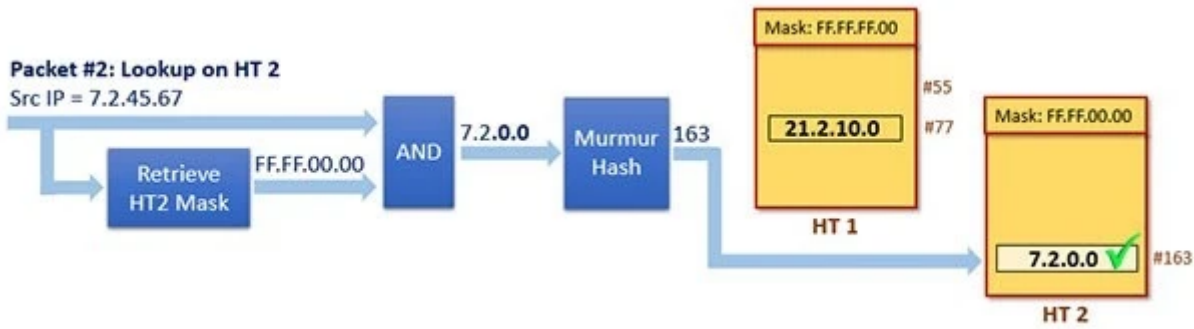
1. 同步骤1，此时又来了一个rule #2，按照同样的步骤，创建一个新的表HT 2



1. 收到另一个包Packet #2，同步骤2进行查找，首先与HT 1对应的mask进行匹配查找，无法找到结果



然后与HT 2对应的mask进行查找，查询到对应的结果



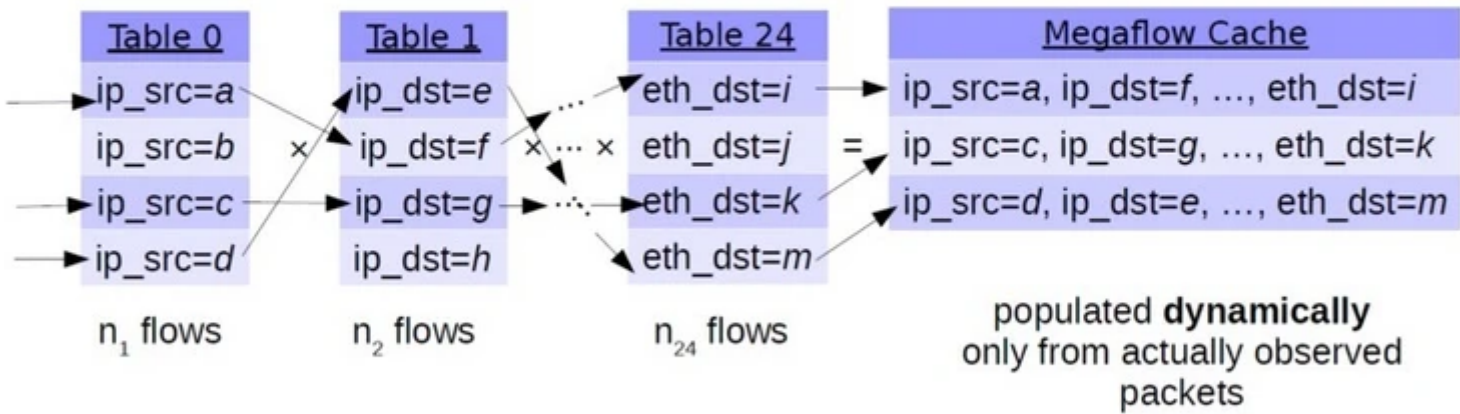
通过上述步骤可以看出来，TTS中的时间复杂度与Tuple的数量相关，如果Tuple的数量越多，则耗费的时间越长，当Tuple的数量==表项的数量，此时等同于挨个遍历所有的表项


OVS与TTS

在[上一篇博文](#)中，其中MegafLOW Cache的实现就是采用了TTS，在如下图中，每个megafLOW cache的表项对应TTS中的rule

Lazy Approach to Populating Cache

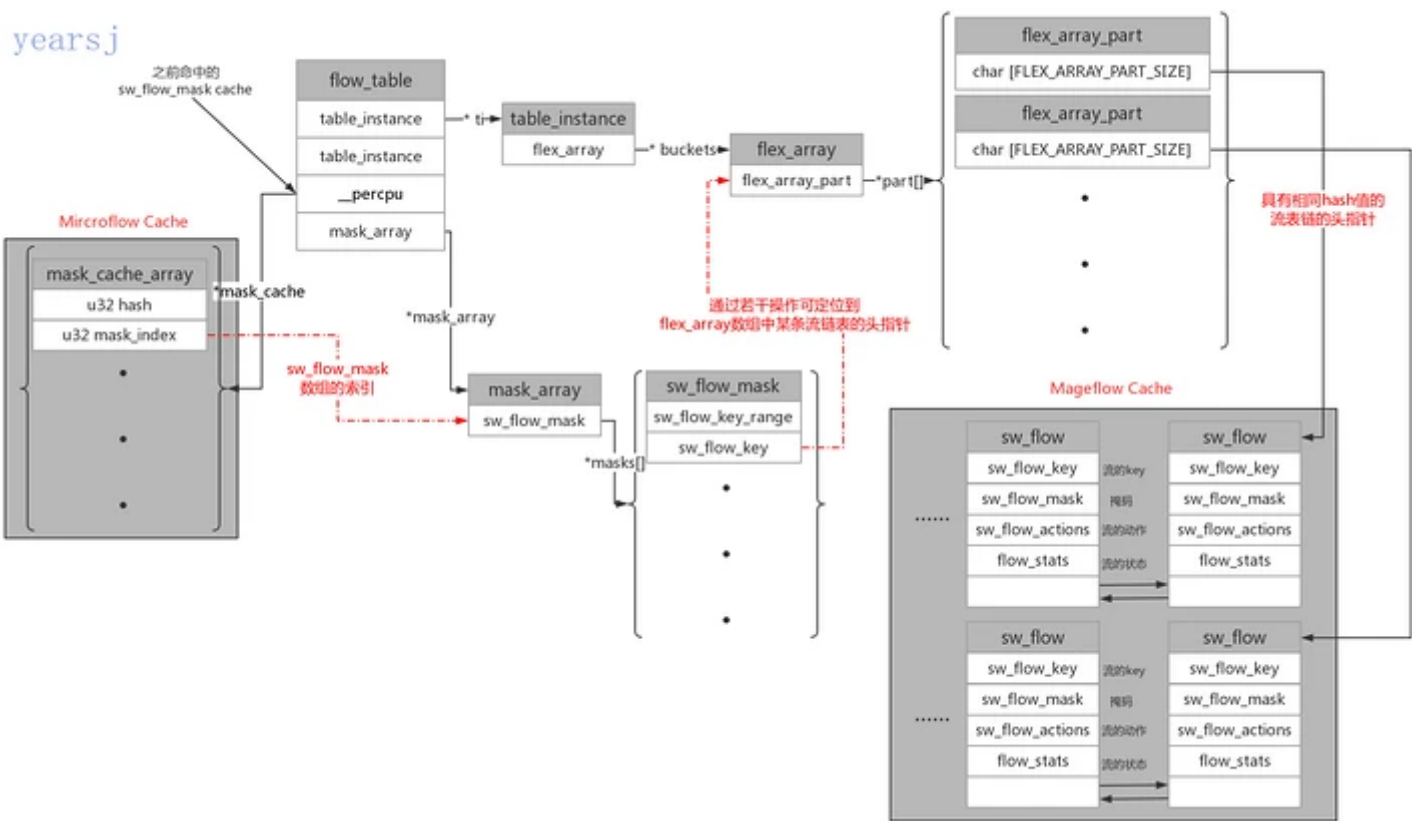
Solution: Build cache of combined “megaflows” **lazily** as packets arrive.





Same (or better!) table lookups as naive approach.
Traffic locality yields practical cache size.

具体的实现结构如下图，在最新的ovs中采用的是Microflow cache和Megaflow Cache结合的方式，其中可以看到Megaflow Cache是通过链表的形式进行组合的，sw_flow_mask结构体相当于是mask（TTS中的tuple），sw_flow结构体相当于是rule，其中Microflow cache中存放的是上次访问的sw_flow_mask索引，具体的流程会在接下来的博客进行详细的介绍。



参考资料

[OVS+DPDK Datapath 包分类技术](#)

作者：[yearsj](#)

转载请注明出处：<https://segmentfault.com/a/11...>

[ovs](#) [sdn](#) [云计算](#) [网络](#) [linux](#)

阅读 1.9k · 更新于 2018-09-14

👍 赞 1

🔖 收藏 1

🔗 分享

本作品系原创，采用《署名-非商业性使用-禁止演绎 4.0 国际》许可协议



yearsj

166 声望 13 粉丝

关注作者

0 条评论

得票数

最新



撰写评论 ...



提交评论

你知道吗？

不要站着调试程序，那会使得你的耐心减半，你需要的是全神贯注。

注册登录

继续阅读

ovs实践--openFlow跨网段组网

sdn (software defines network) 看了些相关的资料，这里记录一下自己对sdn的理解，能力有限，如有错误欢迎指正。 sdn软件定...
[manshu](#) · 阅读 4k · 4 赞

OpenvSwitch 解读

OVS 核心代码 datapath 目录 ovs-switchd OVS数据库管理 ofproto OVS 架构 OVS 主要的数据结构 数据结构关系图 主要的数据结...
[Neal](#) · 阅读 2.6k · 4 赞 · 2 评论

OVS分类器(五)---分类器

分类器用途 classifier在ovs中非常重要，用于流分类，执行相应的动作。属于match+action中的match。在ovs中有三个地方会用...
[ouyangxibao](#) · 阅读 1.2k · 3 赞 · 1 评论

OVS分类器（三） ---Trie

ovs分类器中的前缀树是一个过滤器，将规则的网络层的源目的IP构建前缀树。在查找时，先使用Trie进行匹配，如果不能命中的...
[ouyangxibao](#) · 阅读 753 · 2 赞

从CNI到OVN

诸如calico flannel等CNI实现，通过牺牲一些功能让网络复杂度得以大幅度降低是我极其推崇的，在云原生时代应用不再关心基础...
[sealyun](#) · 阅读 1.6k · 2 赞

OVS DPDK VXLAN隧道处理

vxlan报文进入交换机端口后，根据报文头部信息进行vxlan隧道终结。隧道终结后，根据underlay信息进行overlay映射，得到overl...
[ouyangxibao](#) · 阅读 3.1k · 2 赞

OVS分类器（四） ---subtable

简介 分类器会将规则根据掩码的情况分成子表，每一种掩码情况一个子表。相同掩码的规则放在同一个子表中。规则的匹配是在...

[ouyangxibao](#) · 阅读 860 · 2 赞

OVS分类器（二） ---规则

ovs分类器中，规则从复杂度上可以分为简单规则和关联匹配规则。简单规则独立存在， 关联规则，同时存在。关联规则需要同时...

[ouyangxibao](#) · 阅读 941 · 2 赞

产品

[热门问答](#)

[热门专栏](#)

[热门课程](#)

[最新活动](#)

[技术圈](#)

[酷工作](#)

课程

[Java 开发课程](#)

[PHP 开发课程](#)

[Python 开发课程](#)

[前端开发课程](#)

[移动开发课程](#)

资源

[每周精选](#)

[用户排行榜](#)

[徽章](#)

[帮助中心](#)

[声望与权限](#)

[社区服务中心](#)

[建议反馈](#)

合作

[关于我们](#)

[广告投放](#)

[职位发布](#)

[讲师招募](#)

[联系我们](#)

[合作伙伴](#)

关注

[产品技术日志](#)

[社区运营日志](#)

[市场运营日志](#)

[团队日志](#)

[社区访谈](#)

条款

[服务条款](#)

[隐私政策](#)

[下载 App](#)

Copyright © 2011-2021 SegmentFault. 当前呈现版本 21.06.01

[浙ICP备15005796号-2](#) [浙公网安备33010602002000号](#) ICP 经营许可 [浙B2-20201554](#)

杭州堆栈科技有限公司版权所有

