[**Neutron 理解 (4): Neutron OVS OpenFlow 流表 和 L2 Population [Netruon OVS OpenFlow tables + L2 Population]**](http://www.cnblogs.com/sammyliu/p/4633814.html)

学习 Neutron 系列文章：

（1）[Neutron 所实现的虚拟化网络](http://www.cnblogs.com/sammyliu/p/4622563.html%20)

（2）[Neutron OpenvSwitch + VLAN 虚拟网络](http://www.cnblogs.com/sammyliu/p/4626419.html%20)

（3）[Neutron OpenvSwitch + GRE/VxLAN 虚拟网络](http://www.cnblogs.com/sammyliu/p/4627230.html%20)

（4）[Neutron OVS OpenFlow 流表 和 L2 Population](http://www.cnblogs.com/sammyliu/p/4633814.html%20)

（5）[Neutron DHCP Agent](http://www.cnblogs.com/sammyliu/p/4419195.html)

（6）[Neutron L3 Agent](http://www.cnblogs.com/sammyliu/p/4636091.html%20)

（7）[Neutron LBaas](http://www.cnblogs.com/sammyliu/p/4656176.html%20)

（8）[Neutron Security Group](http://www.cnblogs.com/sammyliu/p/4658746.html%20)

（9）[Neutron FWaas 和 Nova Security Group](http://www.cnblogs.com/sammyliu/p/4675991.html)

（10）[Neutron VPNaas](http://www.cnblogs.com/sammyliu/p/4677386.html%20)

（11）[Neutron DVR](http://www.cnblogs.com/sammyliu/p/4713562.html)

（12）[Neutron VRRP](http://www.cnblogs.com/sammyliu/p/4692081.html)

（13）[High Availability （HA）](http://www.cnblogs.com/sammyliu/p/4741967.html)

      OVS bridge 有两种模式：“normal” 和 “flow”。“normal” 模式的 bridge 同普通的 Linux 桥，而 “flow” 模式的 bridge 是根据其流表（flow tables） 来进行转发的。Neutron 使用两种 OVS bridge：br-int 和 br-tun。其中，br-int 是一个 “normal” 模式的虚拟网桥，而 br-tun 是 “flow” 模式的，它比 br-int 复杂得多。

**1. 基础知识**

**1.1 OpenFlow 结构、流表和数据包处理**

下面左图是 Open vSwitch 中流表的结构。右图这个流程图详细描述了数据包流通过一个 OpenFlow 交换机的过程。

更详细的描述请参见[这里](http://www.sdnap.com/wp-content/uploads/openflow/openflow-spec-v1.3.0-SDNAP_CN.pdf)。

**1.2 ARP Proxy**

Proxy ARP 就是通过一个主机（通常是Router）来作为指定的设备对另一个设备作出 ARP 的请求进行应答。

举个例子：主机A，IP地址是192.168.0.11/24；主机B，IP地址是192.168.1.22/24。主机A和主机B通过路由器R相连接，并且路由器R启用了Proxy ARP，并配置有路由。网络拓扑如下：

     eth0                eth0       eth1                        eth0

    A------------------------Router R----------------------B

192.168.0.11/24   192.168.0.0/24 eth0      192.168.1.22/24

                             192.168.1.0/24 eth1

  在主机A上执行：ping 192.168.1.22，主机 A 不知道主机 B 的 MAC 地址是多少，首先要发送 ARP 查询报文，路由器 R 接收到主机 A 发出的 ARP 查询报文，并代替主机 B 作出应答，应答 ARP 报文中填入的就是路由器 R 的MAC地址。这样，主机A就会认为路由器R的地址是192.168.1.22。以后所有发往192.168.1.22的报文都发到路由器R，路由器R再根据已配置好的路由表将报文转发给主机B。

  这样做的好处就是，主机A上不需要设置任何默认网关或路由策略，不管路由器R的IP地址怎么变化，主机A都能通过路由器B到达主机B，也就是实现了所谓的透明代理。相反，若主机A上设置有默认网关或路由策略时，当主机A向192.168.1.22发送报文，首先要查找路由表，而主机A所在的网段是192.168.0.0/24，主机B所在网段是192.168.1.0/24，主机A只能通过默认网关将报文发送出去，这样代理ARP也就失去了作用。

**优点**：

最主要的一个优点就是能够在不影响其他router的路由表的情况下在网络上添加一个新的router,这样使得子网的变化对主机是透明的。

proxy ARP应该使用在主机没有配置默认网关或没有任何路由策略的网络上

**缺点**：  
　　1.增加了某一网段上 ARP 流量   
　　2.主机需要更大的 ARP table 来处理IP地址到MAC地址的映射   
　　3.安全问题,比如 ARP 欺骗(spoofing)   
　　4.不会为不使用 ARP 来解析地址的网络工作   
　　5.不能够概括和推广网络拓扑

来源：[百度百科](http://baike.baidu.com/view/547486.htm)

**2. 不使用 ARP Responder 和 DVR 时 br-tun 中的流表（flow tables）**

OpenStack 中，Neutron 作为 OVS 的 Controller，向 OVS 发出管理 tunnel port 的指令，以及提供流表。

**2.1 流表分析**

  Neutron 定义了多种流表。以下面的配置（配置了 GRE 和 VXLAN 两种 tunnel types）为例：

1(patch-int): addr:a6:d4:dd:37:00:52

2(vxlan-0a000127): addr:36:ec:de:b4:b9:6b {in\_key=flow, local\_ip="10.0.1.31", out\_key=flow, remote\_ip="10.0.1.39"} 计算节点2

3(vxlan-0a000115): addr:4a:c8:21:3c:3f:f1 {in\_key=flow, local\_ip="10.0.1.31", out\_key=flow, remote\_ip="10.0.1.21"} 网络节点

4(gre-0a000115): addr:4a:8b:0f:9d:59:52 {in\_key=flow, local\_ip="10.0.1.31", out\_key=flow, remote\_ip="10.0.1.21"} 网络节点

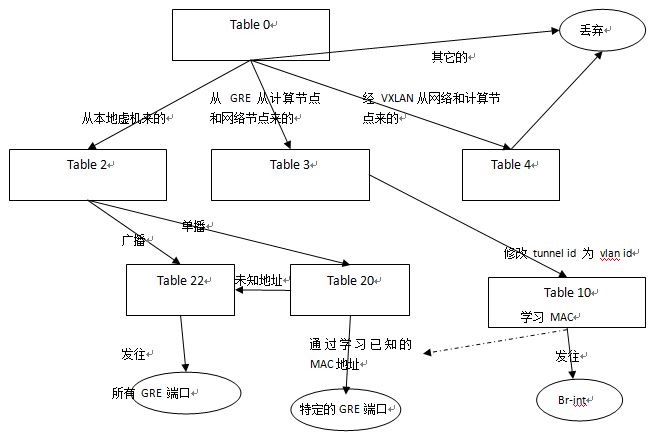
5(gre-0a000127): addr:aa:58:6d:0a:f7:6a {in\_key=flow, local\_ip="10.0.1.31", out\_key=flow, remote\_ip="10.0.1.39"} 计算节点2

其中，10.0.1.31 是计算节点1, 10.0.1.21 是网络节点, 10.0.1.39 是计算节点2。

计算节点1 上 ML2 Agent 启动后的 br-tun 的 flows：

|  |  |  |
| --- | --- | --- |
| 表号 | 用途 | 例子 |
| 0 |  | table=0, priority=1,in\_port=3 actions=resubmit(,4) //从网络节点来的，转 4，结果被丢弃  table=0, priority=1,in\_port=4 actions=resubmit(,3) //从网络节点来的，转 3  table=0, priority=1,in\_port=5 actions=resubmit(,3) //从计算节点来的，转 3  table=0, priority=1,in\_port=2 actions=resubmit(,4) //从计算节点来的，转 4，结果被丢弃  table=0, priority=1,in\_port=1 actions=resubmit(,2) //从虚机来的，转 2  table=0, priority=0 actions=drop //其余的丢弃 |
| DVR\_PROCESS = 1 | handle packets coming from patch\_int unicasts go to table UCAST\_TO\_TUN where remote addresses are learnt | 用于 DVR |
| PATCH\_LV\_TO\_TUN = 2 |  | table=2, priority=0,dl\_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20) //单播包，转 20 table=2, priority=0,dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,22) //组播（包括[广播](http://manpages.ubuntu.com/manpages/oneiric/man8/ovs-ofctl.8.html)）包，转 22 |
| GRE\_TUN\_TO\_LV = 3 |  | table=3, priority=1,tun\_id=0x4 actions=mod\_vlan\_vid:1,resubmit(,10) //将 tun\_id 为 4 的，修改 vlan id 为1，转 10 处理 table=3, priority=0 actions=drop //其余的丢弃 |
| VXLAN\_TUN\_TO\_LV = 4 |  | table=4, priority=0 actions=drop //丢弃 |
| DVR\_NOT\_LEARN = 9 |  | 用于 DVR |
| LEARN\_FROM\_TUN = 10 | 学习table | table=10,priority=1 actions=learn(table=20,hard\_timeout=300,priority=1,NXM\_OF\_VLAN\_TCI[0..11],NXM\_OF\_ETH\_DST[]=NXM\_OF\_ETH\_SRC[],load:0->NXM\_OF\_VLAN\_TCI[],load:NXM\_NX\_TUN\_ID[]->NXM\_NX\_TUN\_ID[],output:NXM\_OF\_IN\_PORT[]),output:1 |
| UCAST\_TO\_TUN = 20 | 外出的单播会被 table 20 处理，table 2 | //学习到的规则  table=20, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:7e:ab:cc actions=strip\_vlan,set\_tunnel:0x3e9,output:5 //如果vlan 为1，而且目的MAC地址等于 fa:16:3e:7e:ab:cc，设置 tunnel id，从端口 5 发出    table=20,priority=0 actions=resubmit(,22) //直接转 22 |
| ARP\_RESPONDER = 21 | ARP table | 当使用 arp\_responder 和 l2population 时候用到 |
| FLOOD\_TO\_TUN  = 22 | Flood table | table=22,dl\_vlan=1 actions=strip\_vlan,set\_tunnel:0x4,output:5,output:4 //对于 dl\_vlan 为1的，设置 tunnel id 为 4，从端口4 和 5 转出 table=22,priority=0 actions=drop |

来个图简单些：



其中比较有意思的是：

（1）为什么从 VXLAN 过来的流量都被丢弃了，最后发出去也用的是 GRE 端口。看来同时有 GRE 和 VXLAN 隧道的话，OVS 只会选择 GRE。具体原因待查。

（2）MAC 地址学习：Table 10 会将学习到的规则（Local VLAN id + Src MAC Addr => IN\_Port）放到 table 20。当表格20 发现一个单播地址是已知的时候，直接从一个特定的 GRE 端口发出；未知的话，视同组播地址从所有 GRE 端口发出。

**2.2 MAC 地址学习**

学习规则：

table=20,hard\_timeout=300,priority=1,NXM\_OF\_VLAN\_TCI[0..11],NXM\_OF\_ETH\_DST[]=NXM\_OF\_ETH\_SRC[],load:0->NXM\_OF\_VLAN\_TCI[],load:NXM\_NX\_TUN\_ID[]->NXM\_NX\_TUN\_ID[],output:NXM\_OF\_IN\_PORT[]

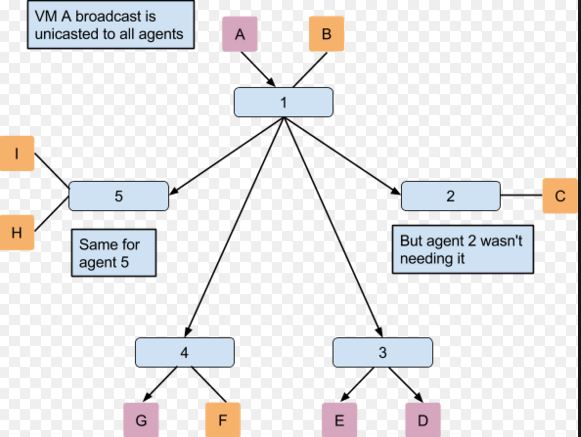
这语法不是很好理解，[这里](http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=tutorial/Tutorial;hb=HEAD)有详细解释。

* table=20：修改 table 20。这是个 MAC 学习流表。
* hard\_timeout：该 flow 的过期时间。
* NXM\_OF\_VLAN\_TCI[0..11] ：记录 vlan tag，所以学习结果中有 dl\_vlan=1
* NXM\_OF\_ETH\_DST[]=NXM\_OF\_ETH\_SRC[] ：将 mac source address 记录，所以结果中有 dl\_dst=fa:16:3e:7e:ab:cc
* load:0->NXM\_OF\_VLAN\_TCI[]：在发送出去的时候，vlan tag设为0，所以结果中有 actions=strip\_vlan
* load:NXM\_NX\_TUN\_ID[]->NXM\_NX\_TUN\_ID[] ：发出去的时候，设置 tunnul id，所以结果中有set\_tunnel:0x3e9
* output:NXM\_OF\_IN\_PORT[]：指定发送给哪个port，由于是从 port2 进来的，因而结果中有output:2。

学到的规则：

table=20, n\_packets=1239, n\_bytes=83620, idle\_age=735, hard\_age=65534, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:7e:ab:cc actions=strip\_vlan,set\_tunnel:0x3e9,output:2

这里可以看到，通过 MAC 地址学习机制，Neutron 可以一定程度地优化网络流向，但是这种机制需要等待从别的节点的流量进来，只能算是一种被动的机制，效率不高。而且，这种机制只对单播帧有效，而对于多播和组播依然无效。其结果是网络成本依然很高。下图中，A 的广播包其实只对 3 和 4 有用，但是 2 和 5 也收到了。



**3. ARP Responder**

    arp\_responder 的原理不复杂。Neutorn DB 中保存了所有的端口的 MAC 和 IP 地址数据。而 ARP 就是一个虚机要根据另一个虚机的 IP 地址查询它的 MAC。因此，只需要 Neutron server 通过 RPC 告诉每个计算节点上的 ML2 agent 所有活动端口的 MAC 和 IP，那么就可以将 br-tun 变成一个供本机适用的 ARP Proxy，这样本机上的虚机的 ARP 请求的响应就可以由 br-tun 在本地解决。Assaf Meller [有篇文章](http://assafmuller.com/2014/05/21/ovs-arp-responder-theory-and-practice/)来阐述 ARP Responder。

  使用 ARP Responder 需要满足两个条件：

（1）设置 arp\_responder = true 来使用 OVS 的ARP 处理能力 。这需要 OVS 2.1 （运行 ovs-vswitchd --version 来查看 OVS 版本） 和 ML2 l2population 驱动的支持。当使用隧道方式的时候，OVS 可以处理一个 ARP 请求而不是使用广播机制。如果 OVS 版本不够的话，Neutorn 是无法设置 arp responder entry 的，你会在 openvswitch agent 日志中看到 “Stderr: '2015-07-11T04:57:32Z|00001|meta\_flow|WARN|destination field arp\_op is not writable\novs-ofctl: -:2: actions are invalid with specified match (OFPBAC\_BAD\_SET\_ARGUMENT)\n'”这样的错误，你也就不会在 ”ovs-ofctl dump-flows br-tun“ 命令的输出中看到相应的 ARP Responder 条目了。

（2）设置 l2\_population = true。同时添加 mechanism\_drivers = openvswitch,l2population。OVS 需要 Neutron 作为 SDN Controller 向其输入 ARP Table flows。

**3.1 升级 OVS**

杀掉 neutron openvswitch， ovs-\* 各种进程

#编译安装

去 http://openvswitch.org/download/ 下载最新版本的代码，解压，进入解压后的目录

安装依赖包，比如 gcc，make

uname -r

./configure --with-linux=/lib/modules/3.13.0-51-generic/build

make && make install

#查看安装的版本

root@compute2:/home/s1# ovs-vsctl --version  
ovs-vsctl (Open vSwitch) 2.3.2  
Compiled Jul 12 2015 09:09:42  
DB Schema 7.6.2

#处理 db

rm /etc/openvswitch/conf.db （老的db要删除掉，否则会报错）

ovsdb-tool create /etc/openvswitch/conf.db vswitchd/vswitch.ovsschema

#启动 ovs

cp /usr/local/bin/ovs-\* /usr/bin

ovsdb-server /etc/openvswitch/conf.db -vconsole:emer -vsyslog:err -vfile:info --remote=punix:/usr/local/var/run/openvswitch/db.sock --private-key=db:Open\_vSwitch,SSL,private\_key --certificate=db:Open\_vSwitch,SSL,certificate --bootstrap-ca-cert=db:Open\_vSwitch,SSL,ca\_cert --no-chdir --log-file=/var/log/openvswitch/ovsdb-server.log --pidfile=/var/run/openvswitch/ovsdb-server.pid --detach --monitor

ovs-vswitchd unix:/usr/local/var/run/openvswitch/db.sock -vconsole:emer -vsyslog:err -vfile:info --mlockall --no-chdir --log-file=/var/log/openvswitch/ovs-vswitchd.log --pidfile=/var/run/openvswitch/ovs-vswitchd.pid --detach --monitor

#启动 neutron openvswitch agent，确保log 文件中 ovs-vsctl 和 ovs-ofctl 调用没有错误

#修改 /usr/share/openvswitch/scripts/ovs-lib 文件，保证机器重启后 OVS 正常运行

将 rundir=${OVS\_RUNDIR-'/var/run/openvswitch'} 改为 rundir=${OVS\_RUNDIR-'/usr/local/var/run/openvswitch'}

**3.2 ARP Responder**

有了 arp\_responder 以后，br-tun 的流表增加了几项和处理：

（1）table 2 中增加一条 flow，是的从本地虚机来的 ARP 广播帧转到table 21

# ARP broadcast-ed request go to the local ARP\_RESPONDER table to be locally resolved  
table=2, n\_packets=0, n\_bytes=0, idle\_age=3, priority=1,arp,dl\_dst=ff:ff:ff:ff:ff:ff actions=resubmit(,21)

（2）在 table 21 中增加一条 flow 将其发发往 table 22

# If none of the ARP entries correspond to the requested IP, the broadcast-ed packet is resubmitted to the flooding table  
table=21, n\_packets=0, n\_bytes=0, idle\_age=4, priority=0 actions=resubmit(,22)  
  
如果下面第 （3）步增加的 flow rule 都处理不了这条 request，那么转到table 22 去 flood 到所有端口。

（3）由 L2 population 发来的 entry 来更新 table 21。

  table 21 是在新的 l2pop 地址进来的时候更新的。比如说，compute C 上增加了新的虚机 VM3，然后计算节点 A 和 B 收到一条 l2pop 消息说 VM3 （IP 是\*\*\*，MAC 是 \*\*\*） 在 Host C 上，在 network "Z“ 中。然后，Compute A 和 B 会在 table 21 中增加相应的 flows。

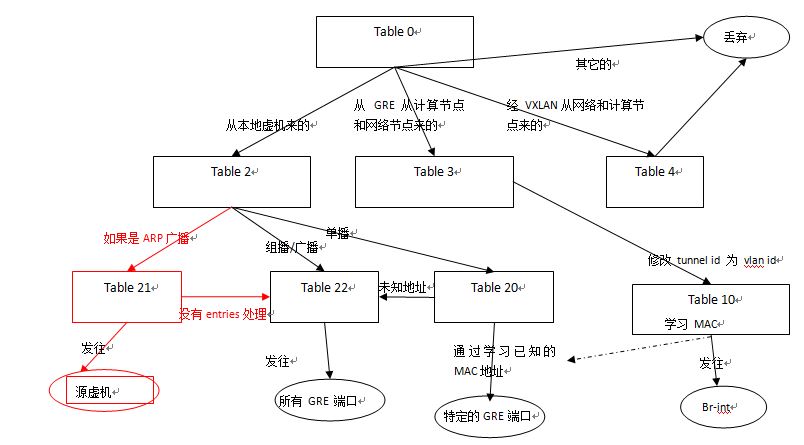
  br.add\_flow(table=21, priority=1, proto='arp', dl\_vlan=local\_vid, nw\_dst= ip, actions=actions)

    其中action为： （好晦涩，这是谁定义的奇葩语法。。幸好 [这里](http://assafmuller.com/2014/05/21/ovs-arp-responder-theory-and-practice/) 有详细解释）

ARP\_RESPONDER\_ACTIONS = ('move:NXM\_OF\_ETH\_SRC[]->NXM\_OF\_ETH\_DST[],'  
 'mod\_dl\_src:%(mac)s,'  
 'load:0x2->NXM\_OF\_ARP\_OP[],'  
 'move:NXM\_NX\_ARP\_SHA[]->NXM\_NX\_ARP\_THA[],'  
 'move:NXM\_OF\_ARP\_SPA[]->NXM\_OF\_ARP\_TPA[],'  
 'load:%(mac)#x->NXM\_NX\_ARP\_SHA[],'  
 'load:%(ip)#x->NXM\_OF\_ARP\_SPA[],'  
 'in\_port')  
  
（4）table 21 的处理过程

     table 21 中的每一条 flow，会和进来的帧的数据做匹配（ARP 协议，network，虚机的 IP）。如果匹配成功，则构造一个 ARP 响应包，其中包括了 IP 和 MAC，从原来的 port 发回到虚机。如果没有吻合的，那么转发到 table 22 做泛洪。

增加的 flow tables 在红色部分：



     因此，通过使用 l2-pop mechanism driver 和 OVS 2.1, Neutorn 可以在本地回答虚机的 ARP 请求，从而避免了昂贵的 ARP 广播。这个功能给 GRE 和  VXLAN 的实现是在 Juno 版本中完成的。 这个 [blueprint](https://bugs.launchpad.net/neutron/+bug/1413056)似乎在支持VLAN 中的这个功能，但是看起来没有完成。

**4. L2 population**

根据[这篇文档](https://github.com/openstack/neutron/tree/master/neutron/plugins/ml2/drivers/l2pop)，l2pop 目前支持 VXLAN with Linux bridge 和 GRE/VXLAN with OVS，其 blueprint 在[这里](https://wiki.openstack.org/wiki/L2population_blueprint)。

**4.1 原理**

    l2pop 的原理也不复杂。Neutron 中保存每一个端口的状态，而端口保存了网络相关的数据。虚机启动过程中，其端口状态会从 down 到build 到 active。因此，在每次端口发生状态变化时，函数 update\_port\_postcommit  都将会被调用：

{'status': 'DOWN/BUILD/ACTIVE', 'binding:host\_id': u'compute1', 'allowed\_address\_pairs': [], 'extra\_dhcp\_opts': [], 'device\_owner': u'compute:nova', 'binding:profile': {}, 'fixed\_ips': [{'subnet\_id': u'4ec65731-35a5-4637-a59b-a9f2932099f1', 'ip\_address': u'81.1.180.15'}], 'id': u'1167e9ac-e10f-4cf5-bd09-6649eab38b32', 'security\_groups': [u'f5377a66-803d-481b-b4c3-a6631e8ab456'], 'device\_id': u'30580ea7-c456-416b-a01e-0fe645edf5dc', 'name': u'', 'admin\_state\_up': True, 'network\_id': u'86c0d29b-4880-4739-bd68-eb3c392f5099', 'tenant\_id': u'74c8ada23a3449f888d9e19b76d13aab', 'binding:vif\_details': {u'port\_filter': True, u'ovs\_hybrid\_plug': True}, 'binding:vnic\_type': u'normal', 'binding:vif\_type': u'ovs', 'mac\_address': u'fa:16:3e:4f:59:9d'}

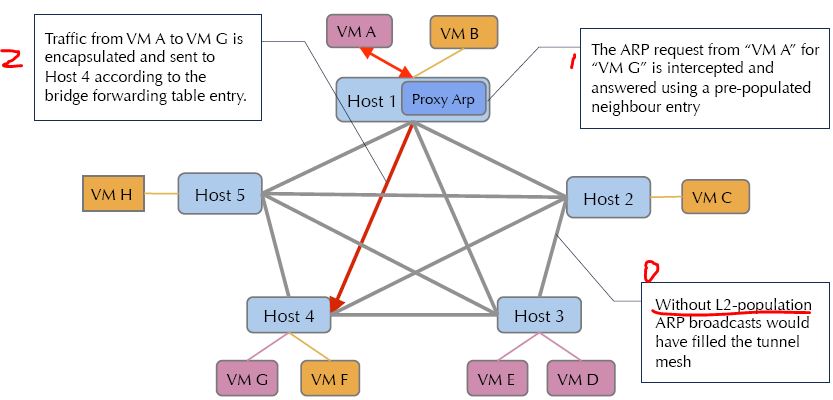
    在某些状态变化下：

* update\_port\_postcommit (down to active) -> \_update\_port\_up -> add\_fdb\_entries -> fdb\_add -> fdb\_add\_tun -> setup\_tunnel\_port  （如果 tunnel port 不存在，则创建 tunnel port）, add\_fdb\_flow -> add FLOOD\_TO\_TUN flow （如果是 Flood port，则将端口添加到 Flood output ports）; setup\_entry\_for\_arp\_reply('add'。如果不是 Flood port，那么 添加 ARP Responder entry （MAC -> IP）) 以及 add UCAST\_TO\_TUN flow Unicast Flow entry （MAC -> Tunnel port number）。
* update\_port\_postcommit (active to down) -> \_update\_port\_down -> remove\_fdb\_entries
* delete\_port\_postcommit (active to down) -> \_update\_port\_down -> remove\_fdb\_entries -> fdb\_remove -> fdb\_remove\_tun -> cleanup\_tunnel\_port, del\_fdb\_flow -> mod/del FLOOD\_TO\_TUN flow; setup\_entry\_for\_arp\_reply ('remove'), delete UCAST\_TO\_TUN flow
* update\_port\_postcommit (fixed ip changed) -> \_fixed\_ips\_changed -> update\_fdb\_entries

    通过这种机制，每个节点上的如下数据得到了实时更新，从而避免了不必要的隧道连接和广播。

* Tunnel port
* FLOOD\_TO\_TUN （table 22）flow
* ARP responder flow
* UCAST\_TO\_TUN （table 20） flow

有和没有 l2pop 的效果：



**4.2 过程实验**

1. def tunnel\_sync(self) 函数除了上报自己的 local\_ip 外不再自己见 tunnels，一切等 l2pop 的通知。

2. 在 compute1 上添加第一个虚机 81.1.180.8

neutron-server：

* 通知 compute1：{'segment\_id': 6L, 'ports': {u'10.0.1.21': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:87:40:f3', u'81.1.180.1']]}, 'network\_type': u'gre'}}
* 通知所有 agent： {'segment\_id': 6L, 'ports': {u'10.0.1.31': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:b3:e7:7a', u'81.1.180.8']]}, 'network\_type': u'gre'}}

compute1：

* 添加和网络节点的tunnel options: {df\_default="true", in\_key=flow, local\_ip="10.0.1.31", out\_key=flow, remote\_ip="10.0.1.21"}
* 添加到网段网关的 Unicast flow：table=20, n\_packets=0, n\_bytes=0, idle\_age=130, priority=2,dl\_vlan=2,dl\_dst=fa:16:3e:87:40:f3 actions=strip\_vlan,set\_tunnel:0x6,output:4
* 增加网段 81.1.180.1 网关的 ARP flows：table=21, n\_packets=0, n\_bytes=0, idle\_age=130, priority=1,arp,dl\_vlan=2,arp\_tpa=81.1.180.1 actions=move:NXM\_OF\_ETH\_SRC[]->NXM\_OF\_ETH\_DST[],mod\_dl\_src:fa:16:3e:87:40:f3,load:0x2->NXM\_OF\_ARP\_OP[],move:NXM\_NX\_ARP\_SHA[]->NXM\_NX\_ARP\_THA[],move:NXM\_OF\_ARP\_SPA[]->NXM\_OF\_ARP\_TPA[],load:0xfa163e8740f3->NXM\_NX\_ARP\_SHA[],load:0x5101b401->NXM\_OF\_ARP\_SPA[],IN\_PORT
* 修改 Flood flow

compute 2 节点：因为它上面还没有运行虚机，所以不做操作。

3. 在 compute 2 上添加一个虚机 81.1.180.9

neutron server：

* 通知 compute 2 ： {'segment\_id': 6L, 'ports': {u'10.0.1.31': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:b3:e7:7a', u'81.1.180.8']], u'10.0.1.21': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:87:40:f3', u'81.1.180.1']]}, 'network\_type': u'gre'}}
* 通知所有 agent： {'segment\_id': 6L, 'ports': {u'10.0.1.39': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:73:49:41', u'81.1.180.9']]}, 'network\_type': u'gre'}

compute1：

* 建立 tunnel（ID 5）：  {df\_default="true", in\_key=flow, local\_ip="10.0.1.31", out\_key=flow, remote\_ip="10.0.1.39"}
* 增加 arp responder flow（compute2 上新的虚机 IP -> MAC）：table=21, n\_packets=0, n\_bytes=0, idle\_age=79, priority=1,arp,dl\_vlan=2,arp\_tpa=81.1.180.9 actions=move:NXM\_OF\_ETH\_SRC[]->NXM\_OF\_ETH\_DST[],mod\_dl\_src:fa:16:3e:73:49:41,load:0x2->NXM\_OF\_ARP\_OP[],move:NXM\_NX\_ARP\_SHA[]->NXM\_NX\_ARP\_THA[],move:NXM\_OF\_ARP\_SPA[]->NXM\_OF\_ARP\_TPA[],load:0xfa163e734941->NXM\_NX\_ARP\_SHA[],load:0x5101b409->NXM\_OF\_ARP\_SPA[],IN\_PORT
* 增加 unicast flow （新虚机的 MAC -> Tunnel port）：table=20, n\_packets=0, n\_bytes=0, idle\_age=79, priority=2,dl\_vlan=2,dl\_dst=fa:16:3e:73:49:41 actions=strip\_vlan,set\_tunnel:0x6,output:5
* 添加新的 Tunnel port 到 Flood flow：table=22, n\_packets=13, n\_bytes=1717, idle\_age=255, hard\_age=78, dl\_vlan=2 actions=strip\_vlan,set\_tunnel:0x6,output:5,output:4

compute2：

* 建立和计算节点以及compute1的tunnel：options: {df\_default="true", in\_key=flow, local\_ip="10.0.1.39", out\_key=flow, remote\_ip="10.0.1.21"}，options: {df\_default="true", in\_key=flow, local\_ip="10.0.1.39", out\_key=flow, remote\_ip="10.0.1.31"}
* 增加 ARP flow（compute 1 上的虚机的 MAC -> IP）：table=21, n\_packets=0, n\_bytes=0, idle\_age=268, priority=1,arp,dl\_vlan=2,arp\_tpa=81.1.180.8 actions=move:NXM\_OF\_ETH\_SRC[]->NXM\_OF\_ETH\_DST[],mod\_dl\_src:fa:16:3e:b3:e7:7a,load:0x2->NXM\_OF\_ARP\_OP[],move:NXM\_NX\_ARP\_SHA[]->NXM\_NX\_ARP\_THA[],move:NXM\_OF\_ARP\_SPA[]->NXM\_OF\_ARP\_TPA[],load:0xfa163eb3e77a->NXM\_NX\_ARP\_SHA[],load:0x5101b408->NXM\_OF\_ARP\_SPA[],IN\_PORT
* 增加 Unicast flow （compute 1 上的虚机 MAC -> Tunnel port）：table=20, n\_packets=0, n\_bytes=0, idle\_age=268, priority=2,dl\_vlan=2,dl\_dst=fa:16:3e:b3:e7:7a actions=strip\_vlan,set\_tunnel:0x6,output:4
* 增加 ARP flow（新虚机的网关的 MAC -> IP） table=21, n\_packets=0, n\_bytes=0, idle\_age=268, priority=1,arp,dl\_vlan=2,arp\_tpa=81.1.180.1 actions=move:NXM\_OF\_ETH\_SRC[]->NXM\_OF\_ETH\_DST[],mod\_dl\_src:fa:16:3e:87:40:f3,load:0x2->NXM\_OF\_ARP\_OP[],move:NXM\_NX\_ARP\_SHA[]->NXM\_NX\_ARP\_THA[],move:NXM\_OF\_ARP\_SPA[]->NXM\_OF\_ARP\_TPA[],load:0xfa163e8740f3->NXM\_NX\_ARP\_SHA[],load:0x5101b401->NXM\_OF\_ARP\_SPA[],IN\_PORT
* 修改 Flood flow（添加到 Compute 1 的 port）：table=22, n\_packets=13, n\_bytes=1717, idle\_age=128, dl\_vlan=2 actions=strip\_vlan,set\_tunnel:0x6,output:5,output:4

3. 删除 compute1 上的一个vm（也是唯一的一个）

neutron server：

* 通知所有 agent： {'segment\_id': 6L, 'ports': {u'10.0.1.31': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:b3:e7:7a', u'81.1.180.8']]}, 'network\_type': u'gre'}

compute 1：

* 因为没有别的虚机了，删除所有 tunnel ports
* 修改或者删除 ARP, Unicast 和 Flood flows

compute 2：

* 删除了 compute1 的 tunnel
* 删除该虚机对应的 ARP flow

 4. 在 compute1 上创建第一个不同网络的虚机

neutron server：

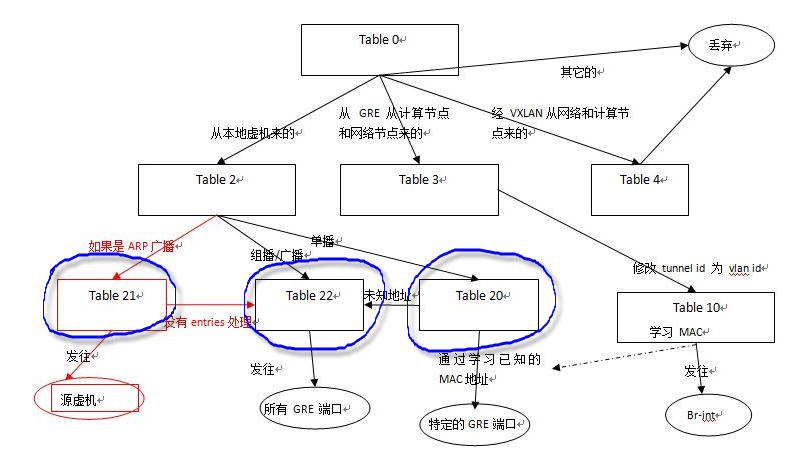
* 通知 compute 1： {u'e2022937-ec2a-467a-8cf1-f642a3f777b6': {'segment\_id': 4L, 'ports': {u'10.0.1.21': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:90:e5:50', u'91.1.180.1'], [u'fa:16:3e:17:c9:26', u'90.1.180.1'], [u'fa:16:3e:69:92:30', u'90.1.180.3'], [u'fa:16:3e:69:92:30', u'91.1.180.2']]}, 'network\_type': u'gre'}}
* 通知所有 agent：{u'e2022937-ec2a-467a-8cf1-f642a3f777b6': {'segment\_id': 4L, 'ports': {u'10.0.1.31': [['00:00:00:00:00:00', '0.0.0.0'], [u'fa:16:3e:e9:ee:0c', u'91.1.180.9']]}, 'network\_type': u'gre'}}

compute 1：建立和网络节点的 tunnel port；更新 Flood flows；添加 ARP flows

compute 2：没什么action，因为该节点上没有新建虚机的网络内的虚机

过程的大概说明：

* 虚机在收到 fannout FDB entries 后，检查其中每个 port 的 network\_id（即 “segment\_id”）。如果本机上有该 network 内的 port，那么就处理 entries 中的 “ports”部分；否则，不处理该 entries。
* 因此，当计算节点上没有运行任何虚机时，不会建立任何 tunnel。如果两个虚机上有相同网络内的虚机，那么建立会建立 tunnel。
* 这种机制能实时建立 tunnel port，Flood entry （创建 Tunnel port 同时添加到 Flood output ports 列表）， Unicast flow （虚机和网关 MAC -> Tunnel port） 和 ARP Responder entry  （虚机和网关 MAC -> IP）。下图中的蓝色部分的流表都会被及时更新。
* Neutron server 在端口创建/删除/修改时，如果是该节点上的第一个虚机，首先发送直接消息；然后发通知消息给所有的计算和网络节点。



**4.3 性能**

**4.3.1 MQ 性能问题**

    应该说 l2pop 的原理和实现都很直接，但是在大规模部署环境中，这种通知机制（通知所有的 ML2 Agent 节点）可能会给 MQ 造成很大的负担。一旦 MQ 不能及时处理消息，虚机之间的网络将受到影响。下面是 l2pop 中通知机制代码：

[复制代码](javascript:void(0);)

def \_\_init\_\_(self, topic=topics.AGENT):

super(L2populationAgentNotifyAPI, self).\_\_init\_\_(

topic=topic, default\_version=self.BASE\_RPC\_API\_VERSION)

self.topic\_l2pop\_update = topics.get\_topic\_name(topic, topics.L2POPULATION, topics.UPDATE)

def \_notification\_fanout(self, context, method, fdb\_entries):

self.fanout\_cast(context, self.make\_msg(method, fdb\_entries=fdb\_entries), topic=self.topic\_l2pop\_update)

def \_notification\_host(self, context, method, fdb\_entries, host):

self.cast(context, self.make\_msg(method, fdb\_entries=fdb\_entries), topic='%s.%s' % (self.topic\_l2pop\_update, host))

def add\_fdb\_entries(self, context, fdb\_entries, host=None):

if fdb\_entries:

if host:

self.\_notification\_host(context, 'add\_fdb\_entries',fdb\_entries, host) #cast 给指定 host

else:

self.\_notification\_fanout(context, 'add\_fdb\_entries', fdb\_entries) #fanout 给所有计算和网络节点

[复制代码](javascript:void(0);)

    这段代码是说，l2pop 采用的 MQ topic 是 “L2POPULATION”，消息通知采用 fanout 或者 cast 机制。如果是 fanout 的话，消息将发到所有的 ML2 agent 节点。这样的话，其覆盖面就有些过于广泛了，就这个问题有人提了一个 [ticket](https://bugs.launchpad.net/neutron/+bug/1337717)，官方答复是 work as design，要改的话只能是添加 new feature 了。

**4.3.2 大规模网络环境中节点上的 OpenFlow flows 过多**

    不知道这个数目有没有上限？数目很多的情况下会不会有性能问题？OVS 有没有处理能力上限？这些问题也许得在实际的生产环境中才能得到证实和答案。