[**理解 OpenStack 高可用（HA）（3）：Neutron 分布式虚拟路由（Neutron Distributed Virtual Routing）**](http://www.cnblogs.com/sammyliu/p/4713562.html)

本系列会分析OpenStack 的高可用性（HA）概念和解决方案：

（1）[OpenStack 高可用方案概述](http://www.cnblogs.com/sammyliu/p/4741967.html%20)

（2）[Neutron L3 Agent HA - VRRP （虚拟路由冗余协议）](http://www.cnblogs.com/sammyliu/p/4692081.html%20)

（3）[Neutron L3 Agent HA - DVR （分布式虚机路由器）](http://www.cnblogs.com/sammyliu/p/4713562.html%20)

（4）[Pacemaker 和 OpenStack Resource Agent （RA）](http://www.cnblogs.com/sammyliu/p/5025362.html%20)

（5）[RabbitMQ HA](http://www.cnblogs.com/sammyliu/p/4730517.html)

（6）[MySQL HA](http://www.cnblogs.com/sammyliu/p/4995868.html%20)

    Neutron 作为 OpenStack 一个基础性关键服务，高可用性（HA）和扩展性是它的基本需求之一。对 neutron server 来说，因为它是无状态的，我们可以使用负载均衡器（Load Balancer）比如 HAProxy 来实现其 HA 和扩展性；对于 Neutron L3 Agent 来说，一个带外（Out-of-band）的 HA 实现方案可以使用 PeaceMaker，但是这会大大增加系统的复杂性，另一个就是之前介绍的 VRRP，但是它也存在不少问题：（1）需要额外的硬件来组成 VRRP 组，这会带来成本增加（2）它无法解决扩展性问题，东-西和南-北网络流量都需要经过活动的 VRRP Router。而 Juno 中引入的 DVR 功能正是用来解决这两点不足的。

**1.基础知识**

**1.1 路由 （Routing）**

**1.1.1 路由策略 （使用 ip rule 命令操作路由策略数据库）**

    基于策略的路由比传统路由在功能上更强大，使用更灵活，它使网络管理员不仅能够根据目的地址而且能够根据报文大小、应用或IP源地址等属性来选择转发路径。

ip rule 命令：

* Usage: ip rule [ list | add | del ] SELECTOR ACTION （add 添加；del 删除； llist 列表）
* SELECTOR := [ from PREFIX 数据包源地址] [ to PREFIX 数据包目的地址] [ tos TOS 服务类型][ dev STRING 物理接口] [ pref NUMBER ] [fwmark MARK iptables 标签]
* ACTION := [ table TABLE\_ID 指定所使用的路由表] [ nat ADDRESS 网络地址转换][ prohibit 丢弃该表| reject 拒绝该包| unreachable 丢弃该包]
* [ flowid CLASSID ]
* TABLE\_ID := [ local | main | default | new | NUMBER ]

例子：

* ip rule add from 192.203.80/24 table inr.ruhep prio 220 通过路由表 inr.ruhep 路由来自源地址为192.203.80/24的数据包
* ip rule add from 193.233.7.83 nat 192.203.80.144 table 1 prio 320 把源地址为193.233.7.83的数据报的源地址转换为192.203.80.144，并通过表1进行路由

在 Linux 系统启动时，内核会为路由策略数据库配置三条缺省的规则：

* 0 匹配任何条件 查询路由表local(ID 255) 路由表local是一个特殊的路由表，包含对于本地和广播地址的高优先级控制路由。rule 0非常特殊，不能被删除或者覆盖。
* 32766 匹配任何条件 查询路由表main(ID 254) 路由表main(ID 254)是一个通常的表，包含所有的无策略路由。系统管理员可以删除或者使用另外的规则覆盖这条规则。
* 32767 匹配任何条件 查询路由表default(ID 253) 路由表default(ID 253)是一个空表，它是为一些后续处理保留的。对于前面的缺省策略没有匹配到的数据包，系统使用这个策略进行处理。这个规则也可以删除。

不要混淆路由表和策略：规则指向路由表，多个规则可以引用一个路由表，而且某些路由表可以没有策略指向它。如果系统管理员删除了指向某个路由表的所有规则，这个表就没有用了，但是仍然存在，直到里面的所有路由都被删除，它才会消失。

（[资料来源](http://blog.chinaunix.net/uid-27575921-id-5060664.html)）

**1.1.2 路由表 （使用 ip route 命令操作静态路由表）**

    所谓路由表，指的是路由器或者其他互联网网络设备上存储的表，该表中存有到达特定网络终端的路径，在某些情况下，还有一些与这些路径相关的度量。路由器的主要工作就是为经过路由器的每个数据包寻找一条最佳的传输路径，并将该数据有效地传送到目的站点。由此可见，选择最佳路径的策略即路由算法是路由器的关键所在。为了完成这项工作，在路由器中保存着各种传输路径的相关数据——路由表（Routing Table），供路由选择时使用，表中包含的信息决定了数据转发的策略。打个比方，路由表就像我们平时使用的地图一样，标识着各种路线，路由表中保存着子网的标志信息、网上路由器的个数和下一个路由器的名字等内容。路由表根据其建立的方法，可以分为动态路由表和静态路由表。

    linux 系统中，可以自定义从 1－252个路由表，其中，linux系统维护了4个路由表：

* 0#表： 系统保留表
* 253#表： defulte table 没特别指定的默认路由都放在改表
* 254#表： main table 没指明路由表的所有路由放在该表
* 255#表： locale table 保存本地接口地址，广播地址、NAT地址 由系统维护，用户不得更改

路由表的查看可有以下二种方法：

* ip route list table table\_number
* ip route list table table\_name

路由表序号和表名的对应关系在 /etc/iproute2/rt\_tables 文件中，可手动编辑。路由表添加完毕即时生效，下面为实例：

* ip route add default via 192.168.1.1 table 1 在一号表中添加默认路由为192.168.1.1
* ip route add 192.168.0.0/24 via 192.168.1.2 table 1 在一号表中添加一条到192.168.0.0网段的路由为192.168.1.2

以下面的路由表为例：

[复制代码](javascript:void(0);)

Destination Netmask Gateway Interface Metric

0.0.0.0 0.0.0.0 192.168.123.254 192.168.123.88 1 #缺省路由，目的地址不在本路由表中的数据包，经过本机的 192.168.123.88 接口发到下一个路由器 192.168.123.254

127.0.0.0 255.0.0.0 127.0.0.1 127.0.0.1 1 #发给本机的网络包

192.168.123.0 255.255.255.0 192.168.123.68 192.168.123.68 1 #直连路由。目的地址为 192.168.123.0/24 的包发到本机 192.168.123.88 接口

192.168.123.88 255.255.255.255 127.0.0.1 127.0.0.1 1 #目的地址为 192.168.123.88的包是发给本机的包

192.168.123.255 255.255.255.255 192.168.123.88 192.168.123.88 1 #广播包的网段是 192.168.123.0/24，经过 192.168.123.88 接口发出去

224.0.0.0 224.0.0.0 192.168.123.88 192.168.123.88 1 #多播包，经过 192.168.123.88 接口发出去

255.255.255.255 255.255.255.255 192.168.123.68 192.168.123.68 1 #全网广播包

Default Gateway: 192.168.123.254

[复制代码](javascript:void(0);)

各字段说明：

* destination：目的网段
* mask：与网络目标地址相关联的网掩码（又称之为子网掩码）。子网掩码对于 IP 网络地址可以是一适当的子网掩码，对于主机路由是 255.255.255.255 ，对于默认路由是 0.0.0.0。如果忽略，则使用子网掩码 255.255.255.255。定义路由时由于目标地址和子网掩码之间的关系，目标地址不能比它对应的子网掩码更为详细。换句话说，如果子网掩码的一位是 0，则目标地址中的对应位就不能设置为 1。
* interface：到达该目的地的本路由器的出口ip
* gateway： 下一跳路由器入口的 ip，路由器通过 interface 和 gateway 定义一调到下一个路由器的链路。通常情况下，interface 和 gateway 是同一网段的metric 跳数，该条路由记录的质量，一般情况下，如果有多条到达相同目的地的路由记录，路由器会采用metric值小的那条路由

根据子网掩码，可以将路由分为三种类型：

* 主机路由：机路由是路由选择表中指向单个IP地址或主机名的路由记录。主机路由的Flags字段为H。

Destination Gateway Genmask Flags Metric Ref Use Iface

----------- ------- ------- ----- ------ --- --- -----

10.0.0.10 192.168.1.1 255.255.255.255 UH 0 0 0 eth0

* 网络路由：网络路由是代表主机可以到达的网络。网络路由的Flags字段为N。例如，在下面的示例中，本地主机将发送到网络192.19.12的数据包转发到IP地址为192.168.1.1的路由器。

Destination Gateway Genmask Flags Metric Ref Use Iface

----------- ------- ------- ----- ----- --- --- -----

192.19.12 192.168.1.1 255.255.255.0 UN 0 0 0 eth0

* 默认路由：当主机不能在路由表中查找到目标主机的IP地址或网络路由时，数据包就被发送到默认路由（默认网关）上。默认路由的Flags字段为G。

Destination Gateway Genmask Flags Metric Ref Use Iface

----------- ------- ------- ----- ------ --- --- -----

default 192.168.1.1 0.0.0.0 UG 0 0 0 eth0

设置和查看路由表都可以用 route 命令，设置内核路由表的命令格式是：route [add|del] [-net|-host] target [netmask Nm] [gw Gw] [[dev] If]

其中：

* add : 添加一条路由规则，del : 删除一条路由规则，-net : 目的地址是一个网络，-host : 目的地址是一个主机，target : 目的网络或主机
* netmask : 目的地址的网络掩码，gw : 路由数据包通过的网关，dev : 为路由指定的网络接口

比如：

* route add 0.0.0.0 mask 0.0.0.0 192.168.12.1
* route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 metric 7

（数据来源：[（1）](http://baike.baidu.com/view/149989.htm)，[（2）](http://wenku.baidu.com/view/beb6a260783e0912a2162aa6.html?re=view)，[（3）](http://blog.csdn.net/shiniji_hh1126/article/details/6000810)）

关于 src 属性：

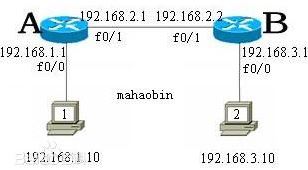
当一个主机有多个网卡配置了多个 IP 的时候，对于它产生的网络包，可以在路由选择时设置源 IP 地址。比如：

ip route add 78.22.45.0/24 via 10.45.22.1 src 10.45.22.12 （发到 78.22.45.0/24 网段的网络包，下一跳的路由器 IP 是 10.45.22.1，包的源IP地址设为10.45.22.12）。

要注意的是，src 选项只会影响该 host 上产生的网络包。如果是一个被路由的外来包，明显地它已经带有了一个源 IP 地址，这时候，src 参数的配置对它没有任何影响，除非你使用 NAT 来改变它。对 Neutron 来说，qrouter 和 qif namespace 中的路由表中的 src 都没有实际意义，因为它们只会处理外来的网络包。

**1.1.3 路由分类之静态路由**

  静态路由是指由用户或网络管理员手工配置的路由信息。当网络的拓扑结构或链路的状态发生变化时，网络管理员需要手工去修改路由表中相关的静态路由信息。静态路由信息在缺省情况下是私有的，不会传递给其他的路由器。当然，网管员也可以通过对路由器进行设置使之成为共享的。静态路由一般适用于比较简单的网络环境，在这样的环境中，网络管理员易于清楚地了解网络的拓扑结构，便于设置正确的路由信息。



以上面的拓扑结构为例，在没有配置路由的情况下，计算机1 和 2 无法互相通信，因为 1 发给 2 的包在到达路由器 A 后，它不知道怎么转发它。B 也同样。管理员可以配置如下的静态路由来实现 1 和 2 之间的通信：

计算机配置默认网关：

* 计算机1 上：route add default gw 192.168.1.1
* 计算机2 上：route add default gw 192.168.3.1

路由器配置：

* R1 上：ip route 192.168.3.0 255.255.255.0 f0/1 （意思为：目标网络地址为 192.168.3.0/24 的数据包，经过 f0/1 端口发出）
* R2 上：ip route 192.168.1.0 255.255.255.0 f0/1 （意思为：目标网络地址为 192.168.1.0/24 的数据包，经过 f0/1 端口发出）

或者

* R1 上：ip route 192.168.3.0 255.255.255.0 192.168.2.2 （意思为：要去 192.168.3.0/24 的数据包，下一路由器 IP 地址为 192.168.2.2）
* R2 上：ip route 192.168.1.0 255.255.255.0 192.168.2.1

（来源：<http://baike.baidu.com/view/911.htm>）

**1.1.4 路由分类之动态路由**

    动态路由是指路由器能够自动地建立自己的路由表，并且能够根据实际情况的变化适时地进行调整。它是与静态路由相对的一个概念，指路由器能够根据路由器之间的交换的特定路由信息自动地建立自己的路由表，并且能够根据链路和节点的变化适时地进行自动调整。当网络中节点或节点间的链路发生故障，或存在其它可用路由时，动态路由可以自行选择最佳的可用路由并继续转发报文。

常见的动态路由协议有以下几个：路由信息协议（RIP）、OSPF(Open Shortest Path First开放式最短路径优先）、IS-IS（Intermediate System-to-Intermediate System，中间系统到中间系统）、边界网关协议（BGP）是运行于 TCP 上的一种自治系统的路由协议。

（来源：<http://baike.baidu.com/view/897.htm>）

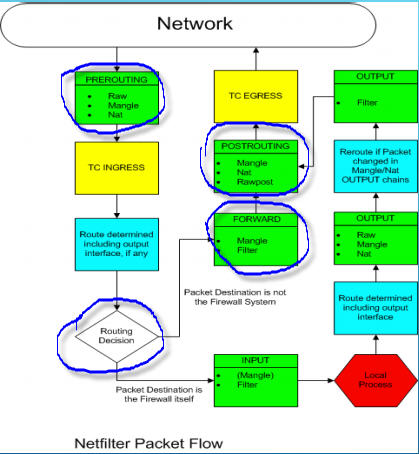
**1.1.5 ip rule，ip route，iptables 三者之间的关系**

以一例子来说明：公司内网要求192.168.0.100 以内的使用 10.0.0.1 网关上网 （电信），其他IP使用 20.0.0.1 （网通）上网。

1. 首先要在网关服务器上添加一个默认路由，当然这个指向是绝大多数的IP的出口网关：ip route add default gw 20.0.0.1
2. 之后通过 ip route 添加一个路由表：ip route add table 3 via 10.0.0.1 dev ethX (ethx 是 10.0.0.1 所在的网卡, 3 是路由表的编号)
3. 之后添加 ip rule 规则：ip rule add fwmark 3 table 3 （fwmark 3 是标记，table 3 是路由表3 上边。 意思就是凡事标记了 3 的数据使用 table3 路由表）
4. 之后使用 iptables 给相应的数据打上标记：iptables -A PREROUTING -t mangle -i eth0 -s 192.168.0.1 - 192.168.0.100 -j MARK --set-mark 3

因为 mangle 的处理是优先于 nat 和 fiter 表的，所以在数据包到达之后先打上标记，之后再通过 ip rule 规则，对应的数据包使用相应的路由表进行路由，最后读取路由表信息，将数据包送出网关。

（来源：[使用 ip route ， ip rule ， iptables 配置策略路由](http://blog.sina.com.cn/s/blog_659b48590100n2q6.html)。[这里](http://wenku.baidu.com/link?url=Zny1LXKB8HfXzS_SC48EXJRiIg1y8fRI6Zu7WMEOAvsuPo7ktKIhW4cV0D6pM09DHvwthU5jtX2a1TD3pu0jLqHRcBn2W9ve7WSz4k8TMvC) 有一个更详细的例子）



这里可以看出 Netfilter 处理网络包的先后顺序：接收网络包，先 DNAT，然后查路由策略，查路由策略指定的路由表做路由，然后 SNAT，再发出网络包。

**1.1.6 Traceroute 工具**

    我们在 linux 机器上，使用 traceroute 来获知从你的计算机到互联网另一端的主机是走的什么路径。当然每次数据包由某一同样的出发点（source）到达某一同样的目的地(destination)走的路径可能会不一样，但基本上来说大部分时候所走的路由是相同的。在 MS Windows 中该工具为 tracert。 在大多数情况下，我们会在linux主机系统下，直接执行命令行：traceroute hostname；而在Windows系统下是执行tracert的命令： tracert hostname。

* 命令格式：traceroute [参数] [主机]
* 命令功能：traceroute 指令让你追踪网络数据包的路由途径，预设数据包大小是 40Bytes，用户可另行设置。
* 具体参数格式：traceroute [-dFlnrvx][-f<存活数值>][-g<网关>...][-i<网络界面>][-m<存活数值>][-p<通信端口>][-s<来源地址>][-t<服务类型>][-w<超时秒数>][主机名称或IP地址][数据包大小]
* 命令参数：
  + -d 使用Socket层级的排错功能，-f 设置第一个检测数据包的存活数值TTL的大小，-F 设置勿离断位，-g 设置来源路由网关，最多可设置8个，-i 使用指定的网络界面送出数据包，-I 使用ICMP回应取代UDP资料信息，-m 设置检测数据包的最大存活数值TTL的大小，-n 直接使用IP地址而非主机名称。
  + -p 设置UDP传输协议的通信端口，-r 忽略普通的Routing Table，直接将数据包送到远端主机上，-s 设置本地主机送出数据包的IP地址，-t 设置检测数据包的TOS数值。
  + -v 详细显示指令的执行过程，-w 设置等待远端主机回报的时间，-x 开启或关闭数据包的正确性检验。

（1）例子

[复制代码](javascript:void(0);)

[root@localhost ~]# traceroute www.baidu.com

traceroute to www.baidu.com (61.135.169.125), 30 hops max, 40 byte packets

1 192.168.74.2 (192.168.74.2) 2.606 ms 2.771 ms 2.950 ms

2 211.151.56.57 (211.151.56.57) 0.596 ms 0.598 ms 0.591 ms

3 211.151.227.206 (211.151.227.206) 0.546 ms 0.544 ms 0.538 ms

4 210.77.139.145 (210.77.139.145) 0.710 ms 0.748 ms 0.801 ms

5 202.106.42.101 (202.106.42.101) 6.759 ms 6.945 ms 7.107 ms

6 61.148.154.97 (61.148.154.97) 718.908 ms \* bt-228-025.bta.net.cn (202.106.228.25) 5.177 ms

7 124.65.58.213 (124.65.58.213) 4.343 ms 4.336 ms 4.367 ms

8 202.106.35.190 (202.106.35.190) 1.795 ms 61.148.156.138 (61.148.156.138) 1.899 ms 1.951 ms

9 \* \* \*

30 \* \* \*

[复制代码](javascript:void(0);)

说明：

* 记录按序列号从1开始，每个纪录就是一跳 ，每跳表示一个网关，我们看到每行有三个时间，单位是 ms，其实就是 -q 的默认参数。
* 探测数据包向每个网关发送三个数据包后，网关响应后返回的时间；如果您用 traceroute -q 4 www.58.com ，表示向每个网关发送4个数据包。
* 有时我们 traceroute 一台主机时，会看到有一些行是以星号表示的。出现这样的情况，可能是防火墙封掉了ICMP 的返回信息，所以我们得不到什么相关的数据包返回数据。
* 有时我们在某一网关处延时比较长，有可能是某台网关比较阻塞，也可能是物理设备本身的原因。当然如果某台 DNS 出现问题时，不能解析主机名、域名时，也会 有延时长的现象；您可以加-n 参数来避免DNS解析，以IP格式输出数据。
* 如果在局域网中的不同网段之间，我们可以通过 traceroute 来排查问题所在，是主机的问题还是网关的问题。如果我们通过远程来访问某台服务器遇到问题时，我们用到traceroute 追踪数据包所经过的网关，提交IDC服务商，也有助于解决问题；但目前看来在国内解决这样的问题是比较困难的，就是我们发现问题所在，IDC服务商也不可能帮助我们解决。

（2）原理

 Traceroute 程序的设计是利用 ICMP 及 IP header 的 TTL（Time To Live）栏位（field）。

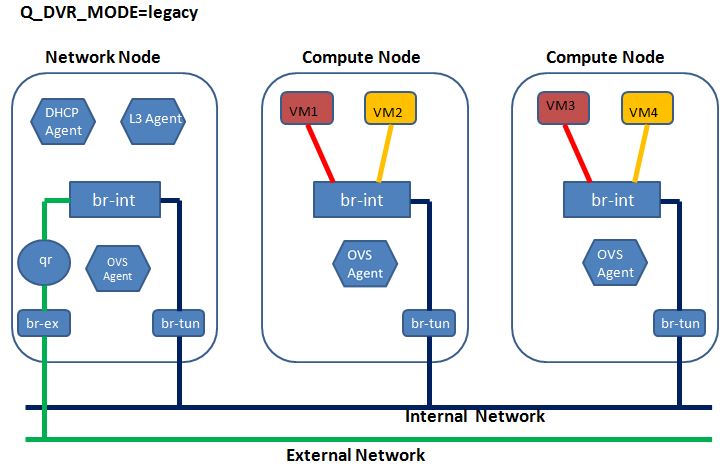
1. 首先，traceroute 送出一个 TTL 是 1 的 IP datagram（其实，每次送出的为3个40字节的包，包括源地址，目的地址和包发出的时间标签）到目的地，当路径上的第一个路由器（router）收到这个datagram 时，它将TTL减1。此时，TTL变为0了，所以该路由器会将此 datagram 丢掉，并送回一个「ICMP time exceeded」消息（包括发IP包的源地址，IP包的所有内容及路由器的IP地址），traceroute 收到这个消息后，便知道这个路由器存在于这个路径上。
2. 接着，traceroute 再送出另一个TTL 是 2  的datagram，发现第2 个路由器......
3. 然后，traceroute  每次将送出的 datagram 的 TTL  加1来发现另一个路由器，这个重复的动作一直持续到某个datagram 抵达目的地。当datagram到达目的地后，该主机并不会送回ICMP time exceeded消息，因为它已是目的地了，那么traceroute如何得知目的地到达了呢？

    Traceroute 在送出 UDP datagrams 到目的地时，它所选择送达的 port number 是一个一般应用程序都不会用的号码（30000 以上），所以当此 UDP datagram 到达目的地后该主机会送回一个「ICMP port unreachable」的消息，而当traceroute 收到这个消息时，便知道目的地已经到达了。所以traceroute 在Server端也是没有所谓的Daemon 程式。Traceroute提取发 ICMP TTL 到期消息设备的 IP 地址并作域名解析。每次 ，Traceroute 都打印出一系列数据,包括所经过的路由设备的域名及 IP地址,三个包每次来回所花时间。

（以上资料来自互联网）

**2. Neutron 的传统和 DVR Router**

**2.1 传统（Legacy） Router**



关于这种 Router，可以参考我的另一篇文章 [Neutron 理解 (6): Neutron 是怎么实现虚拟三层网络的 [How Neutron implements virtual L3 network]](http://www.cnblogs.com/sammyliu/p/4636091.html)。

**2.2 DVR 对 L3 Agent 的影响**

    通过使用 DVR，三层的转发（L3 Forwarding）和 NAT 功能都会被分布到计算节点上，这意味着计算节点也有了网络节点的功能。但是，DVR 依然不能消除集中式的 Virtual Router，这是为了节省宝贵的 IPV4 公网地址，所有依然将 SNAT 放在网络节点上提供。这样，计算和网络节点就看起来如下：

* 网络节点：提供 南-北 SNAT，即在不使用浮动 IP 时，虚机访问外网的网络得经过网络节点。也就是说，网络节点依然必须走传统的 HA 解决方法，比如 VRRP 和 PeaceMaker。但可惜的是，Juno 版本不支持同时使用 HA 和 DVR。
* 计算节点：提供 南-北 DNAT， 即外网访问虚机的网络流量得经过计算节点；以及 东-西 转发，即虚机之间的网络经过计算节点。因为所有计算节点的参与，这部分的网络处理负载也就自然地被均衡了。

**2.3 DVR 对 L2 Agent 的影响**

 DVR 对 L2 Agent 的影响主要有：

* DVR 新创建安的各个 network namespace 需要被 plug 到 OVS bridge
* OVS flows 需要更新来支持 DVR

详细的分析在 <https://wiki.openstack.org/wiki/Neutron/DVR_L2_Agent> 以及本文下文。

**3. 安装和功能分析**

**3.1 安装和配置**

Juno 版本中 DVR 的要求如下：

* 使用 ML2 plugin
* 使用 L2pop mechanism driver
* 使用 Openvswitch mechanism driver, 安装 OVS agent 在所有的计算节点上
* 所有的计算节点连接外网
* Juno 中只支持 Tunnel 虚拟网络模式 (VXLAN or GRE)。 Kilo 版本中会增加 VLAN 模式的支持。

**3.1.1 安装**

使用两个结算节点。在每个计算节点上安装并配置 L3 Agent：

（1）修改系统配置

root@compute2:/var/log/nova# vi /etc/sysctl.conf

root@compute2:/var/log/nova# sysctl -p

net.ipv4.ip\_forward = 1

net.ipv4.conf.all.rp\_filter = 0

net.ipv4.conf.default.rp\_filter = 0

（2）安装 neutron-l3-agent： apt-get install neutron-l3-agent

（3）增加一块访问外网的网卡 eth3

（4）创建 OVS bridge：ovs-vsctl add-br br-ex

（5）增加新的网卡到该 bridge 上：ovs-vsctl add-port br-ex eth3

（6）配置 L3 Agent

interface\_driver = neutron.agent.linux.interface.OVSInterfaceDriver

use\_namespaces = True

external\_network\_bridge = br-eth3

（7）配置 Metadata Agent

**3.1.2 配置 DVR**

|  |  |  |  |
| --- | --- | --- | --- |
| 控制节点 | /etc/neutron/neutron.conf | router\_distributed = True | 注意：需要设置 l3\_ha = false 来禁用 VRRP |
| 计算节点 | /etc/neutron/l3\_agent.ini  /etc/neutron/plugins/ml2/ml2\_conf.ini | agent\_mode = dvr  enable\_distributed\_routing = True |  |
| 网络节点 | /etc/neutron/l3\_agent.ini  /etc/neutron/plugins/ml2/ml2\_conf.ini | agent\_mode = dvr\_snat  enable\_distributed\_routing = True |  |

注意还需要配置使用 l2\_population。

对普通用户来说，neutron 会根据管理员在控制节点上的配置项 router\_distributed 的值，来决定是创建普通 Router 还是 DVR Router；对管理员来说，还可以使用 --distributed {True,False} 参数来指定是否创建 DVR 模式的 router。

 经过以上配置后的 neutron agent如下。可以看到，除了网络节点外，所有的计算介绍上也部署了 L3 Agent 和 Metadata Agent。

[复制代码](javascript:void(0);)

s1@controller:~$ neutron agent-list

+--------------------------------------+--------------------+----------+-------+----------------+---------------------------+

| id | agent\_type | host | alive | admin\_state\_up | binary |

+--------------------------------------+--------------------+----------+-------+----------------+---------------------------+

| 04c360d0-3066-4f04-9af2-d4ef8586ad2b | L3 agent | network | :-) | True | neutron-l3-agent |

| 2ef04905-4a37-4de8-a8f0-9c6488a592b7 | Open vSwitch agent | network | :-) | True | neutron-openvswitch-agent |

| 3f307355-2167-4b15-affa-9f296f698752 | DHCP agent | network | :-) | True | neutron-dhcp-agent |

| 54609006-a769-4b17-b175-1a834e6e7a26 | Open vSwitch agent | compute1 | :-) | True | neutron-openvswitch-agent |

| 90c87c01-1cd1-48b0-8369-30f44c058574 | Loadbalancer agent | network | :-) | True | neutron-lbaas-agent |

| 951b8efc-1f2c-4a51-84d1-2261ff31c12c | Metadata agent | compute2 | :-) | True | neutron-metadata-agent |

| 99d13b27-89f8-4abe-bc03-3f69f5e7e0cc | Metadata agent | network | :-) | True | neutron-metadata-agent |

| aa8cf021-7f3d-4667-9d92-4d77d4c4fb59 | L3 agent | compute2 | :-) | True | neutron-l3-agent |

| beec232b-48d7-4424-83e2-8cc4e49ec339 | L3 agent | compute1 | :-) | True | neutron-l3-agent |

| d65bbede-4b1d-4914-8c8b-ab591975828f | Metadata agent | compute1 | :-) | True | neutron-metadata-agent |

| e3f83dcf-27f2-4c91-bead-adebcab1e3c7 | Open vSwitch agent | compute2 | :-) | True | neutron-openvswitch-agent |

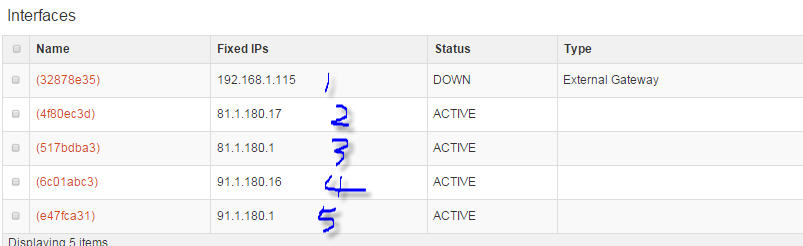
+--------------------------------------+--------------------+----------+-------+----------------+---------------------------+

[复制代码](javascript:void(0);)

**3.2  DVR Router 流程**

**3.2.1 创建 DVR Router**

（1）创建如下的 DVR Router：



可以看到该 router 被分布在neutron network 节点和计算节点上：

[复制代码](javascript:void(0);)

s1@controller:~$ neutron l3-agent-list-hosting-router dvr-r1

+--------------------------------------+----------+----------------+-------+

| id | host | admin\_state\_up | alive |

+--------------------------------------+----------+----------------+-------+

| 04c360d0-3066-4f04-9af2-d4ef8586ad2b | network | True | :-) |

| beec232b-48d7-4424-83e2-8cc4e49ec339 | compute1 | True | :-) |

+--------------------------------------+----------+----------------+-------+

[复制代码](javascript:void(0);)

（2）网络节点上，创建了 SNAT network namespace。该 netns 中，对router 的每一个网络，都有一个 qg 或者 sg interface：

[复制代码](javascript:void(0);)

root@network:/home/s1# ip netns exec snat-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip addr

42: qg-32878e35-a2: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 1，连接外网

link/ether fa:16:3e:4a:40:48 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.115/24 brd 192.168.1.255 scope global qg-32878e35-a2

valid\_lft forever preferred\_lft forever

44: sg-4f80ec3d-f2: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 2，连接内网1

link/ether fa:16:3e:82:a9:ca brd ff:ff:ff:ff:ff:ff

inet 81.1.180.17/24 brd 81.1.180.255 scope global sg-4f80ec3d-f2

valid\_lft forever preferred\_lft forever

46: sg-6c01abc3-5d: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 4，连接内网2

link/ether fa:16:3e:47:55:00 brd ff:ff:ff:ff:ff:ff

inet 91.1.180.16/24 brd 91.1.180.255 scope global sg-6c01abc3-5d

valid\_lft forever preferred\_lft forever

root@network:/home/s1# ip netns exec snat-e8f12f7a-6938-4e65-88c4-97e4cb211b27 iptables -t nat -S

-A POSTROUTING -j neutron-l3-agent-POSTROUTING

-A POSTROUTING -j neutron-postrouting-bottom

-A neutron-l3-agent-POSTROUTING ! -i qg-32878e35-a2 ! -o qg-32878e35-a2 -m conntrack ! --ctstate DNAT -j ACCEPT

-A neutron-l3-agent-snat -s 81.1.180.0/24 -j SNAT --to-source 192.168.1.115

-A neutron-l3-agent-snat -s 91.1.180.0/24 -j SNAT --to-source 192.168.1.115

-A neutron-postrouting-bottom -j neutron-l3-agent-snat

[复制代码](javascript:void(0);)

以及 qrouter network namespace：

[复制代码](javascript:void(0);)

root@network:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip addr

43: qr-517bdba3-b1: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 3

link/ether fa:16:3e:63:3b:4c brd ff:ff:ff:ff:ff:ff

inet 81.1.180.1/24 brd 81.1.180.255 scope global qr-517bdba3-b1

valid\_lft forever preferred\_lft forever

45: qr-e47fca31-db: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 5

link/ether fa:16:3e:a9:da:b5 brd ff:ff:ff:ff:ff:ff

inet 91.1.180.1/24 brd 91.1.180.255 scope global qr-e47fca31-db

valid\_lft forever preferred\_lft forever

root@network:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 iptables -t nat -S

-A neutron-l3-agent-PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 9697

-A neutron-l3-agent-snat -j neutron-l3-agent-float-snat

-A neutron-postrouting-bottom -j neutron-l3-agent-snat

[复制代码](javascript:void(0);)

（3）在计算节点 compute1 上创建一个虚机

在 compute1 上生成了一个 qrouter network namespace：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip addr

29: qr-517bdba3-b1: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 3，作为 81.1.180.1/24 网段内的虚机的默认网关

link/ether fa:16:3e:63:3b:4c brd ff:ff:ff:ff:ff:ff

inet 81.1.180.1/24 brd 81.1.180.255 scope global qr-517bdba3-b1

valid\_lft forever preferred\_lft forever

inet6 fe80::f816:3eff:fe63:3b4c/64 scope link

valid\_lft forever preferred\_lft forever

31: qr-e47fca31-db: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default #对应 port 5，作为 91.1.180.1/24 网段内的虚机的默认网关

link/ether fa:16:3e:a9:da:b5 brd ff:ff:ff:ff:ff:ff

inet 91.1.180.1/24 brd 91.1.180.255 scope global qr-e47fca31-db

valid\_lft forever preferred\_lft forever

inet6 fe80::f816:3eff:fea9:dab5/64 scope link

valid\_lft forever preferred\_lft forever

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 iptables -t nat -S  
-A neutron-l3-agent-PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 9697

[复制代码](javascript:void(0);)

（4）给虚机分配一个浮动IP 后，compute1 上出现了 fip network namespace。该 netns 的命名规则是 fip-<external net id>，这里的 external net 是指该浮动IP所在的外网的：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 ip addr

2: fpr-e8f12f7a-6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000 #该 port 和 qrouter 中的 rpf port 是一对 veth pair，各自配置了自己的 IP 地址

link/ether 9a:d0:23:ba:d2:02 brd ff:ff:ff:ff:ff:ff

inet 169.254.31.29/31 scope global fpr-e8f12f7a-6

valid\_lft forever preferred\_lft forever

inet6 fe80::98d0:23ff:feba:d202/64 scope link

valid\_lft forever preferred\_lft forever

32: fg-9eeb3fb1-25: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UNKNOWN group default

link/ether fa:16:3e:22:2d:90 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.117/24 brd 192.168.1.255 scope global fg-9eeb3fb1-25

valid\_lft forever preferred\_lft forever

inet6 fe80::f816:3eff:fe22:2d90/64 scope link

valid\_lft forever preferred\_lft forever

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 ip route  
default via 192.168.1.1 dev fg-9eeb3fb1-25  
169.254.31.28/31 dev fpr-e8f12f7a-6 proto kernel scope link src 169.254.31.29  
192.168.1.0/24 dev fg-9eeb3fb1-25 proto kernel scope link src 192.168.1.117  
192.168.1.116 via 169.254.31.28 dev fpr-e8f12f7a-6

[复制代码](javascript:void(0);)

而且，compute1 上的 qrouter netns 中的变化：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip addr

2: rfp-e8f12f7a-6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000 #该 port 和 fip 中的 pfr port 是一对 veth，各自配置了自己的 IP 地址，该地址来自 169.254.x.y 网段

link/ether 42:81:66:11:60:66 brd ff:ff:ff:ff:ff:ff

inet 169.254.31.28/31 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

inet 192.168.1.116/32 brd 192.168.1.116 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever  
  
NAP iptables 规则：

-A neutron-l3-agent-OUTPUT -d 192.168.1.116/32 -j DNAT --to-destination 81.1.180.18

-A neutron-l3-agent-POSTROUTING ! -i rfp-e8f12f7a-6 ! -o rfp-e8f12f7a-6 -m conntrack ! --ctstate DNAT -j ACCEPT

-A neutron-l3-agent-PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 9697

-A neutron-l3-agent-PREROUTING -d 192.168.1.116/32 -j DNAT --to-destination 81.1.180.18

-A neutron-l3-agent-float-snat -s 81.1.180.18/32 -j SNAT --to-source 192.168.1.116  
  
以及一条路由策略。具体见下文。

[复制代码](javascript:void(0);)

但是 neutron 网络节点上的 qrouter 上并没有增加该浮动 IP和相应的 NAT iptables 规则。

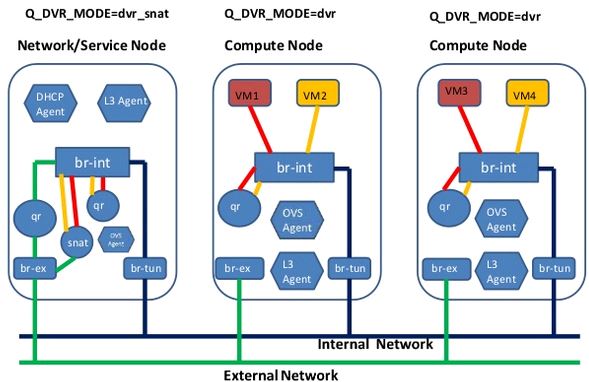
**3.2 网络包走向分析**

不同网段内的虚机之间或者虚机和计算机之间的网络流向可以分为几类：

* 3.2.1 SNAT：当数据包离开 rouer 的 external device 时改变它的源 IP 地址。这在没有浮动IP 时虚机访问外网的情况下使用。
* 3.2.2/3 FIP：有时候也称为 DNAT。当虚机的固定 IP分配了浮动 IP 的时候，虚机和外网中的虚机通信的时候使用。
* 3.2.4 不同服务器上不同网段内的虚机之间的通信
* 3.2.5 同一个服务器上不同网段内的虚机之间的通信

**3.2.1 SNAT：虚机访问外网（没有分配浮动IP 的情况下）**

Neuron 布网：



虚机（81.1.180.18）发出给外网中机器（192.168.1.20）的包，因为是跨网段的，先发给 compute1 上的 qrouter 的 qr-517bdba3-b1 interface。然后，qruoter 查路由表：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip rule

0: from all lookup local

32766: from all lookup main

32767: from all lookup default

1359066113: from 81.1.180.1/24 lookup 1359066113

1526838273: from 91.1.180.1/24 lookup 1526838273  
  
root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route list table 1359066113 #只有一个默认路由    
default via 81.1.180.17 dev qr-517bdba3-b1 #经过 qr 端口发到下一个路由器 81.1.180.17

[复制代码](javascript:void(0);)

查表结果是经过 interface qr-517bdba3-b1 将包发到下一个路由器 81.1.180.17，而这个IP 在 neutron 网络节点上的 SNAT netns 的 44: sg-4f80ec3d-f2 interface。在这里，先做 SNAT：

root@network:/home/s1# ip netns exec snat-e8f12f7a-6938-4e65-88c4-97e4cb211b27 iptables -t nat -S  
-A neutron-l3-agent-POSTROUTING ! -i qg-32878e35-a2 ! -o qg-32878e35-a2 -m conntrack ! --ctstate DNAT -j ACCEPT

-A neutron-l3-agent-snat -s 81.1.180.0/24 -j SNAT --to-source 192.168.1.115 # 命中这条，把网络包的 Src IP 修改为 192.168.1.115

-A neutron-l3-agent-snat -s 91.1.180.0/24 -j SNAT --to-source 192.168.1.115

-A neutron-postrouting-bottom -j neutron-l3-agent-snat

然后查路由表确定下一跳：

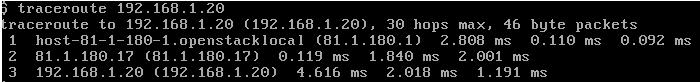
[复制代码](javascript:void(0);)

root@network:/home/s1# ip netns exec snat-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip rule show #这里没定义额外的路由策略，直接查 main 路由表  
0: from all lookup local  
32766: from all lookup main  
32767: from all lookup default

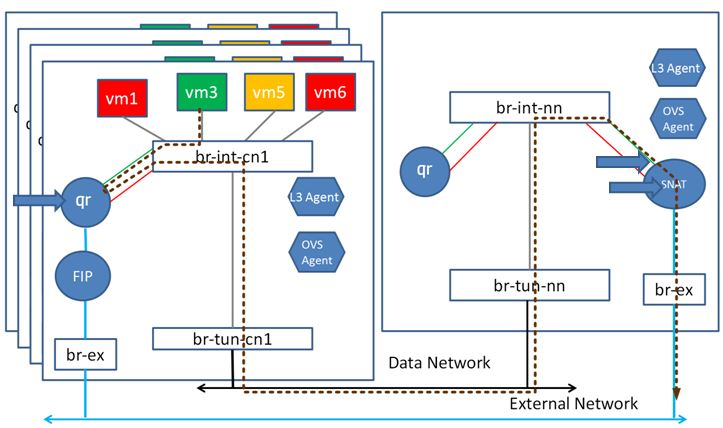
root@network:/home/s1# ip netns exec snat-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route list table main  
default via 192.168.1.1 dev qg-32878e35-a2  
81.1.180.0/24 dev sg-4f80ec3d-f2 proto kernel scope link src 81.1.180.17   
91.1.180.0/24 dev sg-6c01abc3-5d proto kernel scope link src 91.1.180.16  
192.168.1.0/24 dev qg-32878e35-a2 proto kernel scope link src 192.168.1.115 #根据目的地址，命中这条，网络包从 qg 端口发出

[复制代码](javascript:void(0);)

  结论：在没有设置浮动 IP （SNAT）的情况下，虚机访问外网时，虚机发出的网络包首先经过其所在的服务器上的 qrouter 做路由选择（81.1.180.1），然后再经过 neutron network 节点上的 snat network namespace （81.1.180.17）出去到外网。这个结论也和 traceroute 的结果互相验证：

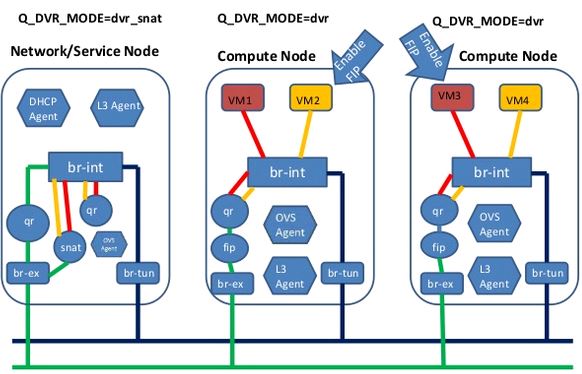


完整过程：



**3.2.2 FIP：在虚机 81.1.180.18 上添加浮动IP，从它 ping 外网**

Neutron 组网：



添加浮动 IP 后，虚机所在的主机上的 qrouter netns 上添加了浮动IP：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip addr

3: rfp-e8f12f7a-6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 46:ef:97:f4:4d:ff brd ff:ff:ff:ff:ff:ff

inet 169.254.31.238/31 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

inet 192.168.1.112/32 brd 192.168.1.112 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

[复制代码](javascript:void(0);)

还增加了一条路由策略及路由表：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip rule

0: from all lookup local

32766: from all lookup main

32767: from all lookup default

32769: from 81.1.180.18 lookup 16 #每个固定 IP 对应一条路由策略，都查 ID 为 16 的路由表

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route list table 16  
default via 169.254.31.239 dev rfp-e8f12f7a-6

[复制代码](javascript:void(0);)

（1）网络包从虚机触发，进入本服务器所在的 qrouter 的 qr interface，首先经过 DNAT，没有命中，然后查路由表，local，main，default 中没有命中的路由规则，查表 16，命中默认路由，需要经过 rfp 端口发到下一个路由器 169.254.31.239。

（2）rfp 端口是这样子：

3: rfp-e8f12f7a-6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 46:ef:97:f4:4d:ff brd ff:ff:ff:ff:ff:ff

inet 169.254.31.238/31 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

inet 192.168.1.112/32 brd 192.168.1.112 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

（3）该 rfp 和 fip netns 上的端口通过 veth 直接连接：

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 ip addr2:   
fpr-e8f12f7a-6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 9a:4d:66:1c:b5:2b brd ff:ff:ff:ff:ff:ff

inet 169.254.31.239/31 scope global fpr-e8f12f7a-6

valid\_lft forever preferred\_lft forever

（4）看 fip netns 的路由表，决定将其从 fg interface 发出去。

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 ip route

default via 192.168.1.1 dev fg-4a292fe1-58

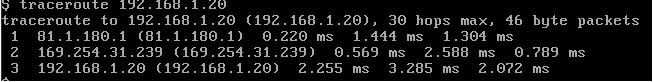
169.254.31.238/31 dev fpr-e8f12f7a-6 proto kernel scope link src 169.254.31.239

192.168.1.0/24 dev fg-4a292fe1-58 proto kernel scope link src 192.168.1.118

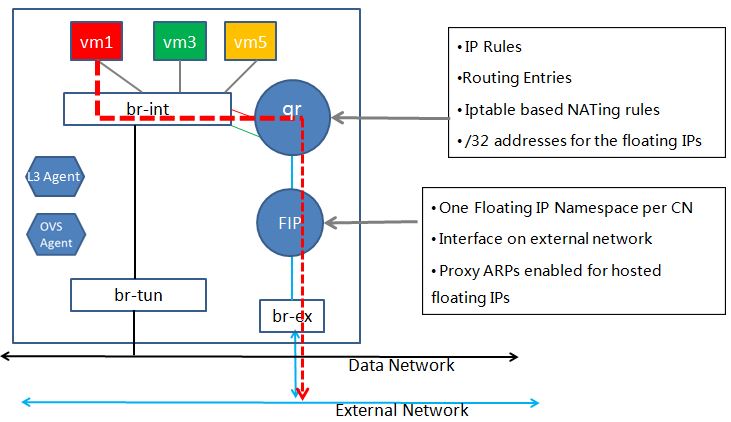
192.168.1.112 via 169.254.31.238 dev fpr-e8f12f7a-6

（5）再经过 SNAT，经 Src 地址换成浮动 IP 地址：-A neutron-l3-agent-float-snat -s 81.1.180.18/32 -j SNAT --to-source 192.168.1.112

 结论：配有浮动IP的虚机上 ping 外网，依次经过虚机所在的服务器上的 qrouter （81.1.180.1）和 fip netns （169.254.31.239） 到外网机器（192.168.1.20）。



完整路径：



**3.2.3 FIP：外网机器通过虚机的浮动 IP 访问虚机**

    外网中的机器首先要通过 ARP 获取虚机浮动 IP 对应的 MAC 地址。浮动 IP 并没有配置在 fip 的端口上，因此 fip 无法直接响应 ARP 请求，那怎么办呢？Neutron 在 fip NS 的 fg 端口上配置了 arp proxy，这样，fip 既可以响应它自己的 interface 上的 IP 地址的 ARP 请求，也可以响应能通过它路由到的 IP 地址的 ARP 请求。

（1）从下图可见，fip netns 的 fg-4a292fe1-58 interface 上配置了 ARP 代理：

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 sysctl net.ipv4.conf.fg-4a292fe1-58.proxy\_arp

net.ipv4.conf.fg-4a292fe1-58.proxy\_arp = 1  
  
而 qrouter 的 interface 没有设置 arp proxy：

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 sysctl net.ipv4.conf.rfp-e8f12f7a-6.proxy\_arp

net.ipv4.conf.rfp-e8f12f7a-6.proxy\_arp = 0

（2）fip netns 收到 ARP 请求后，将其 fg interface 的 MAC 地址返回。其实这是一个 MAC 地址欺骗，但是。。这就是一个 arp proxy 所起的作用。

外网中的机器获取到虚机浮动 IP 的 MAC 地址后，发出 ICMP 网络包（Dest IP: 192.168.1.112，Souce IP: 192.168.1.20，Dest MAC: fa:16:3e:95:55:29

（fip 的 fg interface 的 MAC 地址），Source MAC: MAC of 192.168.1.20）：

（1）网络包经过 br-ex，被 fip 的 fg 端口收到，查路由表，命中最后一条路由，从其 fpr interface 发出，到达 169.254.31.238.

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 ip route

default via 192.168.1.1 dev fg-4a292fe1-58

169.254.31.238/31 dev fpr-e8f12f7a-6 proto kernel scope link src 169.254.31.239

192.168.1.0/24 dev fg-4a292fe1-58 proto kernel scope link src 192.168.1.118

192.168.1.112 via 169.254.31.238 dev fpr-e8f12f7a-6

17:38:41.462365 9a:4d:66:1c:b5:2b > 46:ef:97:f4:4d:ff, ethertype IPv4 (0x0800), length 98: (tos 0x0, flags [DF], proto ICMP (1), length 84)

192.168.1.20 > 192.168.1.112: ICMP echo request, id 3138, seq 180, length 64

（2）被 veth 另一端的 qrouter 的 rfp-e8f12f7a-6 interface 收到。

3: rfp-e8f12f7a-6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 46:ef:97:f4:4d:ff brd ff:ff:ff:ff:ff:ff

inet 169.254.31.238/31 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

inet 192.168.1.112/32 brd 192.168.1.112 scope global rfp-e8f12f7a-6

valid\_lft forever preferred\_lft forever

（3）在 qrouter 上，首先做 DNAT：

-A neutron-l3-agent-PREROUTING -d 192.168.1.112/32 -j DNAT --to-destination 81.1.180.18。将目的 IP 由浮动 IP 修改为固定 IP。

（4）查 qrouter 的 main 路由表，命中第一条，从 qr-517bdba3-b1 发出

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route

81.1.180.0/24 dev qr-517bdba3-b1 proto kernel scope link src 81.1.180.1

91.1.180.0/24 dev qr-e47fca31-db proto kernel scope link src 91.1.180.1

169.254.31.238/31 dev rfp-e8f12f7a-6 proto kernel scope link src 169.254.31.238

17:43:21.363808 fa:16:3e:63:3b:4c > fa:16:3e:30:ee:23, ethertype IPv4 (0x0800), length 98: (tos 0x0, flags [DF], proto ICMP (1), length 84)

192.168.1.20 > 81.1.180.18: ICMP echo request, id 3202, seq 221, length 64

（5）SNAT：没有

（6）网络包经过 br-int，然后到达虚机

**3.2.4 不同服务器上不同网段上的虚机互相访问**

在另一个计算节点上新建虚机，固定 IP 为 90.1.180.6。从虚机 81.1.180.18 上 ping 它，看看网络包的走向。新建虚机后，compute 2 节点上也分布了 router 实例：

[复制代码](javascript:void(0);)

s1@controller:~$ neutron l3-agent-list-hosting-router dvr-r1

+--------------------------------------+----------+----------------+-------+

| id | host | admin\_state\_up | alive |

+--------------------------------------+----------+----------------+-------+

| 04c360d0-3066-4f04-9af2-d4ef8586ad2b | network | True | :-) |

| aa8cf021-7f3d-4667-9d92-4d77d4c4fb59 | compute2 | True | :-) |

| beec232b-48d7-4424-83e2-8cc4e49ec339 | compute1 | True | :-) |

+--------------------------------------+----------+----------------+-------+

[复制代码](javascript:void(0);)

 在 compute 2 上，创建了 qrouter network namespace，其中的配置和 compute 1 上的 qrouter 的配置完全相同。

（1）网络包离开 vm1，通过br-int，进入 compute 1 上的 qrouter 的 qr-517bdba3-b1，查 main 路由表，从 qr-e47fca31-db 出。

08:17:14.483282 fa:16:3e:30:ee:23 > fa:16:3e:63:3b:4c, ethertype IPv4 (0x0800), length 98: (tos 0x0, flags [DF], proto ICMP (1), length 84)  
81.1.180.18 > 90.1.180.6: ICMP echo request, id 12033, seq 4, length 64

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route list table main

81.1.180.0/24 dev qr-517bdba3-b1 proto kernel scope link src 81.1.180.1  
90.1.180.0/24 dev qr-f849ae46-48 proto kernel scope link src 90.1.180.1  
91.1.180.0/24 dev qr-e47fca31-db proto kernel scope link src 91.1.180.1  
169.254.31.238/31 dev rfp-e8f12f7a-6 proto kernel scope link src 169.254.31.238

08:18:30.555867 fa:16:3e:ec:f3:dd > fa:16:3e:13:93:0d, ethertype IPv4 (0x0800), length 98: (tos 0x0, flags [DF], proto ICMP (1), length 84)  
81.1.180.18 > 90.1.180.6: ICMP echo request, id 12033, seq 80, length 64

（2）网络包重新进入 br-int，被它的 flows 处理。

root@compute1:/home/s1# ovs-ofctl dump-flows br-inttable=0, n\_packets=0, n\_bytes=0, idle\_age=6318, priority=2,in\_port=5,dl\_src=fa:16:3f:12:a3:38 actions=resubmit(,1)

table=0, n\_packets=336, n\_bytes=32928, idle\_age=1, priority=2,in\_port=5,dl\_src=fa:16:3f:db:6f:73 actions=resubmit(,1)

table=0, n\_packets=12877, n\_bytes=1220950, idle\_age=1, priority=1 actions=NORMAL

table=1, n\_packets=0, n\_bytes=0, idle\_age=6309, priority=2,ip,dl\_vlan=2,nw\_dst=81.1.180.0/24 actions=strip\_vlan,mod\_dl\_src:fa:16:3e:63:3b:4c,output:4table=1, n\_packets=336, n\_bytes=32928, priority=4,dl\_vlan=2,dl\_dst=fa:16:3e:30:ee:23 actions=strip\_vlan,mod\_dl\_src:fa:16:3e:63:3b:4c,output:4table=1, n\_packets=0, n\_bytes=0, idle\_age=6319, priority=1 actions=drop

table=23, n\_packets=0, n\_bytes=0, idle\_age=6319, priority=0 actions=drop

（3）进入 br-tun，依次被其 flows 处理：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ovs-ofctl dump-flows br-tun  
NXST\_FLOW reply (xid=0x4):  
table=0, n\_packets=6423, n\_bytes=593912, idle\_age=0, priority=1,in\_port=1 actions=resubmit(,1)  
table=1, n\_packets=5751, n\_bytes=563598, idle\_age=0, priority=1,dl\_vlan=1,dl\_src=fa:16:3e:ec:f3:dd actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)  
table=2, n\_packets=5768, n\_bytes=565152, idle\_age=0, priority=0,dl\_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)  
table=20, n\_packets=0, n\_bytes=0, idle\_age=6900, priority=0 actions=resubmit(,22)  
table=20, n\_packets=11, n\_bytes=1022, idle\_age=3650, priority=2,dl\_vlan=2,dl\_dst=fa:16:3e:82:a9:ca actions=strip\_vlan,set\_tunnel:0x6,output:3  
table=20, n\_packets=5, n\_bytes=490, idle\_age=6411, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:47:55:00 actions=strip\_vlan,set\_tunnel:0x4,output:3  
table=20, n\_packets=0, n\_bytes=0, idle\_age=5820, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:54:f8:b8 actions=strip\_vlan,set\_tunnel:0x4,output:3  
table=20, n\_packets=1108, n\_bytes=108584, idle\_age=0, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:13:93:0d actions=strip\_vlan,set\_tunnel:0x4,output:2  
table=20, n\_packets=0, n\_bytes=0, idle\_age=6889, priority=2,dl\_vlan=2,dl\_dst=fa:16:3e:a1:76:41 actions=strip\_vlan,set\_tunnel:0x6,output:2  
table=20, n\_packets=1, n\_bytes=42, idle\_age=6685, priority=2,dl\_vlan=2,dl\_dst=fa:16:3e:c0:8f:2c actions=strip\_vlan,set\_tunnel:0x6,output:3  
table=20, n\_packets=0, n\_bytes=0, idle\_age=6782, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:69:92:30 actions=strip\_vlan,set\_tunnel:0x4,output:3

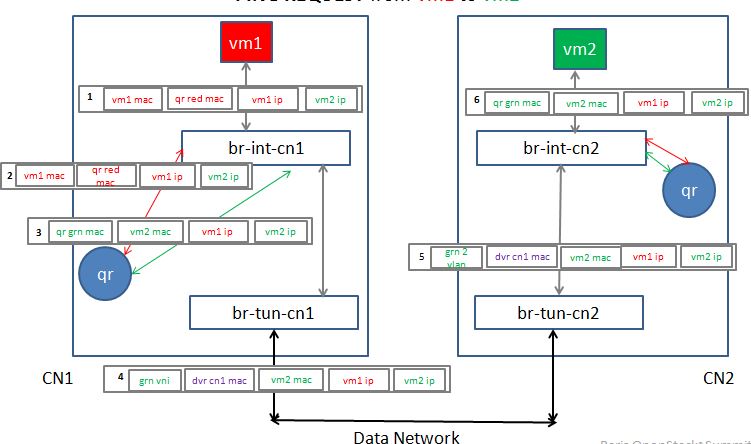
[复制代码](javascript:void(0);)

 从这里可以看出：

* table 1 将网络包的 src mac 修改为了 compute node 1 的 DVR MAC 地址。
* table 20 通过 l2population 获得了到虚机所在的租户网络内的所有存在的虚机的 MAC 地址和 Tunnel 端口号的映射关系。至此，网络包被打上了 Tunnel ID 4，进入 GRE Port 2.
* 其 GRE 隧道的另一头正是 compute 2.

（4）到了 compute 2 上，依次被 br-tun，br-int 处理，直到通过 tap 设备进入 vm 2。

小结：



**3.2.5 同一个服务器上不同网段上的虚机互相访问**

  这里只经过 3.2.4 中的 （1） 和 （2），网络包经过 br-int 直接进入 vm2.

**3.3 小结**

|  |  |  |
| --- | --- | --- |
| 网络流量类型 | 特征 | 转发机制 |
| 本地 | 源和目的 IP 在同一个 subnet 中，虚机在同一个计算节点上 | br-int 通过 MAC 地址学习直接转发网络包给目标虚机 |
| 远程 | 源和目的 IP 在同一个 subnet 中，虚机不在同一个计算节点上 | 标准转发，取决于网络的段类型 |
| 东-西 | 源和目的 IP 不在同一个 subnet 中 | 由源虚机所在计算节点上的 qrouter 负责转发 |
| SNAT | IP 不属于本地 router 的任何 subnet 中，而且虚机没有浮动IP | 由源虚机所在计算节点上的 qrouter 负责转发到 neutron 网络节点上 snat |
| FIP | IP 不属于本地 router 的任何 subnet 中，而且虚机有浮动IP | 由源虚机所在计算节点上的 qrouter转发到本地的 fip |

**4. 代码分析**

DVR 代码修改包括几部分：

1. DVR Router network namespace 的创建和删除
2. DVR Router 相关的 flows
3. DVR Router 的 ARP 表

**4.1 DVR Router 相关 network namespace 的创建和删除**

**4.1.1 qrouter 在计算和网络节点上的删除和创建**

    对于每一个 DVR Router，在每个分布了和 router 连接的网段内的虚机的计算节点上，都会有一个 qrouter 实例。两种情况下会将一个 DVR Router 部署到一个 L3 Agent 上：

1. 当一个子网 subnet 被加入到一个 DVR Router 时，DVR Router 会被分布到所有包含在该子网内的虚机的计算节点上。
   * 计算节点上的 L3 Agent 会收到一个通知，它会配置 router
   * OVS Agent 会将 router 的端口 plug 到 OVS Bridge 上，并且配置 flows
2. 当一个虚机被创建，而且虚机所在的计算节点上不存在该虚机所在 subnet 连接的 DVR Router 时。
3. 当与 DVR Router 相关的最后一个虚机被删除时，router namespace 会被从虚机所在的计算节点上删除。

**4.1.2 snat 在网络节点上的创建和删除**

创建：当设置 router 的 external gateway 时

删除：当删除 router 的 external gateway 时

**4.1.3 fip 在 计算节点上的创建和删除**

创建：当一个浮动 IP 被分配给一个虚机的时候，如果虚机所在的计算节点上 fip namespace 不存在，则创建它

删除：（1）当计算节点上最后一个使用浮动 IP 的虚机被删除后 （2）所有虚机的浮动 IP 被删除后

**4.2 DVR MAC 地址**

    前面提到过，分布到多个计算节点上的 qrouter 的interface 的 MAC 地址都相同。这在传统的网络中是不允许的，在 neutron 网络中某些时候也会导致一些问题。Neutron的做法是会向每个计算节点分配一个唯一的 DVR Host MAC 地址。当使用了 DVR 的 OVS Agent 启动的时候，它通过 RPC 去从 neutron server 上申请该 MAC 地址。该 MAC 地址会被保存在 DB 中，与该计算节点强绑定。比如：

[复制代码](javascript:void(0);)

MariaDB [neutron]> select \* from dvr\_host\_macs;

+----------+-------------------+

| host | mac\_address |

+----------+-------------------+

| network | fa:16:3f:12:a3:38 |

| compute1 | fa:16:3f:b2:34:82 |

| compute2 | fa:16:3f:db:6f:73 |

+----------+-------------------+

[复制代码](javascript:void(0);)

当数据包离开 DVR Router 经过 br-tun 时，OVS flows 会将 DVR Router interface 的源 MAC 地址替换成该 MAC 地址。

root@compute1:/home/s1# ovs-ofctl dump-flows br-tun | grep mod\_dl\_src  
cookie=0x0, duration=6989.394s, table=1, n\_packets=6405, n\_bytes=627690, idle\_age=510, priority=1,dl\_vlan=1,dl\_src=fa:16:3e:ec:f3:dd actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)  
cookie=0x0, duration=8055.165s, table=1, n\_packets=10, n\_bytes=980, idle\_age=4814, priority=1,dl\_vlan=2,dl\_src=fa:16:3e:63:3b:4c actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)  
cookie=0x0, duration=8059.947s, table=1, n\_packets=635, n\_bytes=26950, idle\_age=4843, priority=1,dl\_vlan=1,dl\_src=fa:16:3e:a9:da:b5 actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)

而 src mac 地址分别是 qrouter 上的作为默认各网段的默认网关的 mac 地址：

s1@controller:~$ neutron port-list | grep fa:16:3e:ec:f3:dd

| f849ae46-4819-45f5-a805-5970d4e31951 | | fa:16:3e:ec:f3:dd | {"subnet\_id": "f8841500-b392-4053-bda1-acf419f4a86e", "ip\_address": "90.1.180.1"}

s1@controller:~$ neutron port-list | grep fa:16:3e:63:3b:4c

| 517bdba3-b117-43ce-851b-bb1d039879dc | | fa:16:3e:63:3b:4c | {"subnet\_id": "4ec65731-35a5-4637-a59b-a9f2932099f1", "ip\_address": "81.1.180.1"}

s1@controller:~$ neutron port-list | grep fa:16:3e:a9:da:b5

| e47fca31-dbf6-47e5-9ccb-0950557a55e3 | | fa:16:3e:a9:da:b5 | {"subnet\_id": "13888749-12b3-462e-9afe-c527bd0a297e", "ip\_address": "91.1.180.1"}

因此，这里假设你不会将多个 DVR Router 连接到一个 subnet。当数据包达到该计算节点时，OVS flows 会将其源 MAC 地址替换成 VM gateway 的 MAC 地址。

DVR-MAC-ADDRESS 的更新是 neutron server 通过 RPC Notifier 做的。每当一个新的地址被分配后，它通知所有的 L3 Agent 节点做处理。

**4.3 DVR OVS flows**

使用 DVR Router 的计算节点上，br-int 和 br-tun 中的 flows 会有修改。具体请参见上文的 3.2.4 部分。

**4.3.1 br-int flows 的主要修改**

table 1： DVR\_TO\_SRC\_MAC

table 0：LOCAL\_SWITCHING

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ovs-ofctl dump-flows br-int

NXST\_FLOW reply (xid=0x4):  
#获取所有的 DVR\_MAC\_ADDRESS，然后  
# 从 patch-tun 进入的 src mac 为 network 节点的 DVR HOST MAC 的网络帧（也就是从 network 节点来的帧），重新提交 table 1 处理

table=0, n\_packets=0, n\_bytes=0, idle\_age=9734, priority=2,in\_port=5,dl\_src=fa:16:3f:12:a3:38 actions=resubmit(,1)  
# 从 compute 2 节点来的网络帧，提交到 table 1

table=0, n\_packets=2321, n\_bytes=227458, idle\_age=0, priority=2,in\_port=5,dl\_src=fa:16:3f:db:6f:73 actions=resubmit(,1)  
#本机上的虚机产生的网络帧，走常规的转发模式发到虚机或者 br-tun

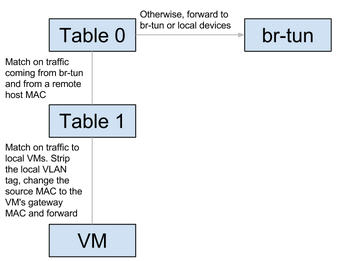
table=0, n\_packets=22797, n\_bytes=2176300, idle\_age=0, priority=1 actions=NORMAL  
  
#目标网段是 81.1.180.0/24 网段的帧，去掉 vlan tag，修改 src mac 为 qrouter 的 81.1.180.1 interface 的 mac 地址，发到虚机

table=1, n\_packets=0, n\_bytes=0, idle\_age=9725, priority=2,ip,dl\_vlan=2,nw\_dst=81.1.180.0/24 actions=strip\_vlan,mod\_dl\_src:fa:16:3e:63:3b:4c,output:4  
#目标网段是 90.1.180.0/24 网段的帧，去掉 vlan tag，修改 src mac 为qrouter 的 90.1.180.1 interface 的 mac 地址，发到虚机

table=1, n\_packets=0, n\_bytes=0, idle\_age=2268, priority=2,ip,dl\_vlan=1,nw\_dst=90.1.180.0/24 actions=strip\_vlan,mod\_dl\_src:fa:16:3e:ec:f3:dd,output:8  
#dst mac 地址为 vm1，去掉 vlan tag，修改 src mac，发到虚机table=1, n\_packets=2321, n\_bytes=227458, idle\_age=0, priority=4,dl\_vlan=2,dl\_dst=fa:16:3e:30:ee:23 actions=strip\_vlan,mod\_dl\_src:fa:16:3e:63:3b:4c,output:4  
#dst mac 为 vm2，则去掉 vlan tag，修改 src mac，发到虚机table=1, n\_packets=0, n\_bytes=0, idle\_age=2268, priority=4,dl\_vlan=1,dl\_dst=fa:16:3e:4a:22:ff actions=strip\_vlan,mod\_dl\_src:fa:16:3e:ec:f3:dd,output:8table=1, n\_packets=0, n\_bytes=0, idle\_age=9734, priority=1 actions=drop

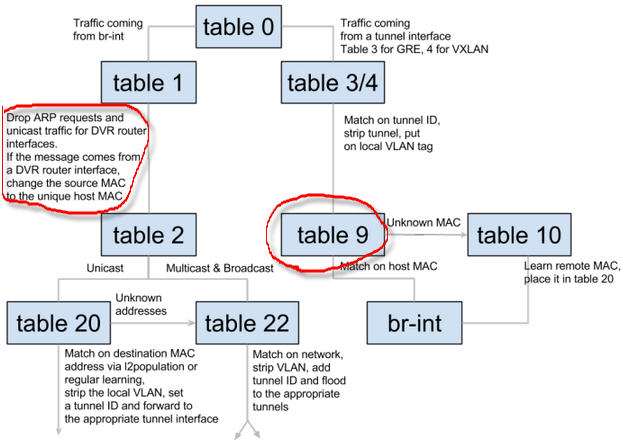
table=23, n\_packets=0, n\_bytes=0, idle\_age=9734, priority=0 actions=drop

[复制代码](javascript:void(0);)



**4.3.2 br-tun flows 主要的修改**

br-tun flows 的主要修改是增加了 table 1 和 9.



* 对于将要离开本机的网络帧：
  + Table 1 （DVR process Table）： 如果网络帧的 src mac 是本机 qrouter 上的 interface 的 mac 地址（dvr-router-intf-mac），将其修改为 DVR-compute-node-unique-mac，然后交给table 2 处理；其它的帧，交给 table 2.

table=1, n\_packets=8655, n\_bytes=848190, idle\_age=1, priority=1,dl\_vlan=1,dl\_src=fa:16:3e:ec:f3:dd actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)

table=1, n\_packets=10, n\_bytes=980, idle\_age=7987, priority=1,dl\_vlan=2,dl\_src=fa:16:3e:63:3b:4c actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)

table=1, n\_packets=635, n\_bytes=26950, idle\_age=8015, priority=1,dl\_vlan=1,dl\_src=fa:16:3e:a9:da:b5 actions=mod\_dl\_src:fa:16:3f:b2:34:82,resubmit(,2)

* + table 2：单播帧转 table 20；多播帧转 table 22

table=2, n\_packets=8673, n\_bytes=849786, idle\_age=1, priority=0,dl\_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)

table=2, n\_packets=675, n\_bytes=30798, idle\_age=3727, priority=0,dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,22)

* + table 20：将 vlan id 转化为 tunnel id，并根据处理进入本机的网络帧的时候学习到的 mac 地址和 tunnel port，查找网络帧的出口 tunnel port

table=20, n\_packets=0, n\_bytes=0, idle\_age=10156, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:54:f8:b8 actions=strip\_vlan,set\_tunnel:0x4,output:3table=20, n\_packets=4012, n\_bytes=393176, idle\_age=1, priority=2,dl\_vlan=1,dl\_dst=fa:16:3e:13:93:0d actions=strip\_vlan,set\_tunnel:0x4,output:2

* + table 22：将 vlan id 转化为 tunnel id，并泛洪到所有的 tunnel 端口

table=22, n\_packets=8, n\_bytes=1126, idle\_age=11018, hard\_age=5501, dl\_vlan=2 actions=strip\_vlan,set\_tunnel:0x6,output:3,output:2table=22, n\_packets=642, n\_bytes=27866, idle\_age=3727, hard\_age=5469, dl\_vlan=1 actions=strip\_vlan,set\_tunnel:0x4,output:3,output:2table=22, n\_packets=25, n\_bytes=1806, idle\_age=3772, priority=0 actions=drop

* 对于进入本机的网络帧：
  + table 0：交给 table 3 处理
  + table 3：只允许目的网络为本机上的虚机所在的网络的网络帧，修改其 vlan id，转 table 9

table=3, n\_packets=98, n\_bytes=6514, idle\_age=3731, priority=1,tun\_id=0x4 actions=mod\_vlan\_vid:1,resubmit(,9) # tunnel id 转换为 vlan id

table=3, n\_packets=3830, n\_bytes=375768, idle\_age=1, priority=1,tun\_id=0x6 actions=mod\_vlan\_vid:2,resubmit(,9) #tunnel id 转换为 vlan id

table=3, n\_packets=0, n\_bytes=0, idle\_age=11238, priority=0 actions=drop

* + Table 9 （DVR Learning blocker）：如果 src mac 是 DVR-Unique-MAC，不做 mac 学习，转发到 patch-int；否则，转到 table 10 做 mac 地址学习

table=9, n\_packets=0, n\_bytes=0, idle\_age=11236, priority=1,dl\_src=fa:16:3f:12:a3:38 actions=output:1 #从 network node 过来的帧，发到 br-inttable=9, n\_packets=3821, n\_bytes=374458, idle\_age=1, priority=1,dl\_src=fa:16:3f:db:6f:73 actions=output:1 #从 compute 2 node 过来的帧，发到 br-inttable=9, n\_packets=107, n\_bytes=7824, idle\_age=3731, priority=0 actions=resubmit(,10) #其余提交 table 10 处理，进行地址学习

* + table 10：mac 地址学习，结果存到 table 20

table=10, n\_packets=107, n\_bytes=7824, idle\_age=3731, 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

注意：在 table 20 中，除了自己通过 mac 地址学习学到的 mac 地址外，还需要借助 l2population 。这就是为什么 DVR 依赖于 l2population 的缘故。没有使用的话，网络包无法发到正确的 tunnel interface。

具体设置 OVS flows 的代码在 setup\_dvr\_flows\_on\_integ\_tun\_br 函数中。更详细的说明可以参考[官方文章](https://wiki.openstack.org/wiki/Neutron/DVR_L2_Agent)。

**4.4 qrouter 中的ARP 表**

虚机 vm1 需要通过 ARP 获取两种 mac 地址：

1. 当目标计算机（vm2）不在其同一个网段时，它需要获取默认网关的 mac 地址，这个将由 qrouter 直接相应 arp 请求。

2. 当目标计算机（vm2）在其同一个网段时，它需要直接获取 vm2 的 mac 地址。这个应该仍然是通过 ARP 广播获得。简单的做法是使用 arp responder。

qrouter 在做完 vm1 的网络包的路由后，将网络包从 vm2 所在网段的 interface 上发出前，需要获取 vm2 的 mac 地址。而这个是通过它查询自身的 ARP table 获得的。这是 compute 1 上 qrouter netns 中的 ip neighbour 表：

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip neigh

90.1.180.6 dev qr-f849ae46-48 lladdr fa:16:3e:13:93:0d PERMANENT  
81.1.180.10 dev qr-517bdba3-b1 lladdr fa:16:3e:c0:8f:2c PERMANENT  
90.1.180.3 dev qr-e47fca31-db lladdr fa:16:3e:69:92:30 PERMANENT

这里面可以看到大量的 PERMANENT MAC 地址。这是因为，L3 Agent 配置 DVR Router 的时候，它通过 RPC 从 neturon server 获取该 router 各 interface 的 subnet 中获取所有虚机的 MAC 地址。当一个 subnet 被加到 DVR Router 的时候，每个相关的 L3 Agent 都会被通知到，然后它通过 RPC 获取各 MAC 地址。当一个新的 port 被创建，或者 port 的 MAC 有更新的时候，所有相关的 L3 Agent 会被通知到去更新 ARP 表。

通过该由 L3 Agent 动态维护的 ARP 表，qrouter 就能直接查到它要通信的 interface 的 MAC 地址了，而不需要通过广播的方式去被动获取。具体原因是：

Why all of this complexity? Remember that the router’s MAC addresses are present on all hosts and cannot leave the host. If a router scheduled on host 1 generated an ARP request for a port scheduled on host 2, host 2 would receive the message and its virtual switches would suddenly re-learn a MAC they supposedly already know to be connected to br-int. They would see it coming from a tunnel connected to br-tun!

大致的更新过程为：

1. 每个L2 Agent 进程循环检查其管理的 port 的状态
2. 当 port 状态由 down 变为 up 时，它通过 RPC 通知 neutron server 该变化，neutron server 然后发出 fanout 通知其他的 L2 agent 去添加 arp entry （add\_arp\_entry），再调用 ip neigh replace方法在 qrouter network namespace 中 增加一个 arp entry
3. 当 port 状态由 up 变为 down 时，它通过 RPC 通知 neutron server 该变化，neutron server 然后发出 fanout 通知其他的 L2 agent 去添加 delete entry （del\_arp\_entry），再调用 ip neigh del 方法在 qrouter network namespace 中 删除该 arp entry

**4.5 ip rule 和 route 操作**

**4.5.1 增加一个 internal subnet 时**

在 qrouter namespace 上：

（1）计算该 subnet  cidr（81.1.180.1/24）的 index 1359066113，作为新增 ip rule 的优先级和路由表的名称。

（2）增加 default gateway，运行 ['ip route replace default via 81.1.180.17 dev qr-517bdba3-b1 table 1359066113]。这里的 81.1.180.17 正是 snat namespace 的 IP。

（3）增加 ip rule， 允许 [ip rule add from 81.1.180.1/24 lookup 1359066113 priority 1359066113]。这样就将该 subnet 中的虚机的网络帧转到 route table

（4）执行 ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 sysctl -w net.ipv4.conf.qr-517bdba3-b1.send\_redirects=0

效果如下：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip rule

0: from all lookup local

32766: from all lookup main

32767: from all lookup default

1359066113: from 81.1.180.1/24 lookup 1359066113

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route list table 1359066113

default via 81.1.180.17 dev qr-517bdba3-b1

[复制代码](javascript:void(0);)

 这样，当虚机还没有配置浮动IP时，访问外网的话，网络帧的路线为：vm ---- qrouter subnet 1 interface --- SNAT  ---- external port ----- pc

因此，当 router 上连接有多个 subnet 时，qrouter 中也有相应数量的 ip rule 和 routing table：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip rule

0: from all lookup local

32766: from all lookup main

32767: from all lookup default

1359066113: from 81.1.180.1/24 lookup 1359066113

1510061057: from 90.1.180.1/24 lookup 1510061057

1526838273: from 91.1.180.1/24 lookup 1526838273

[复制代码](javascript:void(0);)

**4.5.2 给虚机绑定浮动 IP 时**

在 qrouter namespace 中：

（1）增加 ip rule，通过运行 ['add', 'from', u'81.1.180.18', 'lookup', 16, 'priority', 32768]，其中，ID 16 为写死的，其优先级是从 32768 开始到 36768 这个区间内依次分配。

（2）在路由表 16 中添加路由项 default via 169.254.31.239 dev rfp-e8f12f7a-6。这使得虚机访问外网的网络包会通过 rfp-e8f12f7a-6 发到 169.254.31.239。而这个 IP 正是 fip 上 pfr 端口的IP。

在 fip namespace 中：

（1）增加 route：192.168.1.0/24 dev fg-6b744484-88  proto kernel  scope link  src 192.168.1.119。这使得访问外网机器的网络包能从  fg-6b744484-88 出去。

（2）增加 route：192.168.1.116 via 169.254.31.238 dev fpr-e8f12f7a-6。使得访问虚机的网络包会发给 169.254.31.238，进入 qrouter。这个 router 上的每个浮动 IP 有这么一条 route。

配置了两个浮动 IP 的情况下是这样的结果：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip rule

32768: from 81.1.180.18 lookup 16 #去 fip 的  
32769: from 90.1.180.8 lookup 16  #去 fip 的  
1359066113: from 81.1.180.1/24 lookup 1359066113 #去 snat 的  
1510061057: from 90.1.180.1/24 lookup 1510061057 #去 snat 的

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route list table 16

default via 169.254.31.239 dev rfp-e8f12f7a-6 # route 是一样的

root@compute1:/home/s1# ip netns exec fip-557e9f0c-9c66-46da-b289-218d49c218d2 ip route  
default via 192.168.1.1 dev fg-6b744484-88  
169.254.31.238/31 dev fpr-e8f12f7a-6 proto kernel scope link src 169.254.31.239  
192.168.1.0/24 dev fg-6b744484-88 proto kernel scope link src 192.168.1.119  
192.168.1.104 via 169.254.31.238 dev fpr-e8f12f7a-6 #每个浮动 IP 一个 route item  
192.168.1.116 via 169.254.31.238 dev fpr-e8f12f7a-6

[复制代码](javascript:void(0);)

这里能看到 qroute 的 ip rule 上，针对一个虚机/子网，有两条 rule，一条查路由表 16 到 fip，另一条查表到 snat。但是，在有浮动 IP 的情况下，前一条策略的优先级数值将小于后后一条的，这就决定了查路由表 16，数据包走 fip。

**4.5.3 qrouter 的 main 路由表**

main 路由表是为虚拟子网服务的，每个 subnet 对应一条路由规则，使得目的为每个 subnet 的网络包从指定的 qrouter 的 qr interface 上发出。

root@compute1:/home/s1# ip netns exec qrouter-e8f12f7a-6938-4e65-88c4-97e4cb211b27 ip route

81.1.180.0/24 dev qr-517bdba3-b1 proto kernel scope link src 81.1.180.1 #为到子网 1 中的虚机的网络包做路由

90.1.180.0/24 dev qr-f849ae46-48 proto kernel scope link src 90.1.180.1 #为到子网 2 中的虚机的网络包做路由

91.1.180.0/24 dev qr-e47fca31-db proto kernel scope link src 91.1.180.1

169.254.31.238/31 dev rfp-e8f12f7a-6 proto kernel scope link src 169.254.31.238 #为从 fip 进来的外网访问内部虚机的网络包做路由

**5. Neutron 其它服务与 DVR**

**5.1 FWaas DVR**

DVR 与传统的 FWaas 不兼容，因为它作用于neuron 网络节点上的 virtual router，过滤进出租户网络的网络包。传统的 FWaas 可以参考我的[另一篇文章](http://www.cnblogs.com/sammyliu/p/4675991.html)。

DVR 实现后，FWaas 需要做相应的修改。

官方文档在这里：

<https://wiki.openstack.org/wiki/Neutron/FWaaS/FWaaS-DVR>

Spec：<https://review.openstack.org/#/c/106225/9/specs/juno/neutron-dvr-fwaas.rst>

目标：FWaas 保持对 南-北流量做防火墙，而不影响东-西流量。

做法：Neutron 网络节点上的 FWaas Agent 安装在 SNAT network namespace 中；计算和网络节点上的 FWaas Agent 安装在 qrouter network namespace 中。

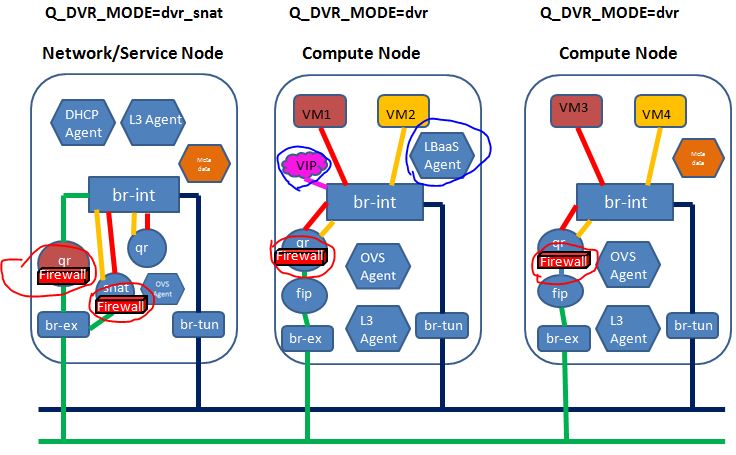
**5.2 VPNaas DVR**

Juno 版本中 VPNaas 不支持 DVR，只支持传统的 router。Kilo 版本中会实现 VPNaas 对 DVR  的支持。新的 VPN 服务只会在 dvr\_snat 节点上的 snat namespace 上运行。

**5.3 LBaas 与 DVR**

两者之间没有相互依赖关系，所以 DVR 对 LBaas 没有影响。

总体情况：



**6. 后续版本中 DVR 开发**

**6.1 Kilo 版本中**

* VPNaaS 对 DVR 的支持
* 从传统 router 迁移到 DVR router
* 网络节点上 HA + DVR 支持
* VLAN 支持

**6.2 Liberty 版本中**

* L3 Agent 重构
* 分布式 DHCP
* 性能调优
* 分布式 SNAT

    从第五和第六两个章节也可以看出，Juno 版本才添加的 DVR 功能还很不完善，难以满足生产环境的使用要求，主要是因为它还不支持目前实际部署中应该很广泛的 VLAN 组网模式，以及无法解决 HA 和 DVR 共存的问题。可喜的是这两个主要问题会在 K 版本中解决，因此 K 版本中的 DVR 至少可以用来做测试用了。等到 L 版本，实现分布式 DHCP 和 SNAT，以及性能优化以后，离生产环境的要求基本就差不多了。

参考链接：

* <http://assafmuller.com/>
* <http://assafmuller.com/2015/04/15/distributed-virtual-routing-snat/>
* <http://bingotree.cn/?p=706>
* <https://docs.google.com/document/d/1jCmraZGirmXq5V1MtRqhjdZCbUfiwBhRkUjDXGt5QUQ/edit>
* <http://assafmuller.com/2015/04/15/distributed-virtual-routing-overview-and-eastwest-routing/>
* <http://greenstack.die.upm.es/2015/04/15/distributed-virtual-routing-overview-and-eastwest-routing/>
* Architectural Overview of Distributed Virtual Routers in OpenStack Neutron by Vivekanandan Narasimhan
* Openstack 官网

分类: [Neutron](http://www.cnblogs.com/sammyliu/category/677824.html),[OpenStack](http://www.cnblogs.com/sammyliu/category/636967.html),[高可用（HA）](http://www.cnblogs.com/sammyliu/category/724243.html)