[**Neutron 理解 (8): Neutron 是如何实现虚机防火墙的 [How Neutron Implements Security Group]**](http://www.cnblogs.com/sammyliu/p/4658746.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)

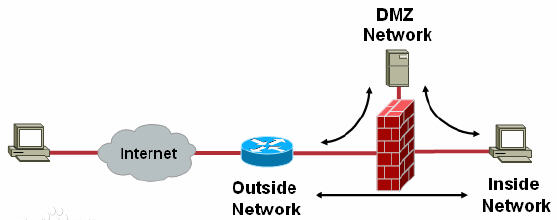
（12）[Neutron VRRP](http://www.cnblogs.com/sammyliu/p/4692081.html)

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

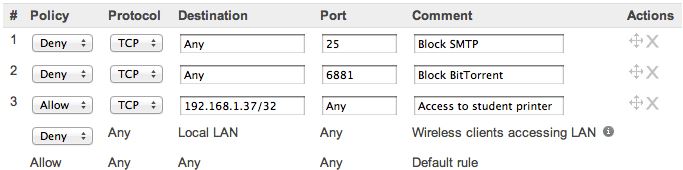
**1. 基础知识**

**1.1 防火墙（firewall）**

    防火墙是依照特定的规则来控制进出它的网络流量的网络安全系统。一个典型的场景是在一个受信任的内网和不受信任的外网比如 Internet 之间建立一个屏障。防火墙可以是电脑上运行的软件，也可以是独立的硬件设备。



与负载均衡器类似，按照其工作的网络层次，防火墙可以分为：

* 网络层防火墙（四层）：网络层防火墙可视为一种 IP 封包过滤器，运作在底层的TCP/IP协议堆栈上。我们可以以枚举的方式，只允许符合特定规则的封包通过，其余的一概禁止穿越防火墙（病毒除外，防火墙不能防止病毒侵入）。这些规则通常可以经由管理员定义或修改，不过某些防火墙设备可能只能套用内置的规则。 我们也能以另一种较宽松的角度来制定防火墙规则，只要封包不符合任何一项“否定规则”就予以放行。操作系统及网络设备大多已内置防火墙功能。较新的防火墙能利用封包的多样属性来进行过滤，例如：来源 IP地址、来源端口号、目的 IP 地址或端口号、服务类型（如 HTTP 或是 FTP）。也能经由通信协议、TTL 值、来源的网域名称或网段...等属性来进行过滤。
* 
* 应用层防火墙（七层）：应用层防火墙是在 TCP/IP 堆栈的“应用层”上运作，您使用浏览器时所产生的数据流或是使用 FTP 时的数据流都是属于这一层。应用层防火墙可以拦截进出某应用程序的所有封包，并且封锁其他的封包（通常是直接将封包丢弃）。理论上，这一类的防火墙可以完全阻绝外部的数据流进到受保护的机器里。

按照防火墙的位置，可以分为：

* 主防火墙：如上面图1所示，主防火墙位于网络边界，控制进出网络的网络包。Neutron FWaas 是一种主防火墙，安装在网络节点上。
* 分布式防火墙：分布式防火墙位于内部网络节点上，比如终端计算机上，控制进出该计算机的网络包。Neutron 安全组（Security Group）是一种分布式防火墙，安装在计算节点上。

主防火墙的特点：

* 内部网络和外部网络之间的所有网络数据流都必须经过防火墙。防火墙适用于用户网络系统的边界，属于用户网络边界的安全保护设备。
* 只有符合安全策略的数据流才能通过防火墙。防火墙是一个类似于桥接或路由器的、多端口的（网络接口>=2）转发设备，它跨接于多个分离的物理网段之间，并在报文转发过程之中完成对报文的审查工作。
* 应用层防火墙具备更细致的防护能力。下一代防火墙具备应用层分析的能力，能够基于不同的应用特征，实现应用层的攻击过滤，在具备传统防火墙、IPS、防毒等功能的同时，还能够对用户和内容进行识别管理，兼具了应用层的高性能和智能联动两大特性，能够更好的针对应用层攻击进行防护。

防火墙的主要概念：

* 规则：只有符合所制定的规则的网络包才能通过防火墙。
* 策略：规则的逻辑集合。

（[资料来源](http://baike.baidu.com/link?url=04B17eMudf1N1OlJYkzpojBxmsLcUDQ4UBbWONi027Wxug6zaGyIgdDBjl547XTohEkyq4bUrCV566jD9nVgBYxjz3pYCyMNeyfsvnKvkmy)）

**1.2 IP 地址欺骗 （IP Spoofing）**

    正常情况下，二层数据帧的源IP 地址就是发出数据的机器的 IP 地址，对方计算机接收到以后，向该 IP 地址发出回复数据帧：

（该例子中，node1 发往 node2 的帧的 IP 地址就是 node1 的地址，因此 node2 能够通过  ARP 获取 node1 的 MAC，并将回复将发回到它）

如果源计算机的数据帧的源 IP 地址不是它自己的IP地址而是一个不存在的地址或者另外一台机器的地址，目的计算机接受到数据帧后，它就会一直不停的发出 ARP 广播，最终也无法获取到MAC地址，或者发送返回帧到另一台的计算机。这就是所谓的（源） IP 地址欺骗。

（该例子中，node1 将它发往 node2 的数据帧的源IP设置为 node3 的IP，导致 node2 的回复发到了node3）

如果大量的计算机使用另外一台计算机的 IP 作为源 IP 向很多的计算机发出 ping 命令，那么那一台计算机将会收到很多的 ping 回复。这将导致它的网络带宽被塞满而不能对外提供网络服务。

（[以上资料来源](https://sandilands.info/sgordon/address-spoofing-with-iptables-in-linux)）

常见的防止IP地址欺骗的方法：

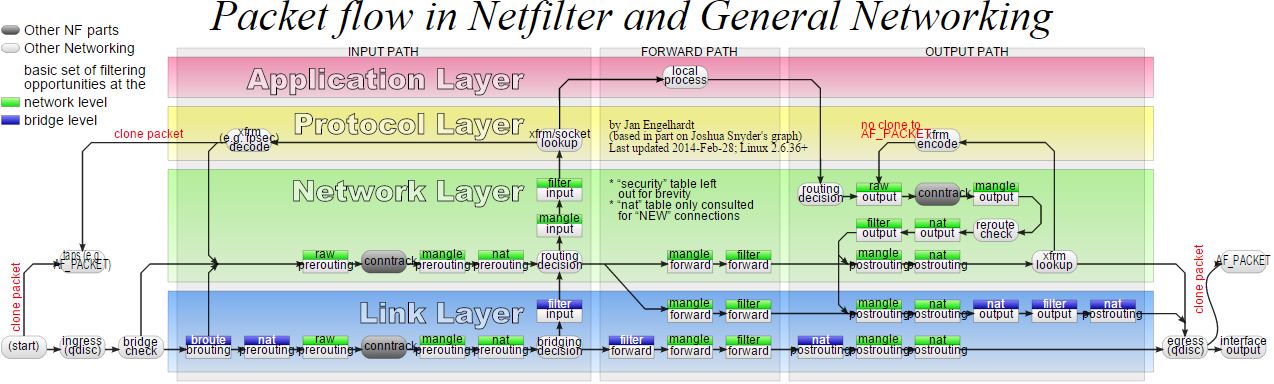
* 任何进入网络的数据包不能把网络内部的地址作为源地址。
* 任何进入网络的数据包必须把网络内部的地址作为目的地址。
* 任何离开网络的数据包必须把网络内部的地址作为源地址。
* 任何离开网络的数据包不能把网络内部的地址作为目的地址。
* 任何进入或离开网络的数据包不能把一个私有地址(Private Address)或在RFC1918中列出的属于保留空间(包括10.x.x.x/8、172.16.x.x/12 或192.168.x.x/16 和网络回送地址127.0.0.0/8.)的地址作为源或目的地址。
* 阻塞任意源路由包或任何设置了IP选项的包。

Neutron 安全组针对 IP 地址欺骗，特别设置了相应的防火墙规则。

**1.3 iptables bridge （在 linux 桥上做防火墙）**

首先说说 bridge。[这篇文章](http://www.cnblogs.com/zmkeil/archive/2013/04/21/3034733.html) 详细解释了 bridge 的工作原理。简单地说，brige 是 linux 上的一个虚拟（逻辑）设备，它依赖于从设备，并将从设备虚拟化为端口（port），从设备的 IP 及 MAC 都不再可用，且被设置为接收任何包，最终由 bridge 设备来决定数据包的去向：接收到本机、转发、丢弃。

[上一篇文章](http://www.cnblogs.com/sammyliu/p/4636091.html) 介绍了 iptables 是如何处理三层网络包的。显然，普通的 Linux 内核 IP 协议栈代码和 netfilter 无法处理经过网桥的数据包，因为网桥是属于网络二层，对于 IP 栈来说是透明的。因此，Linux 内核和 iptables 增加了处理经过二层网桥的数据包的功能。下面是增加了网桥处理能力的 netfilter 的数据包处理过程：

****

（引用自 <https://upload.wikimedia.org/wikipedia/commons/3/37/Netfilter-packet-flow.svg>）

    linux-bridge/netfilter 在内核（代码模块成为 br-nf，代码 [在这里](http://lxr.free-electrons.com/source/net/bridge/br_netfilter.c)）被启用后，iptables 链的 Hook 函数也被注册到桥处理代码（bridging code）中。然而，这并不意味着它们不再被注册到标准的 IP协议栈代码中。对于被桥代码处理的网络包来说，br-nf 会判断 iptables 哪些链会在网络栈的什么位置被遍历。显然，它会保证同一个包不会被链处理两次，因此所有没有被桥代码处理的包会被标准的IP包处理方式来处理。但是，如果一个计算机既有 bridge 也有 router，那么进入该计算机的网络包，有些会被 bridge code 中 netfiler Hook 方法处理，有些会被 IP code 中的 netfilter hook 方法处理，这点需要在定义 iptables rules 时加以注意。

   再来谈谈关键的 ”Bridging Decision“ 部分，其代码在（<http://lxr.free-electrons.com/source/net/bridge/br_input.c#L226> rx\_handler\_result\_t br\_handle\_frame(struct sk\_buff \*\*pskb)）。它会基于目的MAC地址做出如下判断：

1. bridge 网络帧，如果帧的目的 MAC 地址是在桥的另一侧的某个网络设备上。
2. 泛洪该网络帧，如果帧的目的 MAC 对网桥是不认识的。
3. 转到更高一层的三层IP协议栈代码处理，如果帧的目的地址是桥本身的或者它的某个端口的。
4. 忽略它，如果帧的目的地址是位于它来自的桥的方向的同一侧。

    对 （1）和 （2）来说，接下来该数据帧被 filter 的 FORWARD 链处理。对（3）来说，它会被 filter 的 INPUT 链处理。发生这种情况时，该 bridge 其实是被用作一个路由器（一个对比例子是现实世界中的带路由的交换机设备）。包含 IP 包的以太帧的目的 MAC 地址是桥的MAC地址，但是目的 IP 地址不是桥的IP地址。  这篇文章 [ebtables/iptables interaction on a Linux-based bridge](http://ebtables.netfilter.org/br_fw_ia/br_fw_ia.html) 详细分析了其处理过程。

    要在 Linux bridge 上做防火墙（firewall packets as they are being bridged，或者说在 linux bridge 上应用 iptables 规则），必须满足以下条件：

* Linux 内核支持 bridge   (CONFIG\_BRIDGE=m or CONFIG\_BRIDGE=y).
* Linux 内核支持 bridge/netfilter (CONFIG\_BRIDGE\_NETFILTER=y).
* Linux 内核需要包含 Netfilter physdev match 支持 (CONFIG\_IP\_NF\_MATCH\_PHYSDEV=m or CONFIG\_IP\_NF\_MATCH\_PHYSDEV=y). 它在 2.6 版本的内核中开始作为标准模块。
* 你的 iptables 工具需要支持 physdev match，以及支持在一条规则中使用多个 '-m physdev'. iptables 从 1.3.6 版本开始支持。
* 你需要安装 bridge-utils 包。

    需要注意的，各个 Linux 发行版并不是都默认启用了 netfilter for bridge，比如 Ubuntu 默认启用了， 而 RedHat 默认没启用。Neutron 会执行下面的命令来启用它：

sysctl -w net.bridge.bridge-nf-call-arptables=1

sysctl -w net.bridge.bridge-nf-call-iptables=1

    从Linux 2.6 内核版本上，iptables 1.3.6 版本以后就能够过滤 Linux bridge 上的网络包了。它使用如下参数：

|  |  |
| --- | --- |
| 参数 | 说明 |
| **-m physdev** | 匹配桥端口的进入和出去的设备。这只在内核版本 2.5.44 以上有效**（**This module matches on the bridge port input and output devices enslaved to a bridge device. This module is a part of the infrastructure that enables a transparent bridging IP firewall and is only useful for kernel versions above version 2.5.44.） |
| **--physdev-in [!] name** | Name of a bridge port via which a packet is received。（收到数据包的 bridge port 的名字，只对 INPUT, FORWARD and PREROUTING 链有效）。可以添加 ”+“ 来做头部部分匹配。如果数据包不是从桥设备接收到的，则匹配不成功。 |
| **--physdev-out [!] name** | Name of a bridge port via which a packet is going to be sent.（数据包要发到的 bridge port name，只对 FORWARD, OUTPUT and POSTROUTING 有效） |
| **[!] --physdev-is-in** | Matches if the packet has entered through a bridge interface （在数据包是通过一个桥interface进入的时候会匹配成功） |
| **[!] --physdev-is-out** | Matches if the packet will leave through a bridge interface. （在数据包将要通过一个桥的interface出去时匹配成功） |
| **[!] --physdev-is-bridged** | Matches if the packet is being bridged and therefore is not being routed （在数据包是正被桥接的而不是正被路由时匹配成功。它只对 FORWARD and POSTROUTING 有效。） |

举例（[来源](http://bwachter.lart.info/linux/bridges.html)）：

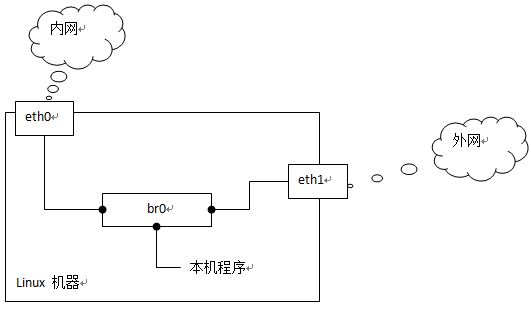
在一台 linux 机器上创建 linux bridge br0，连接 eth0 和 eth1.其中 eth0 是内网端口，eth1和 eth2是外网端口。

# brctl addbr br0

# brctl addif br0 eth0

# brctl addif br0 eth1

# ifconfig br0 netmask 255.255.255.0 192.168.32.1 up



实际上，在将 eth1 的 IP 设置给 br0 以后，该机器可以通过 br0/eth1 访问外网（[这篇文章](http://www.ibm.com/developerworks/cn/linux/1310_xiawc_networkdevice/index.html) 详细地阐述了该原理），这样，该 linux 机器是同时作为一台路由器和一台终端机：

* 作为内网和外网之间的路由器，这些数据包经过 br0 时会被 iptables 的 filter 表的 FORWARD 链处理；
* 作为可以访问外网的终端机，外网的数据包通过 eth1 直接进出该计算机，进来的包会被 iptables 的 filter 表的 INPUT 链过滤，出去的包被 OUTPUT 链过滤。

可以创建如下的 iptables 规则来实现经过 br0 的网络包：

[复制代码](javascript:void(0);)

#允许本机通过访问外网，但是将进来的 udp，tcp 和 icmp 的网络包写日志（INPUT 规则的 physdev-in 肯定是 eth1 了）

iptables -A INPUT -p udp -m physdev --physdev-in eth1 -j LOG

iptables -A INPUT -p tcp -m physdev --physdev-in eth1 -j LOG

iptables -A INPUT -p icmp -m physdev --physdev-in eth1 -j LOG

# 允许 ssh, smtp and http 到 br0(INPUT!)

iptables -A INPUT -p tcp --dport 22 -m physdev --physdev-in eth1 -j ACCEPT

iptables -A INPUT -p tcp --dport 25 -m physdev --physdev-in eth1 -j ACCEPT

iptables -A INPUT -p tcp --dport 80 -m physdev --physdev-in eth1 -j ACCEPT

# 拒绝到 br0 的别的网络包

iptables -A INPUT -p tcp --syn -m physdev --physdev-in eth1 -J REJECT

# 允许通过 tcp 端口 22（ssh），25 （smtp），80（http）到内网

iptables -A FORWARD -p tcp --dport 22 -m physdev --physdev-in eth1 --physdev-out eth0 -j ACCEPT

iptables -A FORWARD -p tcp --dport 25 -m physdev --physdev-in eth1 --physdev-out eth0 -j ACCEPT

iptables -A FORWARD -p tcp --dport 80 -m physdev --physdev-in eth1 --physdev-out eth0 -j ACCEPT

# 禁止 tcp 端口 6667 （IRC）

iptables -A FORWARD -p tcp --dport 6667 -m physdev --physdev-in eth1 -j REJECT

# 禁止其它连接到内网

iptables -A FORWARD -p tcp --syn -m physdev --physdev-in eth1 --physdev-out eth0 -j REJECT

[复制代码](javascript:void(0);)

**1.4 ipset 和在 iptables 规则中使用 ipset**

  在 iptables 中，如果去匹配多个 IP 地址的话，就会写入多条 iptables 的规则（这些 IP 都是无规律性的），当如果需要匹配几百甚至上千个 IP 地址的话，那么性能就会受到严重的影响。使用 ipset 的话，这种情况可以有很大的改善，其最主要是的在结构和规则的查找上面做了很大的改善。当出现上面的情况的时候，ipset 对性能就始终稳定在一个相对值上。根据提供的测试结果表明，当规则在300-1500之间的时候其对性能的影响基本是水平线。所以当你的防火墙规则过多的时候不妨试试看。

安装：[http://ipset.netfilter.org/](http://ipset.netfilter.org/%20)下载然后安装。

使用：  
(1). 首先 ipset 里面好多的命令是和 iptables 一样的，比如 -F ，-X， -A， -nL等等。  
(2). 用户如果什么都没有添加的话，这个时候 ipset list 就会发现都是空的。  
(3) 这个时候我们试着添加一个 set，如：ipset -N test\_policy ipmap --network 192.168.100.1/24。该命令：

* test\_policy：自定义set；
* ipmap：自定义set的类型，表示是 IP 地址； bitmap:ip,mac 表示是 IP 或者 MAC 地址。
* network 192.168.100.1/24：代表一个网段

(4) 自定义 set 创建好了后就需要在上面添加一些IP了，如：

* ipset -A test\_policy 192.168.100.1
* ipset -A test\_policy 192.168.100.2
* ipset -A test\_policy 192.168.100.3
* ipset -A test\_policy 192.168.100.4

（5）这个时候你ipset -nL就会看到该 set 以及它的 members：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ipset list

Name: test\_policy

Type: bitmap:ip

Revision: 2

Header: range 192.168.100.0-192.168.100.255

Size in memory: 160

References: 0

Members:

192.168.100.1

192.168.100.2

192.168.100.3

192.168.100.4

[复制代码](javascript:void(0);)

（6）把它加到 iptables 链里面，比如说加到 FORWARD 链里面。比如要拒绝该 ipset 中计算机的访问，则添加规则 iptables -A FORWARD -m set --match-set test\_policy src -j DROP。其中：

* src：也就是只是匹配的源地址，如果你需要匹配目的地址的话那么就写成 dst

 以上部分资料来自 [来源](http://bbs.chinaunix.net/thread-2064657-1-1.html)。更详细的信息，请 man ipset。

**2. Neutron 防火墙和安全组概述**

    为了虚机和网络安全，Neutron 提供安全组和 FWaas（firewall-as-a-service）。两者的概念非常类似。一个安全组定义了哪些进入的网络流量能被转发给虚机。安全组包含一组防火墙策略，称为安全组规则（Security Group Rule）。可以定义n个安全组，每个安全组可以有n个规则，可以给每个实例绑定n个安全组。FWaas 则作用于虚拟路由器上，对进出租户网络的网络流量进行过滤。OpenStack 的相关的组件为：

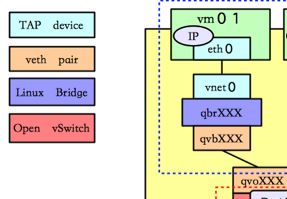
1. Nova Security Group：Nova 项目提供给虚机的安全组。
2. Neutron Security Group：Neutorn 项目提供给虚机的安全组。
3. Neutron FWaas：基于 Neutron L3 Agent 实现的虚机防火墙的一个参考实现。

这三个组件底层都通过控制 Linux iptables 来控制进出虚机或者租户网络的网络包，但是在不同的位置使用不同的方法来实现不同的目的。三者之间的比较如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | Nova Security Group | Neutron Security Group | Neutron FWaas |
| 作用 | 控制进入虚机的网络包 | 控制进出虚机的网络包 | 控制进出租户网络的网络包。对不经过 Virutal Router 的网络包不起作用。 |
| 防火墙节点 | Nova 计算节点 | Nova 计算节点（L2 Agent 节点） | L3 Agent 节点上该 firewall 所在的租户的所有 router 的 namespace 中 |
| IPV6 支持 | 支持。通过配置项 use\_ipv6 来指定是否使用 IPV6，默认值为 false。 | 判断 /proc/sys/net/ipv6/conf/default/disable\_ipv6 文件的内容。如果是 ”0“，表示支持 ipv6，否则不支持。如果支持的话，则同时往 iptables 和 ip6tables 中添加链和规则。 | 支持，同时往 iptables 和 ip6tables 中添加链和规则，但是似乎没有判断系统是否支持 IPV6. |
| 控制粒度 | 虚机 | Neutron port | 租户的所有 virtual router 的所有 Interface |
| 控制实施者 | Nova 计算节点的 IP 栈 | Nova 计算节点的 qbr 桥 | router 的 namespace 的 IP 栈 |
| 控制的网络流量方向 | ingress only （进入虚机的网络包） | ingress 和 egress （进出虚机的网络包） | ingress 和 egress （进出租户网络的网络包） |
| 匹配的数据项 | 协议、源 IP 网段、目的端口号 | 协议、端口、网段、方向 | 协议、源端口、目的端口、源 IP 网段、目的 IP、方向 |
| 匹配的结果行为 | 允许（allow） | 允许（allow） | 允许（allow）和拒绝（deny） |
| 使用 | 启动虚机时指定安全组，不指定的话使用默认安全组。虚机启动后，对所使用的安全组的变更会实时应用到虚机。可以向虚机添加和删除安全组。 | 动态（随时，启动虚机时和启动后）使用 neutron port-update 命令更新指定 port 的安全组。 | 动态（随时）应用到租户的所有虚拟路由器。 |
| 没有显式配置安全组时的默认行为 | 使用默认安全组（Default Security Group  ），它允许使用同一个安全组的虚机访问该虚机 | 禁止该机器的网络访问，除了允许它访问 DHCP 服务 | 没有默认防火墙，防火墙没有默认规则。配置了 firewall 但是没添加规则的话，禁止所有网络进出数据网络。 |
| 允许的数目 | 多个，Security Group 数目（默认10）和 Security Group Rules 数目（默认100）受租户的 quota 限制 | | 最多一个，不允许多个。 |
| CLI | #列表，增加，删除 虚机的安全组  add-secgroup: Add a Security Group to a server               list-secgroup: List Security Group(s) of a server.              remove-secgroup: Remove a Security Group from a server.  #操作安全组  secgroup-add-group-rule    secgroup-add-rule          secgroup-create            secgroup-delete            secgroup-delete-group-rule secgroup-delete-rule       secgroup-list              secgroup-list-rules        secgroup-update  #启动虚机时指定安全组  nova boot --flavor FLAVOR\_ID --image IMAGE\_ID **--security-groups** SEC\_GROUP | #操作安全组  neutron security-group-create  neutron security-group-list neutron security-group-rule-create  neutron security-group-rule-list neutron security-group-rule-delete  neutron security-group-delete  #创建 port 时指定安全组： neutron port-create --security-group SECURITY\_GROUP\_ID1 --security-group SECURITY\_GROUP\_ID2 NETWORK\_ID  #显示 port 的安全组 neutron port-list -c security\_groups -c id -c mac\_address -c fixed\_ips -c device\_id  #删除 port 的安全组 neutron port-update --no-security-groups PORT\_ID  #使用带安全组的 port 启动虚机 nova boot vm1 --flavor 6 --nic **port-id=** | #防火墙操作  firewall-create  firewall-delete  firewall-list  firewall-show  firewall-update  #防火墙策略操作 firewall-policy-create  firewall-policy-delete  firewall-policy-insert-rule firewall-policy-list  firewall-policy-remove-rule firewall-policy-show  firewall-policy-update  #防火墙规则操作 firewall-rule-create  firewall-rule-delete  firewall-rule-list  firewall-rule-show  firewall-rule-update |
| 配置 | 见下文 | | |
| 底层实现 | 见下文 | | |

**3. Neutron 安全组**

    我们知道，每个网桥（bridge）都有若干个端口（port）连接到它。从一个 port 进来的数据包都根据其目的 MAC 地址会被转发到其他端口，或者被丢弃。同样地，nova-compute 节点（计算节点）也充当两个角色：数据网络的转发器（bridge）（它使用 qbr bridge 将外部访问虚机的流量转发到虚机。当然了，在进入 qbr 桥之前，OVS 需要对 traffic 做一些操作）和 终端机（这个节点往往会配置多块网卡，一块用于数据网络，一块用于管理网络，这样，别的计算机通过管理网络来访问该计算机，比如配置 nova-compute 等）。跟上面例子区别的是，qbr 网桥并没有设置 IP 地址，因此需要别的网卡。而 qbr 桥是一个简单的网桥，它一头连接的是虚机网卡 eth0 的 tap 设备（比如 tap59cfa0b8-2f），另一头连接 veth pari 的一端（比如qvb59cfa0b8-2f），该 veth 设备的另一端是 OVS 上的端口 qvo59cfa0b8-2f。可见，qbr bridge 连接了虚机的网卡和 OVS 桥。使用 qbr 后，进入 qvo 的网络包都将被 qbr 转发到 tap 设备，而离开 tap 设备的网络包都将被转发到 qvo 上。 Neutron 安全组就是作用在 qbr 桥上。



    至于为什么不将虚机的网卡直接连接到 OVS 桥 br-int，官方说法是：“理想地，TAP 设备最好能直接挂在 br-int 上。不幸的是，因为 OpenStack Security Group 的实现方式，这种挂载是不可能实现的。OpenStack 使用 TAP 设备上的 iptables 规则来实现 Security Group，而 open vswitch 不支持在直接连到其网桥上的 TAP 设备上使用 iptables。因此，不得不增加 Linux bridge qbr 来将 TAP 设备连到 OVS bridge 上。”。简单地说，OpenStack 需要在 qbr 桥上使用 iptables 过滤进出虚机的数据包，而 br-int 上无法做到这一点。

**3.1 配置**

|  |  |  |  |
| --- | --- | --- | --- |
| 节点 | 配置文件 | 配置项 | 说明 |
| **controller** | /etc/nova/nova.conf | security\_group\_api = neutron | 使得 nova secgroup\* 命令使用的是 neutron 安全组的 API |
| /etc/neutron/plugins/ml2/ml2\_conf.ini | enable\_security\_group = True firewall\_driver = neutron.agent.linux.iptables\_firewall.OVSHybridIptablesFirewallDriver | 启用 Neutron 安全组并指定驱动 |
| **nova-compute** | /etc/nova/nova.conf | firewall\_driver = nova.virt.firewall.NoopFirewallDriver | 禁用 nova 安全组 |
| /etc/neutron/plugins/ml2/ml2\_conf.ini | enable\_security\_group = True enable\_ipset = True firewall\_driver = neutron.agent.linux.iptables\_firewall.OVSHybridIptablesFirewallDriver | 启用 Neutron 安全组并指定驱动 |
| **network** | /etc/neutron/plugins/ml2/ml2\_conf.ini | enable\_security\_group = True enable\_ipset = True firewall\_driver = neutron.agent.linux.iptables\_firewall.OVSHybridIptablesFirewallDriver | 启用 Neutron 安全组并指定驱动 |

neutron 也提供两种安全组的实现：IptablesFirewallDriver 和 OVSHybridIptablesFirewallDriver

* neutron.agent.linux.iptables\_firewall.IptablesFirewallDriver : iptables-based FirewallDriver implementation
* neutron.agent.linux.iptables\_firewall.OVSHybridIptablesFirewallDriver: subclass of IptablesFirewallDriver with additional bridge

默认值是 neutron.agent.firewall.NoopFirewallDriver，表示不使用 neutron security group。

**3.2 CLI 示例**

列表 安全组：

**s1@controller:~$ neutron security-group-list**  
+--------------------------------------+------------+-------------+  
| id                                   | name       | description |  
+--------------------------------------+------------+-------------+  
| 0aff643b-7b34-4384-b482-f37eccef5b90 | sg-all-ssh | allow ssh   |  
| c98a2a3f-1b06-429d-b419-aa862662f116 | default    | default     |  
| f5377a66-803d-481b-b4c3-a6631e8ab456 | default    | default     |  
+--------------------------------------+------------+-------------+

列表配置了某安全组的所有 port：

**s1@controller:~$ neutron port-list -c security\_groups -c id -c mac\_address -c fixed\_ips -c device\_id | grep c98a2a3f-1b06-429d-b419-aa862662f116**  
| [u'c98a2a3f-1b06-429d-b419-aa862662f116'] | 29eaea66-6614-47e3-b251-0104ff9f12e2 | fa:16:3e:bc:52:eb | {"subnet\_id": "d803eaa3-63c9-4c2b-834a-9780db31df70", "ip\_address": "100.1.250.1"}   | 1214b1b4-7c3a-41ad-ad8e-1e520b9c2830                                          |  
| [u'c98a2a3f-1b06-429d-b419-aa862662f116'] | a7e26c3e-c1ae-4c65-a0e4-2d81266a11b3 | fa:16:3e:b8:f9:ad | {"subnet\_id": "4ac56c61-84f3-4d00-b87a-1ab2441e8437", "ip\_address": "100.1.150.1"}   | 1214b1b4-7c3a-41ad-ad8e-1e520b9c2830

**s1@controller:~$ neutron port-list -c security\_groups -c id -c mac\_address -c fixed\_ips -c device\_id | grep 29eaea66-6614-47e3-b251-0104ff9f12e2**

| [u'0aff643b-7b34-4384-b482-f37eccef5b90'] | 29eaea66-6614-47e3-b251-0104ff9f12e2 | fa:16:3e:bc:52:eb | {"subnet\_id": "d803eaa3-63c9-4c2b-834a-9780db31df70", "ip\_address": "100.1.250.1"}   | 1214b1b4-7c3a-41ad-ad8e-1e520b9c2830                                          |

删除指定 port 的安全组：

**s1@controller:~$ neutron port-update --no-security-groups 29eaea66-6614-47e3-b251-0104ff9f12e2**  
Updated port: 29eaea66-6614-47e3-b251-0104ff9f12e2

删除后查看：

**s1@controller:~$ neutron port-list -c security\_groups -c id -c mac\_address -c fixed\_ips -c device\_id | grep 29eaea66-6614-47e3-b251-0104ff9f12e2**  
| []                                        | 29eaea66-6614-47e3-b251-0104ff9f12e2 | fa:16:3e:bc:52:eb | {"subnet\_id": "d803eaa3-63c9-4c2b-834a-9780db31df70", "ip\_address": "100.1.250.1"}   | 1214b1b4-7c3a-41ad-ad8e-1e520b9c2830                                          |

设置指定 port 的安全组：

**s1@controller:~$ neutron port-update --security-group 0aff643b-7b34-4384-b482-f37eccef5b90 29eaea66-6614-47e3-b251-0104ff9f12e2**

Updated port: 29eaea66-6614-47e3-b251-0104ff9f12e2

设置后查看

**s1@controller:~$ neutron port-list -c security\_groups -c id -c mac\_address -c fixed\_ips -c device\_id | grep 29eaea66-6614-47e3-b251-0104ff9f12e2**  
| [u'0aff643b-7b34-4384-b482-f37eccef5b90'] | 29eaea66-6614-47e3-b251-0104ff9f12e2 | fa:16:3e:bc:52:eb | {"subnet\_id": "d803eaa3-63c9-4c2b-834a-9780db31df70", "ip\_address": "100.1.250.1"}   | 1214b1b4-7c3a-41ad-ad8e-1e520b9c2830

**3.3 Neutron 安全组驱动**

    如上所述，Neutron 安全组提供两种驱动，OVSHybridIptablesFirewallDriver 和 IptablesFirewallDriver。不幸的是，在 Juno 版本上使用 IptablesFirewallDriver 遇到下面的 bug 而无法使用：

Command: ['sudo', '/usr/bin/neutron-rootwrap', '/etc/neutron/rootwrap.conf', 'iptables-restore', '-c']

Exit code: 2

Stdout: ''

Stderr: "iptables-restore v1.4.21: interface name `406e066f-5d2a-4df9-b763-2bff230738cb' must be shorter than IFNAMSIZ (15)\nError occurred at line: 96\nTry `iptables-restore -h' or 'iptables-restore --help' for more information.\n"

2015-07-13 23:30:54.312 9389 ERROR neutron.agent.linux.iptables\_manager [req-63695db6-6bdf-4ec0-8186-55218a076ab5 None] IPTablesManager.apply failed to apply the following set of iptables rules。

下面的分析是基于 OVSHybridIptablesFirewallDriver 的。

**3.3.1 Neuron 安全组使用的 iptables 链和规则**

    理论上，qbr 桥只负责在虚机和 br-int 之间转发网络帧，因此 bridge 代码应该是用 filter 表的 FORWARD 链来处理这些网络帧。但是，基于*还没有被找到的原因*，Neutron 还是对 filter 表的 INPUT 链进行了处理。根据上面基础知识部分的描述，只有当 qbr 承担路由器功能时（达到的网络帧的目的 MAC 是它自己但是目的 IP 地址不是它自己时）才会调用 LOCAL\_IN hook，从而走到 filter 的 INPUT 链。但是很明显，目前的 Neutron 中 qbr 只是 bridge，不承担 router 任务。[这篇文章](https://developer.rackspace.com/blog/software-defined-networks-in-the-havana-release-of-openstack-part-2/) 也认为大多数经过 qbr 进出虚机的网络包会被 FORWARD 链处理，还是有网络包会被 INPUT 或者 OUTPUT 链处理。幸运的是，INPUT 和 OUTPUT 链的规则和 FORWARD 链的规则完全相同。这一块还会继续摸索，有大牛知晓还请不吝告知。

    Neutron L2 Agent 承担使用 iptables 维护链和规则的任务。它为虚机的每块网卡的 tap 设备建立 i（进）、o（出） 和 s（防IP欺骗）链和规则，来实现：

* prepares, updates, removes firewall filters （准备、更新和删除防火墙过滤器）
* drops all packets by default （默认丢弃所有包）
* prevents IP spoofing based on port's mac address (compatible with allowed\_address\_pairs extension) （防 IP 欺骗）
* allows incoming DHCP and ICMPv6 RA （允许进入虚机的 DHCP 包和 ICMPV6 RA）
* blocks outgoing DHCP （禁止出虚机的 DHCP 包）
* drops INVALID packets （丢弃无效状态的包）
* allows stateful, established connections （允许有状态的并且已建立的连接，比如允许进来的 ICMP 的时候，从外面 ping 虚机时虚机的响应包是可以返回的）
* converts security group rules to iptables rules (IPv4, IPv6, TCP, UDP, ICMP, ICMPv6) （将用户配置的规则转化为 iptables 规则）
* multiple TCP/UDP ports per iptables rule using multiport module
* 支持 IPV4 和 IPV6

来看看 Neutron 为了实现这些功能添加的 iptables 链：

[复制代码](javascript:void(0);)

-N neutron-filter-top

-N neutron-openvswi-FORWARD #neutorn 定义的 FORWARD 链

-N neutron-openvswi-INPUT #Neutron 定义的 INPUT 链

-N neutron-openvswi-OUTPUT #Neutron 定义的 OUTPUT 链

-N neutron-openvswi-i6e3f2eda-3 #处理进入该虚机的网络包

-N neutron-openvswi-local

-N neutron-openvswi-o6e3f2eda-3 #处理出该虚机的网络包

-N neutron-openvswi-s6e3f2eda-3 #处理出该虚机的网络包的防IP欺骗

-N neutron-openvswi-sg-chain

-N neutron-openvswi-sg-fallback

-A INPUT -j neutron-openvswi-INPUT #将 INPUT 链转到 neutron 的 INPUT 链

-A FORWARD -j neutron-filter-top

-A FORWARD -j neutron-openvswi-FORWARD #将 FORWARD 链转到 neutorn 的 forward 链

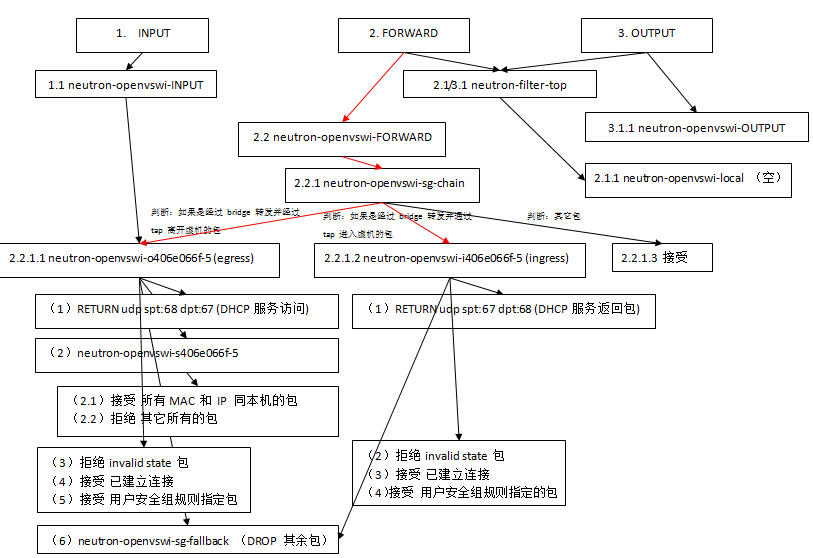
-A OUTPUT -j neutron-filter-top

-A OUTPUT -j neutron-openvswi-OUTPUT #将 OUTPUT 链转到 neutron 的 output 链

-A neutron-filter-top -j neutron-openvswi-local

[复制代码](javascript:void(0);)

这些链之间的关系：



**3.3.2 当在 port 上不应用安全组时：只允许虚机访问 DHCP 服务器和 DHCP 服务器给虚机的返回包，禁止其余所有的网络访问。**

     先补充两个iptabels 的知识点：

* -j RETURN：退出当前CHIAN，如果当前CHIAN是别的CHAIN调用的子CHIAN，那么返回到调用点下一条规则处开始执行，如果当前CHIAN不是子CHAIN，那么就以默认策略执行。
* -m state：匹配连接的状态。Linux 网络连接的状态包括：

|  |  |
| --- | --- |
| 状态 | 说明 |
| NEW | NEW 说明这个包是我们看到的第一个包。意思就是，这是 conntrack 模块看到的某个连接第一个包，它即将被匹配了。比如，我们看到一个SYN 包，是我们所留意的连接的第一个包，就要匹配它。第一个包也可能不是SYN包，但它仍会被认为是NEW状态。这样做有时会导致一些问题，但对某些情况是有非常大的帮助的。例如，在我们想恢复某条从其他的防火墙丢失的连接时，或者某个连接已经超时，但实际上并未关闭时。 |
| ESTABLISHED | ESTABLISHED 已经注意到两个方向上的数据传输，而且会继续匹配这个连接的包。处于 ESTABLISHED 状态的连接是非常容易理解的。只要发送并接到应答，连接就是ESTABLISHED的了。一个连接要从NEW变为ESTABLISHED，只需要接到应答包即可，不管这个包是发往防火墙的，还是要由防火墙转发的。ICMP的错误和重定向等信息包也被看作是ESTABLISHED，只要它们是我们所发出的信息的应答。 |
| RELATED | RELATED 是个比较麻烦的状态。当一个连接和某个已处于 ESTABLISHED 状态的连接有关系时，就被认为是 RELATED 的了。换句话说，一个连接要想是 RELATED的，首先要有一个ESTABLISHED的连接。这个ESTABLISHED连接再产生一个主连接之外的连接，这个新的连接就是 RELATED的了，当然前提是conntrack模块要能理解RELATED。ftp是个很好的例子，FTP-data 连接就是和 FTP-control 有 RELATED 的。还有其他的例子，比如，通过IRC的DCC连接。有了这个状态，ICMP应答、FTP传输、DCC 等才能穿过防火墙正常工作。注意，大部分还有一些UDP协议都依赖这个机制。这些协议是很复杂的，它们把连接信息放在数据包里，并且要求这些信息能被正确 理解。 |
| INVALID | INVALID说明数据包不能被识别属于哪个连接或没有任何状态。有几个原因可以产生这种情况，比如，内存溢出，收到不知属于哪个连接的ICMP 错误信息。一般地，我们DROP这个状态的任何东西。 |

[复制代码](javascript:void(0);)

-A neutron-openvswi-FORWARD -m physdev --physdev-out tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-sg-chain #处理进入虚机的数据包

-A neutron-openvswi-FORWARD -m physdev --physdev-in tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-sg-chain #处理出虚机的数据包

-A neutron-openvswi-INPUT -m physdev --physdev-in tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-o6e3f2eda-3 #处理出虚机的数据包  
  
#处理进虚机的包

-A neutron-openvswi-i6e3f2eda-3 -m state --state INVALID -j DROP #丢弃无效连接

-A neutron-openvswi-i6e3f2eda-3 -m state --state RELATED,ESTABLISHED -j RETURN #允许已建立的连接

-A neutron-openvswi-i6e3f2eda-3 -s 90.1.180.3/32 -p udp -m udp --sport 67 --dport 68 -j RETURN #允许 DHCP 服务器 91.1.180.3 的返回包

-A neutron-openvswi-i6e3f2eda-3 -s 91.1.180.2/32 -p udp -m udp --sport 67 --dport 68 -j RETURN #允许 DHCP 服务器 91.1.180.2 的返回包

-A neutron-openvswi-i6e3f2eda-3 -j neutron-openvswi-sg-fallback #其余包交给 fallback 链处理   
  
#处理出虚机的包

-A neutron-openvswi-o6e3f2eda-3 -p udp -m udp --sport 68 --dport 67 -j RETURN #允许 DHCP 访问

-A neutron-openvswi-o6e3f2eda-3 -j neutron-openvswi-s6e3f2eda-3 #防止 IP 欺骗

-A neutron-openvswi-o6e3f2eda-3 -p udp -m udp --sport 67 --dport 68 -j DROP #不允许对外提供 DHCP 服务

-A neutron-openvswi-o6e3f2eda-3 -m state --state INVALID -j DROP #禁止无效连接

-A neutron-openvswi-o6e3f2eda-3 -m state --state RELATED,ESTABLISHED -j RETURN #允许已建立的连接

-A neutron-openvswi-o6e3f2eda-3 -j neutron-openvswi-sg-fallback #其余交给 fallback 链处理  
  
#防 IP 欺骗规则：防止该虚机被利用来做 IP 欺骗攻击。

-A neutron-openvswi-s6e3f2eda-3 -s 91.1.180.14/32 -m mac --mac-source FA:16:3E:F3:1E:C0 -j RETURN #允许从虚机发出的MAC 和IP 同虚机的 MAC 和 IP 的包

-A neutron-openvswi-s6e3f2eda-3 -j DROP #禁止其余包

-A neutron-openvswi-sg-chain -m physdev --physdev-out tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-i6e3f2eda-3 #处理进入虚机的包

-A neutron-openvswi-sg-chain -m physdev --physdev-in tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-o6e3f2eda-3 #处理出虚机的包

-A neutron-openvswi-sg-chain -j ACCEPT

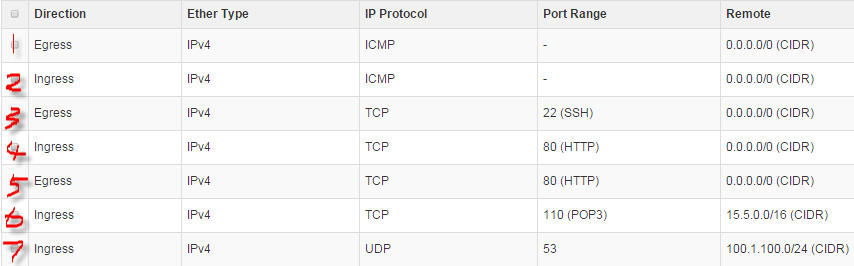
-A neutron-openvswi-sg-fallback -j DROP #丢弃其它包

[复制代码](javascript:void(0);)

可见，此时，虚机的网络访问能力 = 基本网络能力（DHCP访问、允许已建立的连接、防 IP 欺骗）。也就是说虚机此时是无法网络访问和被网络访问，比如 ping，ssh 都无法连接。

**3.3.3 当在 port 上应用安全组时：只允许用户规则制定的网络访问。**

（1）创建如下的安全组规则



各条规则：

1. 允许 ping 别的机器
2. 允许别的机器 ping 它
3. 允许访问别的机器 22 端口上的 tcp 服务 （ssh）
4. 允许 别的机器访问它的80 端口上的 tcp 服务 （http）
5. 允许访问别的机器的 80 端口上的 tcp 服务 （http）
6. 允许 15.5.0.0/16 网段的机器访问它 110 端口上的 TCP 服务（pop3）
7. 允许 100.1.100.0/24 网段上的机器访问它的 53 端口上的 UDP 服务 （DNS 服务器）

（2）将安全组绑定到 port

s1@controller:~$ neutron port-update --security-group 0aff643b-7b34-4384-b482-f37eccef5b90 6e3f2eda-3230-45e8-9be5-f382af5b83ea

Updated port: 6e3f2eda-3230-45e8-9be5-f382af5b83ea

（3）看看此时的 iptables filter 表的规则

[复制代码](javascript:void(0);)

-A neutron-openvswi-FORWARD -m physdev --physdev-out tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-sg-chain

-A neutron-openvswi-FORWARD -m physdev --physdev-in tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-sg-chain

-A neutron-openvswi-INPUT -m physdev --physdev-in tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-o6e3f2eda-3  
  
#适用于进入虚机的网络包

-A neutron-openvswi-i6e3f2eda-3 -m state --state INVALID -j DROP

-A neutron-openvswi-i6e3f2eda-3 -m state --state RELATED,ESTABLISHED -j RETURN

-A neutron-openvswi-i6e3f2eda-3 -s 90.1.180.3/32 -p udp -m udp --sport 67 --dport 68 -j RETURN #DHCP server 1

-A neutron-openvswi-i6e3f2eda-3 -s 91.1.180.2/32 -p udp -m udp --sport 67 --dport 68 -j RETURN #DHCP Server 2

-A neutron-openvswi-i6e3f2eda-3 -s 15.5.0.0/16 -p tcp -m tcp --dport 110 -j RETURN #第 6 条自定义rule

-A neutron-openvswi-i6e3f2eda-3 -p tcp -m tcp --dport 80 -j RETURN #第 4 条自定义 rule

-A neutron-openvswi-i6e3f2eda-3 -s 100.1.100.0/24 -p udp -m udp --dport 53 -j RETURN #第 7 条自定义 rule

-A neutron-openvswi-i6e3f2eda-3 -p icmp -j RETURN #第 2 条自定义 rule

-A neutron-openvswi-i6e3f2eda-3 -j neutron-openvswi-sg-fallback #处理不满足以上 rule 的包，默认是丢弃  
  
#处理出虚机的网络包

-A neutron-openvswi-o6e3f2eda-3 -p udp -m udp --sport 68 --dport 67 -j RETURN #允许访问 DHCP server

-A neutron-openvswi-o6e3f2eda-3 -j neutron-openvswi-s6e3f2eda-3 #转防 IP 欺骗链

-A neutron-openvswi-o6e3f2eda-3 -p udp -m udp --sport 67 --dport 68 -j DROP #禁止提供 DHCP 服务

-A neutron-openvswi-o6e3f2eda-3 -m state --state INVALID -j DROP #禁止无效连接

-A neutron-openvswi-o6e3f2eda-3 -m state --state RELATED,ESTABLISHED -j RETURN #允许已建立连接

-A neutron-openvswi-o6e3f2eda-3 -p tcp -m tcp --dport 22 -j RETURN #自定义第 3 条规则

-A neutron-openvswi-o6e3f2eda-3 -p icmp -j RETURN #自定义第 1 条规则

-A neutron-openvswi-o6e3f2eda-3 -p tcp -m tcp --dport 80 -j RETURN #自定义第 5 条规则

-A neutron-openvswi-o6e3f2eda-3 -j neutron-openvswi-sg-fallback #处理不满足以上规则的数据包  
  
#防 IP 欺骗规则

-A neutron-openvswi-s6e3f2eda-3 -s 91.1.180.14/32 -m mac --mac-source FA:16:3E:F3:1E:C0 -j RETURN

-A neutron-openvswi-s6e3f2eda-3 -j DROP  
  
#继续链转发

-A neutron-openvswi-sg-chain -m physdev --physdev-out tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-i6e3f2eda-3 #处理经过 qbr 进入虚机的网络包

-A neutron-openvswi-sg-chain -m physdev --physdev-in tap6e3f2eda-32 --physdev-is-bridged -j neutron-openvswi-o6e3f2eda-3 #处理经过 qbr 桥出虚机的包

-A neutron-openvswi-sg-chain -j ACCEPT #接受不经过 qbr 桥的网络包

-A neutron-openvswi-sg-fallback -j DROP #丢弃

[复制代码](javascript:void(0);)

 结论：

此时，虚机的网络访问能力 = 基本网络能力（DHCP访问、允许已建立的连接、防 IP 欺骗）+ 用户自定义的规则允许的网络能力。

**3.3.4 ipset**

根据 [官方文档](http://specs.openstack.org/openstack/neutron-specs/specs/juno/add-ipset-to-security.html) 描述，每当有新的port被创建后，L2 Agent 会增加新的 ipset set 到 iptable 链之中（在我的测试中，如果没有后面的条件，ipset 是不会被创建的）；如果该 port 所属的安全组有与别的组共享的规则，那么别的安全组的成员就会被加入到该组中。

neutron 的 security-group-rule-create CLI 支持以两种形式指定允许（被）访问的对象：

* CIDR：待被匹配的网段，使用 **remote-ip-prefix**参数指定
* Remote security group id 或者 name：表示允许所有使用该 security group 的 port 的 IP，使用 **remote-group-id** 参数。

--remote-ip-prefix REMOTE\_IP\_PREFIX          CIDR to match on.  
--remote-group-id REMOTE\_GROUP               Remote security group name or ID to apply rule.

比如下面的命令指定 remote security group 为 “default” security group：

[复制代码](javascript:void(0);)

s1@controller:~$ neutron security-group-rule-create --direction ingress --ethertype IPv4 --protocol tcp --port-range-min 22 --port-range-max 22 --remote-group-id f5377a66-803d-481b-b4c3-a6631e8ab456 0aff643b-7b34-4384-b482-f37eccef5b90

Created a new security\_group\_rule:

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

| Field | Value |

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

| direction | ingress |

| ethertype | IPv4 |

| id | 3340b429-22dc-4176-b5c0-01a4f449e812 |

| port\_range\_max | 22 |

| port\_range\_min | 22 |

| protocol | tcp |

| remote\_group\_id | f5377a66-803d-481b-b4c3-a6631e8ab456 |

| remote\_ip\_prefix | |

| security\_group\_id | 0aff643b-7b34-4384-b482-f37eccef5b90 |

| tenant\_id | 74c8ada23a3449f888d9e19b76d13aab |

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

[复制代码](javascript:void(0);)

这时候在 port 所在的计算节点上，有个新的 ipset set 被创建了。其命名规则为 protocol + remote-group-id 的一部分：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# ipset list

Name: IPv4f5377a66-803d-481b-b

Type: hash:ip

Revision: 2

Header: family inet hashsize 1024 maxelem 65536

Size in memory: 16536

References: 1

Members:

91.1.180.5

81.1.180.13

[复制代码](javascript:void(0);)

其 members 是应用了该规则的所有 port 的 IP：

s1@controller:~$ neutron port-list -c security\_groups -c id -c mac\_address -c fixed\_ips -c device\_id | grep f5377a66-803d-481b-b4c3-a6631e8ab456

| [u'f5377a66-803d-481b-b4c3-a6631e8ab456'] | 402fe6b1-7670-4b6b-84a3-097beed64015 | fa:16:3e:45:6b:8b | {"subnet\_id": "13888749-12b3-462e-9afe-c527bd0a297e", "ip\_address": "91.1.180.5"} | |

| [u'f5377a66-803d-481b-b4c3-a6631e8ab456'] | 617d20fb-a635-4a03-95b1-bc15093fe32f | fa:16:3e:d8:29:9e | {"subnet\_id": "4ec65731-35a5-4637-a59b-a9f2932099f1", "ip\_address": "81.1.180.13"} |

然后该 port 所在的计算节点上的 iptables 中增加了一项：

-A neutron-openvswi-i6e3f2eda-3 -p tcp -m tcp --dport 22 -m set --match-set IPv4f5377a66-803d-481b-b src -j RETURN

这意味着该虚机允许上述 ipset chain 的members 访问。使用 “src” 是因为那条 rule 的方向是 “ingress”。

如果不使用 ipset 的话（设置 enable\_ipset = False），iptables 规则为：

-A neutron-openvswi-i6e3f2eda-3 -s 91.1.180.5/32 -p tcp -m tcp --dport 22 -j RETURN

-A neutron-openvswi-i6e3f2eda-3 -s 81.1.180.13/32 -p tcp -m tcp --dport 22 -j RETURN

这就可以看出使用 ipset 的价值：

* 提高 neutron 操作 iptables 的性能
* 提高 linux 内核管理 iptables 规则的性能

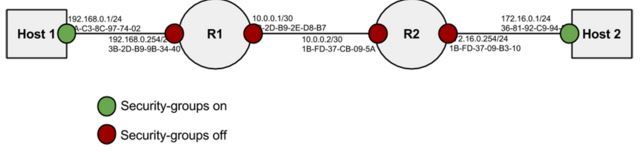
**3.4 Kilo 版本允许 port 不使用安全组 <更新于 2015/11/27>**

出发点：Kilo 版本之前，安全组是必须应用于整个network的，包括防欺骗规则，而这个在保证安全的同时也限制了在虚机上运行某些网络服务。

变化：增加 ML2PortSecurityExtensionDriver，应有该externsion dirver 后，可以指定某个 port 不使用安全组

neutron port-update c080dbeb-491e-46e2-ab7e-192e7627d050 ---port-security-enabled=False

效果图：（在作为 router 的虚机 R1 和 R1 的端口上禁止安全组）



详情可参考 <https://wiki.openstack.org/wiki/Neutron/ML2PortSecurityExtensionDriver>

**4. 示例分析**

**4.1 计算机 192.168.1.15 ping 虚机 91.1.180.14**

root@compute1:/home/s1# ping 192.168.1.104

PING 192.168.1.104 (192.168.1.104) 56(84) bytes of data.

64 bytes from 192.168.1.104: icmp\_seq=1 ttl=63 time=8.76 ms

64 bytes from 192.168.1.104: icmp\_seq=2 ttl=63 time=2.41 ms

64 bytes from 192.168.1.104: icmp\_seq=3 ttl=63 time=2.07 ms

64 bytes from 192.168.1.104: icmp\_seq=4 ttl=63 time=1.23 ms

**4.1.1 经过 qbr bridge 的网络包**

这种数据包来回共 4 次，seq 1 ~ 4。可以看到，达到 qbr 的网络包已经经过了 DNAT。

09:29:28.718256 fa:16:3e:90:e5:50 > fa:16:3e:f3:1e:c0, ethertype IPv4 (0x0800), length 98: (tos 0x0, ttl 63, id 49778, offset 0, flags [DF], proto ICMP (1), length 84)

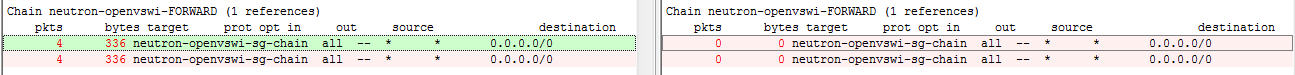
192.168.1.15 > 91.1.180.14: ICMP echo request, id 15849, seq 3, length 64

09:29:28.718670 fa:16:3e:f3:1e:c0 > fa:16:3e:90:e5:50, ethertype IPv4 (0x0800), length 98: (tos 0x0, ttl 64, id 53018, offset 0, flags [none], proto ICMP (1), length 84)

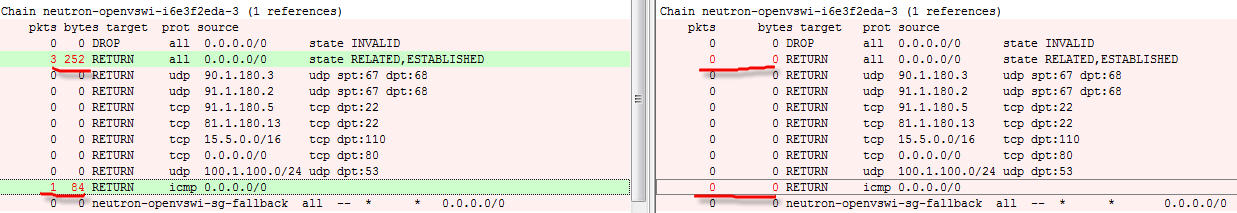
91.1.180.14 > 192.168.1.15: ICMP echo reply, id 15849, seq 3, length 64

**4.2.2 被 iptables filter 表各链处理的数据包的数目**

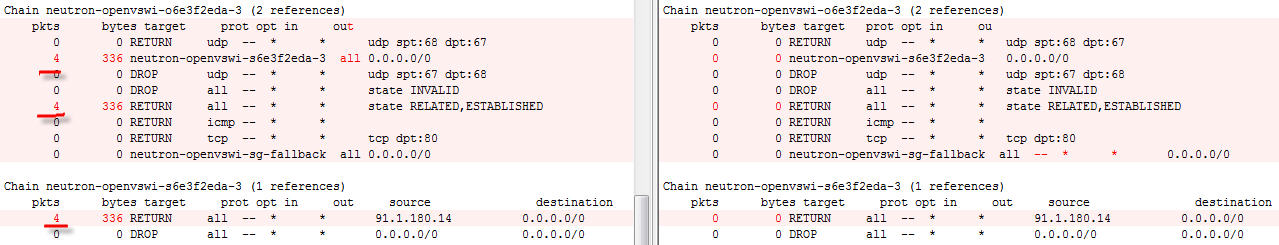
（1）被 Forward 链处理的包进出各4个



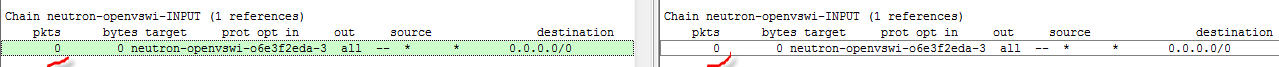
（2）被 neutron-openvswi-i6e3f2eda-3 处理的进入虚机的 4 个包：第一各包被匹配到符合要求的 icmp 协议类型，以后三个是匹配到连接状态是 ESTABLISHED。



（3）被 neutron-openvswi-o6e3f2eda-3 处理出虚机的 4 各网络包：4 各皆通过 IP Spoofing 链的处理，然后被匹配到连接状态是 ESTABLISHED （连接应该是第一次有 icmp 包进来时被建立好的）



（4）没有经过 qbr 的网络包被 INPUT 链处理



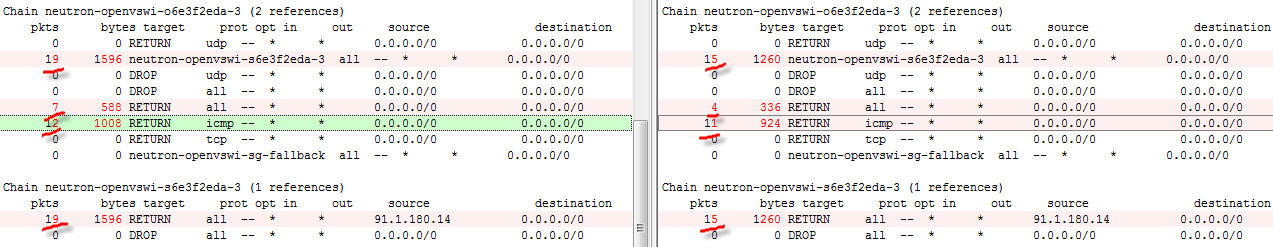
**4.2 虚机 91.1.180.14 ping 192.168.1.115**

**4.2.1 同样是有 8 个网络包被 FORWARD 链处理**

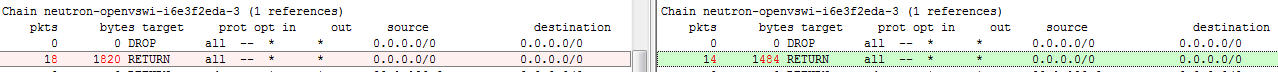


**4.2.2网络包顺序是出、 进、出、进、出、进、出、进，因此第一个包被匹配到 icmp，后面的包就是被匹配到 ESTABLISHED。**

（1）4个从虚机出去的包



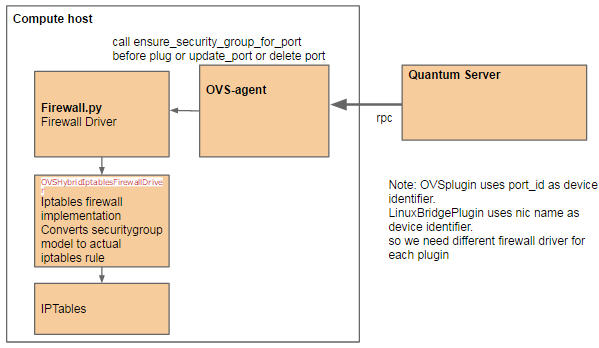
（2）4个进入到虚机的包



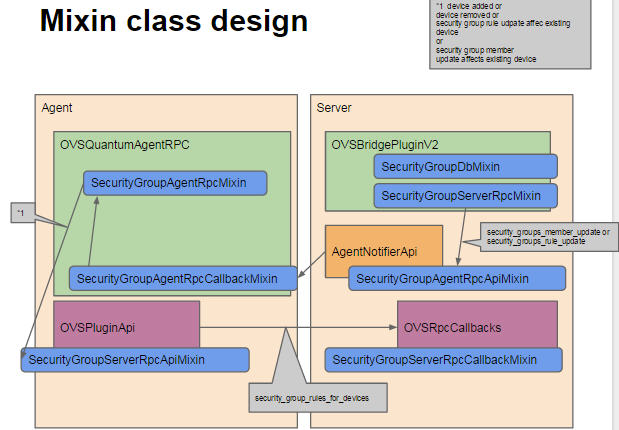
**5. 代码分析**

注：以下材料引用自 [这里](https://docs.google.com/presentation/d/1nXzNXKIfCfotdav5BzkceDiOfDypEkvtTfVXCGdq6rY/edit#slide=id.g59e93568_2_0)，与最新的实现可能有些出入，仅供参考。

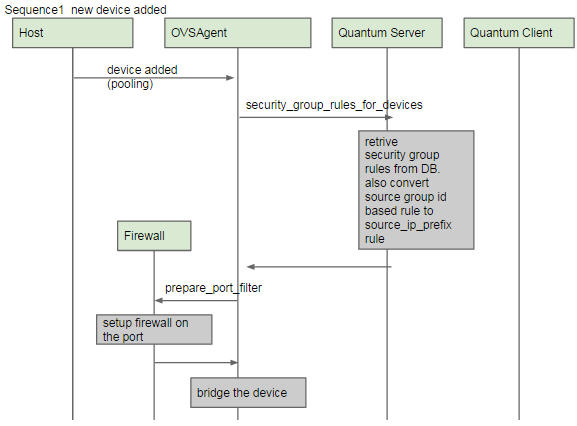
（1）neutron server 和 ML2 Agent



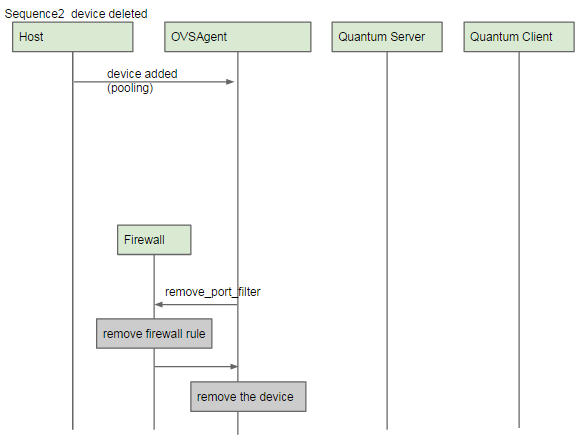
（2）neutron server 在 DB 操作后通过 RPC 调用 ML2 Agent



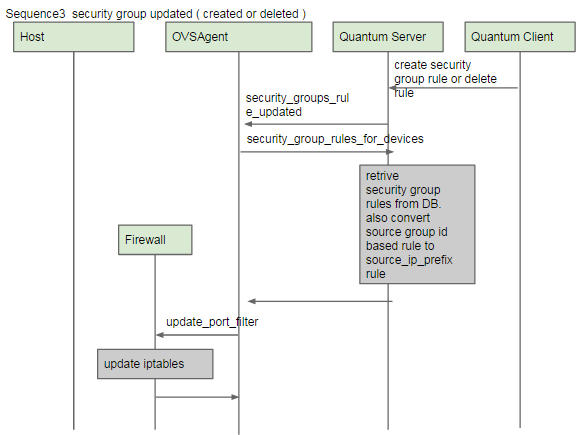
（3）当 ML2 Agent 检测到有新的虚机的 port 被 plug 到 OVS 后，它增加该 tap 设备的 iptables 链和规则



（4）当 ML2 Agent 监测到有虚机的 port 已经被从 OVS plug out 时，它删除对应的 tap 设备的 iptables 链和规则



（5）当安全组有更新的时候，Neutron Server 通过 RPC 通知 ML2 Agent 去更新 iptables rules



（6）当有安全组成员（policy）变更的时候，Neutron Server 通过 RPC 通知 ML2 Agent 去更新 iptables rules

