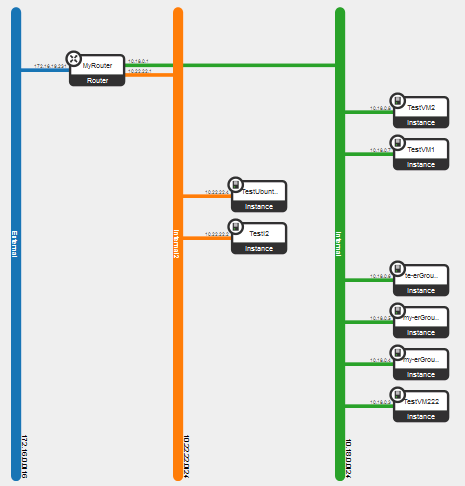
**OpenStack计算节点上虚拟网络（Neutron）详解**

场景（一个租户，两个网络，一个路由，内部网络使用GRE，Libvirt VIF Driver使用LibvirtHybridOVSBridgeDriver）：

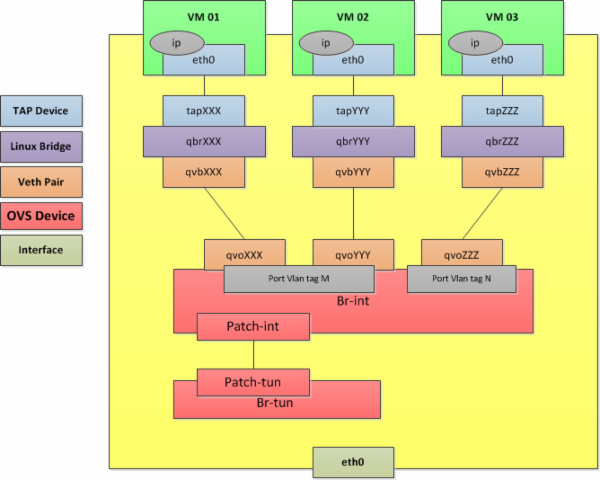


场景一虚拟网络拓扑

Figure 11 场景一虚拟网络拓扑

如图我们有一个外网（External Network），IP段为172.16.0.0/16，两个内网，分别是Internal：10.18.0.0/24，和Internal2：10.22.22.0/24，值得注意的是这是两个网络（network），而不是子网（subnet）。

在这个场景下，计算节点的内部应当是这样的：



计算节点网络连接原理

下面我将解释如何得到这幅图。首先我们看下我们的虚拟机在libvirt的名称，通过 nova show 命令我们大概可以获得像这样输出（截取前半部分）：

|  |
| --- |
| +--------------------------------------+-------------------------------  |  | Property                             | Value                                                    |  +--------------------------------------+-------------------------------  | Internal network                     | 10.18.0.3, 172.16.19.232                                 |  | OS-DCF:diskConfig                    | MANUAL                                                   |  | OS-EXT-AZ:availability\_zone          | nova                                                     |  | OS-EXT-SRV-ATTR:host                 | compute1                                                 |  | OS-EXT-SRV-ATTR:hypervisor\_hostname  | compute1                                                 |  | OS-EXT-SRV-ATTR:instance\_name        | instance-0000001e                                        | |

我们看到这台虚拟机被部署在compute1节点上，instance\_name为instance-0000001e，我们上compute1节点使用virsh dumpxml将instance-0000001e的信息打印出来（截取网络相关）：

|  |
| --- |
| <interface type='bridge'><mac address='fa:16:3e:e9:26:5a'/> <source bridge='qbr48e06cd2-60'/> <target dev='tap48e06cd2-60'/> <model type='virtio'/> <alias name='net0'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/> </interface> |

在这里我们看到这台虚拟机的网络设备是tap48e06cd2-60，而且似乎连到了qbr48e06cd2-60上，让我们用brctl show再看下（截取相关部分）：

|  |
| --- |
| qbr48e06cd2-60       8000.bed5536ff312 no     qvb48e06cd2-60tap48e06cd2-60 |

看到这里网桥qbr48e06cd2-60上接了两个接口，qvb48e06cd2-60和tap48e06cd2-60，其中的tap设备是我们虚拟机使用的虚拟网络设备，那qvb48e06cd2-60是什么？我们先用lshw –class network把所有网络设备打印出来（截取相关部分）：

|  |
| --- |
| \*-network:5description: Ethernet interface physical id: 7 logical name: qvb48e06cd2-60 serial: be:d5:53:6f:f3:12 size: 10Gbit/s capabilities: ethernet physical configuration: autonegotiation=off broadcast=yes driver=veth driverversion=1.0 duplex=full firmware=N/A link=yes multicast=yes port=twisted pair promiscuous=yes speed=10Gbit/s |

我们注意到这里显示这个设备的driver是veth，而veth总是成对出现的，我们用ethtool -S 看下这个veth的另一端连到了那里：

|  |
| --- |
| # ethtool -S qvb48e06cd2-60NIC statistics: peer\_ifindex: 16 |

OK，看下16号是哪个设备，ip link（截取相关部分）：

|  |
| --- |
| 16: qvo48e06cd2-60: <BROADCAST,MULTICAST,PROMISC,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP qlen 1000link/ether aa:c0:0f:d2:e2:43 brd ff:ff:ff:ff:ff:ff |

通过上面两个步骤我们已经知道了这对从虚拟机的网络设备到veth pair这个流程，这个过程在官方文档中针对不同的 Libvirt VIF Driver有不同的简单的描述，见    <https://wiki.openstack.org/wiki/LibvirtVIFDrivers>     。

下面应该是连到Open vSwitch上吧，让我们验证下：

# ovs-vsctl show

1910d375-2692-4214-acdf-d364382c25a4

Bridge br-int

Port br-int

Interface br-int

type: internal

Port patch-tun

Interface patch-tun

type: patch

options: {peer=patch-int}

Port "qvo48e06cd2-60"

tag: 1

Interface "qvo48e06cd2-60"

Port "qvodfdc29e2-9a"

tag: 2

Interface "qvodfdc29e2-9a"

Port "qvo18cec000-80"

tag: 2

Interface "qvo18cec000-80"

Port "qvob86d15f1-8f"

tag: 1

Interface "qvob86d15f1-8f"

Bridge br-tun

Port br-tun

Interface br-tun

type: internal

Port patch-int

Interface patch-int

type: patch

options: {peer=patch-tun}

Port "gre-1"

Interface "gre-1"

type: gre

options: {in\_key=flow, local\_ip="192.168.10.11", out\_key=flow, remote\_ip="192.168.10.10"}

ovs\_version: "1.11.0"

果然qvo48e06cd2-60是连到了br-int上， OpenStack采用这么复杂的机制，而不是把tap设备直接连到Open vSwitch上，这与安全组有关，将在3.2.4基于iptables的Security Group介绍。

在研究到OVS内部前，我们先注意下在poty “qvo48e06cd2-60”下有一个“tag: 1”，这个tag是Open vSwitch用来区分不同子网的。在这里，tag1表示我们的10.18.0.0/24子网，tag2表示10.22.22.0/24子网。

br-int和br-tun通过patch连接，在官方文档上patch的介绍并不多，但一旦两个OVS网桥通过网桥连接，这两个网桥将近乎为同一个网桥，参考资料见：    [Open vSwitch FAQ](http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=FAQ;hb=HEAD)     和    [Connecting OVS Bridges with Patch Ports](http://blog.scottlowe.org/2012/11/27/connecting-ovs-bridges-with-patch-ports/)     。

首先看下bt-int的流表规则：

|  |
| --- |
| # ovs-ofctl dump-flows br-intNXST\_FLOW reply (xid=0×4):  cookie=0×0, duration=246746.016s, table=0, n\_packets=702, n\_bytes=78521, idle\_age=1324, hard\_age=65534, priority=1 actions=NORMAL |

只有一个NORMAL的动作，在Open vSwitch的官方文档里解释为将包以传统的，非OpenFlow的方式进行交换，也就是说效果和没设置OpenFlow规则一样（见    [Open vSwitch Advanced Features Tutorial](http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=tutorial/Tutorial;hb=HEAD)     ）。那么我们分析br-tun的流表规则，首先在计算节点上用ovs-ofctl dump-ports-desc查看br-tun上所有接口：

|  |
| --- |
| OFPST\_PORT\_DESC reply (xid=0x2):1(patch-int): addr:ea:a2:71:f5:9f:ad config:     0 state:      0 speed: 0 Mbps now, 0 Mbps max 2(gre-1): addr:d6:89:b0:03:d2:72 config:     0 state:      0 speed: 0 Mbps now, 0 Mbps max LOCAL(br-tun): addr:9a:49:9a:35:d1:4e config:     0 state:      0 speed: 0 Mbps now, 0 Mbps max |

然后用ovs-ofctl dump-flows或者EasyOVS查看br-tun的流表规则（这里使用EasyOVS使排版相对好看）：

|  |
| --- |
| ID TAB PKT       PRI   MATCH                                                       ACT  0  0   339       1     in=1                                                        resubmit(,1)  1  0   285       1     in=2                                                        resubmit(,2)  2  0   3         0     \*                                                           drop  3  1   216       0     dl\_dst=00:00:00:00:00:00/01:00:00:00:00:00              resubmit(,20)  4  1   123       0     dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00              resubmit(,21)  5  10  363       1     \*                                                           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  6  2   341       1     tun\_id=0x2                                             mod\_vlan\_vid:1,resubmit(,10)  7  2   17        1     tun\_id=0x3                                             mod\_vlan\_vid:2,resubmit(,10)  8  2   3         0     \*                                                           drop  9  20  0         0     \*                                                           resubmit(,21)  10 21  3         1     vlan=2                                          strip\_vlan,set\_tunnel:0x3,output:2  11 21  16        1     vlan=1                                          strip\_vlan,set\_tunnel:0x2,output:2  12 21  4         0     \*                                                            drop  13 3   0         0     \*                                                            drop |

这里为了好看只显示了ID、表名、计数器、匹配规则和行为。先看这几条流：0、3、4、9、10、11、12，这些流定义了从br-int进入的包的行为，逐条从上往下看：

|  |
| --- |
| 0. 表0：当匹配到从port 1（patch-int）进入的包时，提交给表1继续匹配；3. 表1：当目标MAC地址为单播地址时，提交给表20继续匹配；  4. 表1：当目标MAC地址为多播/广播地址时，提交给表21继续匹配；、  9. 表20：提交给21继续匹配（这个表并非只是转发，当OVS根据表10动态建立自动学习的规则时，会添加到表20，比如下面这条流表规则是自动建立的目标MAC地址为路由的规则：“cookie = 0×0, duration = 11.099s, table = 20, n\_packets = 45, n\_bytes = 6132, hard\_timeout = 300, idle\_age = 3, hard\_age = 2, priority = 1,vlan\_tci = 0×0001/0x0fff,dl\_dst = fa:16:3e:a1:3f:19 actions = load:0 -> NXM\_OF\_VLAN\_TCI[], load:0×2 -> NXM\_NX\_TUN\_ID[], output:2”）；  10. 表21：当目标VLan标签为2时，剥去VLan标签，然后将Tunnel Key设置为3（GRE通道的Key，详见            [rfc2890](http://tools.ietf.org/html/rfc2890)             的相关描述）并从port 2（gre-1）发出去；  11. 表21：当目标VLan标签为1时，剥去VLan标签，然后将Tunnel Key设置为2并从port 2（gre-1）发出去；  12. 表21：对没成功匹配的包，丢弃。 |

再看1、6、7、5，这几个流定义了来自GRE通道（Network节点）的包的行为：

|  |
| --- |
| 1. 表0：当匹配到从port 2（gre-1）进入的包时，提交给表2继续匹配；6. 表2：当Tunnel Key为2时，添加VLan tag 1，提交给表10继续匹配；  7. 表2：当Tunnel Key为3时，添加VLan tag 2，提交给表10继续匹配；  5. 表10：首先从报文中学习VLan、MAC等信息并把规则添加表20，然后再从port 1（patch-int）发出去。 |

至此，计算节点的网络分析已经基本完成。后面到网络节点的连接等主要涉及到3层路由，暂且不表。