深入理解 OpenStack 中的网络实现

yeasy@github

v0.1: 2014-02-20

开始整体结构。

1.1 概述

1.1.1 术语

bridge：网桥，Linux 中用于表示一个能连接不同网络设备的虚拟设备，linux 中传统实现的网桥类似一个 hub 设备，而 ovs 管理的网桥一般类似交换机。

br-int：bridge-integration，综合网桥，常用于表示实现主要内部网络功能的网桥。

br-ex：bridge-external，外部网桥，通常表示负责跟外部网络通信的网桥。

GRE：General Routing Encapsulation，一种通过封装来实现隧道的方式。

VETH：虚拟 ethernet 接口，通常以 pair 的方式出现，一端发出的网包，会被另一端接收，可以形成两个网桥之间的通道。

qvb：Quantum veth, bridge-side

qvo：Quantum veth, OVS-side

TAP 设备：模拟一个二层的网络设备，可以接受和发送二层网包。

TUN 设备：模拟一个三层的网络设备，可以接受和发送三层网包。

iptables：Linux 上常见的实现安全策略的防火墙软件。

Vlan：虚拟 lan，同一个物理 lan 下用标签实现隔离，可用标号为 1-4094。

namespace：用来实现隔离的一套机制，不同 namespace 之间彼此不可见。

1.2 概念

Neutron 管理下面的实体：

- 网络： 隔离的 L2 域，可以是虚拟、逻辑或交换，同一个网络中的主机彼此 L2 可见。
- 子网： IP 地址快，其中每个虚拟机有一个 IP，同一个子网的主机彼此 L3 可见。
- 端口： 网络上虚拟、逻辑或交换端口。

所有这些实体都是虚拟的，拥有自动生成的唯一标示 id，支持 CRUD 功能，并在数据库中跟踪记录状态。

1.2.1 网络

隔离的 L2 广播域，一般是创建它的用户所有。用户可以拥有多个网络。网络是最基础的，子网和端口都需要关联到网络上。网络的属性如所示。

1.2.2 子网

子网代表了一组分配了 IP 的虚拟机。每个子网必须有一个 CIDR 和关联到一个网络。IP 可以从 CIDR 或者用户指定池中选取。

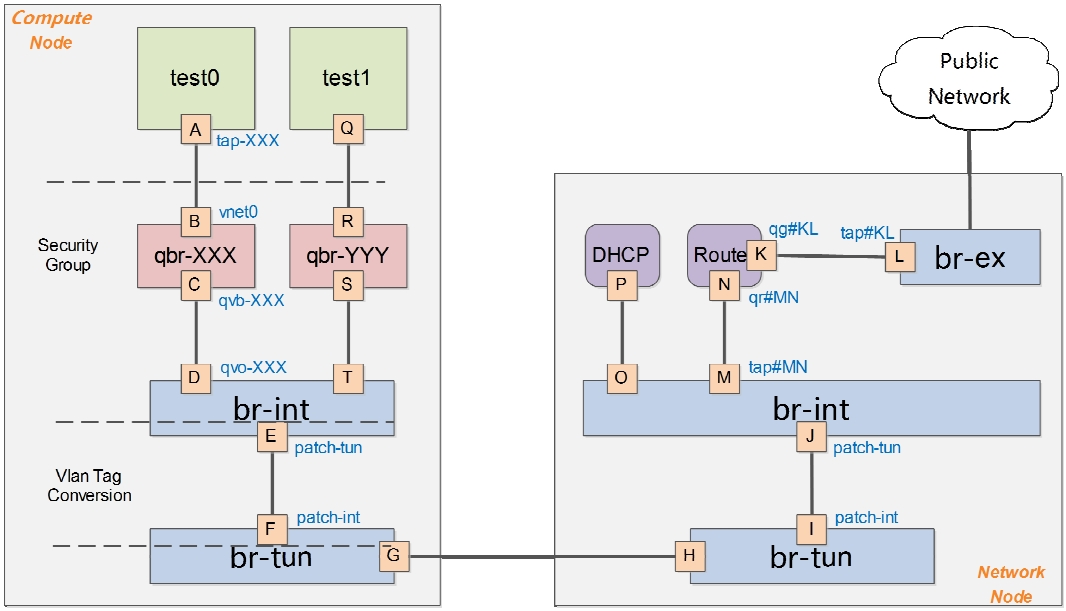
子网可能会有一个网关、一组 DNS 和主机路由。

1.2.3 端口

逻辑网络交换机上的一个虚拟交换口。虚拟机挂载他们的网卡到这些端口上。逻辑口往往定义了挂载到它上面的网卡的 MAC 地址和 IP 地址。当端口有 IP 的时候，意味着它属于某个子网。

1.3 GRE 模式

Error! Reference source not found.给出了在 OpenStack 中网络实现的一个简化的架构示意。OpenStack 中网络实现包括 vlan 和 gre 两种模式，此处以 gre 模式为例。



图表 1 网络基本架构

在 OpenStack 中，所有网络有关的逻辑管理均在 Network 节点中实现，例如 DNS、DHCP 以及路由等。Compute 节点上只需要对所部属的虚拟机提供基本的网络功能支持，包括隔离不同租户的虚拟机和进行一些基本的安全策略管理（即 security group）。

1.3.1 Compute 节点

以 **Error! Reference source not found.**为例，Compute 节点上包括两台虚拟机 test0 和 test1，分别经过一个网桥（如 qbr-XXX）连接到 br-int 网桥上。br-int 网桥再经过 br-tun 网桥（物理网络是 GRE 实现）连接到物理主机外部网络。

对于物理网络通过 vlan 来隔离的情况，一般会存在一个 br-eth 网桥。

1.3.1.1 qbr

在 test0 中，虚拟机的网卡实际上连接到了物理机的一个 TAP 设备（即 A，常见名称如 tap-XXX）上，A 则进一步通过 VETH pair（A-B）连接到网桥 qbr-XXX 的端口 vnet0（端口 B）上，之后再通过 VETH pair（C-D）连到 br-int 网桥上。一般 C 的名字格式为 qvb-XXX，而 D 的名字格式为 qvo-XXX。注意它们的名称除了前缀外，后面的 id 都是一样的，表示位于同一个虚拟机网络到物理机网络的连接上。

之所以 TAP 设备 A 没有直接连接到网桥 br-int 上，是因为 OpenStack 需要单独在一个网桥上通过 iptables 实现 security group 的安全策略功能。目前 openvswitch 并不支持应用 iptables 规则的 Tap 设备。

因为 qbr 的存在主要是为了实现 security group 功能，有时候也被称为 firewall bridge。详见1.3.1.4。

1.3.1.2 br-int

br-int 是集成综合的 bridge，主要负责处理 vlan 标签。

一个典型的 br-int 如下所示：

```
# ovs-vsctl show
Bridge br-int
    Port "qvo-XXX"
        tag: 1
        Interface "qvo-XXX"
    Port patch-tun
        Interface patch-tun
            type: patch
            options: {peer=patch-int}
    Port br-int
        Interface br-int
            type: internal
```

其中 br-int 为内部端口。

端口 qvo-XXX 带有 tag1，说明这个口是一个 1 号 vlan 的 access 端口。虚拟机从该端口发送到 br-int 的网包将被分配上 vlan tag 1，而其他带有 vlan tag 1 的网包则可以在去掉 vlan tag 后从该端口发出（具体请查询 vlan access 端口）。这个 vlan tag 是用来实现不同网络相互隔离的，比如租户创建一个网络（neutron net-create），则会被分配一个唯一的 vlan tag。

端口 patch-tun（即端口 E）则连接到 br-tun 上，实现到外部网络的隧道。

1.3.1.3 br-tun

br-tun 将带有 vlan tag 的流量转换到对应的 gre 隧道。

如果是基于 vlan 的物理网络，则 br-eth 网桥会将带有内部 vlan tag 的流量转换到外部的 vl

an tag 来实现隔离。

查看 br-tun 上的转发规则：

```
# ovs-ofctl dump-flows br-run
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=422.158s, table=0, n_packets=2, n_bytes=120, idle_age=55, priority=3,tun_id=0x2,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=mod_vlan_vid:1,output:1
 cookie=0x0, duration=421.948s, table=0, n_packets=64, n_bytes=8337, idle_age=31, priority=3,tun_id=0x2,dl_dst=fa:16:3e:dd:c1:62 actions=mod_vlan_vid:1,NORMAL
 cookie=0x0, duration=422.357s, table=0, n_packets=82, n_bytes=10443, idle_age=31, priority=4,in_port=1,dl_vlan=1 actions=set_tunnel:0x2,NORMAL
 cookie=0x0, duration=1502.657s, table=0, n_packets=8, n_bytes=596, idle_age=423, priority=1 actions=drop
```

其中最后一条是默认规则，丢弃所有的包。

第一条规则是匹配所有的在 tunnel 2 上的多播包，打上 vlan tag 1，然后从端口 patch-int（即端口 F，在 br-tun 上的标号为 1）发出去。

第二条规则是匹配所有的在 tunnel 2 上的，且目标 mac 是 fa:16:3e:dd:c1:62（即 Compute 节点上某个虚拟机）的网包，打上 vlan tag 1，然后从 patch-int 口发出去。

第三一条是所有从端口 patch-int 到达的，带有 vlan tag 1 的网包，则打上 tunnel 号 2，然后从 GRE 隧道发出。

1.3.1.4 Security group 实现

Security group 的实现，目前是放在 qbr***这样的 Linux 传统 bridge 上的，是基于 iptables 服务。例如查找 qbr-XXX 上相关的 iptables 规则。

```
# iptables -S | grep tap-XXX
-A quantum-openvswi-FORWARD -m physdev --physdev-out tap-XXX --physdev-is-bridged -j quantum-openvswi-sg-chain
-A quantum-openvswi-FORWARD -m physdev --physdev-in tap-XXX --physdev-is-bridged -j quantum-openvswi-sg-chain
-A quantum-openvswi-INPUT -m physdev --physdev-in tap-XXX --physdev-is-bridged -j quantum-openvswi-o7c7ae61e-0
-A quantum-openvswi-sg-chain -m physdev --physdev-out tap-XXX --physdev-is-bridged -j quantum-openvswi-i7c7ae61e-0
-A quantum-openvswi-sg-chain -m physdev --physdev-in tap-XXX --physdev-is-bridged -j quantum-openvswi-o7c7ae61e-0
```

可以看出，进出 tap-XXX 口的 FORWARD 链上的流量都被扔到了 quantum-openvswi-sg-chain 这个链，quantum-openvswi-sg-chain 上是 security group 具体的实现（两条规则，访问虚拟机的流量扔给 quantum-openvswi-i7c7ae61e-0；从虚拟机出来的扔给 quantum-openvswi-o7c7ae61e-0）。

INPUT 链上的流量被扔到了 quantum-openvswi-o7c7ae61e-0 链。

quantum-openvswi-o7c7ae61e-0 链负责从虚拟机出来的流量的处理，上面的默认规则有：

```
-A quantum-openvswi-o7c7ae61e-0 -m mac ! --mac-source FA:16:3E:03:00:E7 -j DROP
-A quantum-openvswi-o7c7ae61e-0 -p udp -m udp --sport 68 --dport 67 -j RETURN
-A quantum-openvswi-o7c7ae61e-0 ! -s 10.1.0.2/32 -j DROP
-A quantum-openvswi-o7c7ae61e-0 -p udp -m udp --sport 67 --dport 68 -j DROP
-A quantum-openvswi-o7c7ae61e-0 -m state --state INVALID -j DROP
-A quantum-openvswi-o7c7ae61e-0 -m state --state RELATED,ESTABLISHED -j RETURN
-A quantum-openvswi-o7c7ae61e-0 -j RETURN
-A quantum-openvswi-o7c7ae61e-0 -j quantum-openvswi-sg-fallback
```

第 1、3 条是仅允许指定的源 mac 和源 IP 的流量。

第 2 条是对于 DHCP 请求，返回到上层链。

第 3 条是禁止虚拟机往外发送 DHCP 的响应。

其他条是处理失败状态和默认行为。

quantum-openvswi-i7c7ae61e-0 这条链管理要访问虚拟机的入口流量，在通过命令：

```
# neutron security-group-rule-create --protocol tcp \
  --port-range-min 22 --port-range-max 22 --direction ingress default
```

打开对虚拟机的 ssh 访问之后，默认的规则包括：

```
-A quantum-openvswi-i7c7ae61e-0 -m state --state INVALID -j DROP
-A quantum-openvswi-i7c7ae61e-0 -m state --state RELATED,ESTABLISHED -j RETURN
-A quantum-openvswi-i7c7ae61e-0 -p icmp -j RETURN
-A quantum-openvswi-i7c7ae61e-0 -p tcp -m tcp --dport 22 -j RETURN
-A quantum-openvswi-i7c7ae61e-0 -p tcp -m tcp --dport 80 -j RETURN
-A quantum-openvswi-i7c7ae61e-0 -s 10.1.0.3/32 -p udp -m udp --sport 67 --dport 68 -j
RETURN
-A quantum-openvswi-i7c7ae61e-0 -j quantum-openvswi-sg-fallback
```

这些规则默认允许访问虚拟机的 22 和 80 端口，允许 DHCP 应答从 DHCP 服务器 10.1.0.3 这个地址返回。

1.3.2 Network 节点

1.3.2.1 br-tun

Compute 节点上发往 GRE 隧道的网包最终抵达 Network 节点上的 br-tun，该网桥的规则包括：

```
# ovs-ofctl dump-flows br-tun
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1239.229s, table=0, n_packets=23, n_bytes=4246, idle_age=15,
priority=3,tun_id=0x2,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00
actions=mod_vlan_vid:1,output:1
```

```
 cookie=0x0, duration=524.477s, table=0, n_packets=15, n_bytes=3498, idle_age=10, priority=3,tun_id=0x2,dl_dst=fa:16:3e:83:69:cc actions=mod_vlan_vid:1,NORMAL
 cookie=0x0, duration=1239.157s, table=0, n_packets=50, n_bytes=4565, idle_age=148, priority=3,tun_id=0x2,dl_dst=fa:16:3e:aa:99:3c actions=mod_vlan_vid:1,NORMAL
 cookie=0x0, duration=1239.304s, table=0, n_packets=76, n_bytes=9419, idle_age=10, priority=4,in_port=1,dl_vlan=1 actions=set_tunnel:0x2,NORMAL
 cookie=0x0, duration=1527.016s, table=0, n_packets=12, n_bytes=880, idle_age=527, priority=1 actions=drop
```

这些规则跟 Compute 节点上 br-tun 的规则相似，完成 tunnel 跟 vlan 之间的转换。

第一条规则也是处理 tunnel 来的多播包，打上 vlan tag 1（这里必须跟 Compute 节点上内部 vlan tag 保持一致），然后从 patch-int 端口（即 I，在 br-tun 上端口号是 1）发出。

第二条规则从 tunnel 来的，目标 mac 是 fa:16:3e:83:69:cc（目标是 dhcp server）的网包，打上 vlan tag 1，然后转发。

第三条规则从 tunnel 来的，目标 mac 是 fa:16:3e:aa:99:3c（目标是一个 router）的网包，打上 vlan tag 1，然后转发。

第四条规则，将从 patch-int 端口抵达的，vlan tag 是 1 的网包，转化到 tunnel 号是 2，然后发出。

最后一条规则是默认规则，丢弃所有其他网包。

1.3.2.2 br-int

该集成网桥上挂载了很多进程来提供网络服务，包括路由器、DHCP 服务器等。这些进程不同的租户可能都需要，彼此的地址空间可能冲突，也可能跟物理网络的地址空间冲突，因此都运行在独立的网络名字空间中。

1.3.2.3 网络名字空间

在 linux 中，网络名字空间可以被认为是隔离的拥有单独网络栈（网卡、路由转发表、iptables）的环境。网络名字空间经常用来隔离网络设备和服务，只有拥有同样网络名字空间的设备，才能看到彼此。

可以用 ip netns list 命令来查看已经存在的名字空间。

```
# ip netns
qdhcp-88b1609c-68e0-49ca-a658-f1edff54a264
qrouter-2d214fde-293c-4d64-8062-797f80ae2d8f
```

qdhcp 开头的名字空间是 dhcp 服务器使用的，qrouter 开头的则是 router 服务使用的。

可以通过 ip netns exec namespaceid command 来在指定的网络名字空间中执行网络命令，例如

```
# ip netns exec qdhcp-88b1609c-68e0-49ca-a658-f1edff54a264 ip addr
71: ns-f14c598d-98: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether fa:16:3e:10:2f:03 brd ff:ff:ff:ff:ff:ff
```

```
    inet 10.1.0.3/24 brd 10.1.0.255 scope global ns-f14c598d-98
    inet6 fe80::f816:3eff:fe10:2f03/64 scope link
        valid_lft forever preferred_lft forever
```

可以看到，dhcp 服务的网络名字空间中只有一个网络接口“ns-f14c598d-98”，它连接到 br-int 的 tapf14c598d-98 接口上。

1.3.2.4 dhcp 服务

dhcp 服务是通过 dnsmasq 进程（轻量级服务器，可以提供 dns、dhcp、tftp 等服务）来实现的，该进程绑定到 dhcp 名字空间中的 br-int 的接口上。可以查看相关的进程。

```
# ps -fe | grep 88b1609c-68e0-49ca-a658-f1edff54a264
nobody    23195     1  0 Oct26 ?        00:00:00 dnsmasq --no-hosts --no-resolv
--strict-order --bind-interfaces --interface=ns-f14c598d-98 --except-interface=lo
--pid-file=/var/lib/quantum/dhcp/88b1609c-68e0-49ca-a658-f1edff54a264/pid
--dhcp-hostsfile=/var/lib/quantum/dhcp/88b1609c-68e0-49ca-a658-f1edff54a264/host
--dhcp-optsfile=/var/lib/quantum/dhcp/88b1609c-68e0-49ca-a658-f1edff54a264/opts
--dhcp-script=/usr/bin/quantum-dhcp-agent-dnsmasq-lease-update --leasefile-ro
--dhcp-range=tag0,10.1.0.0,static,120s --conf-file= --domain=openstacklocal
root      23196 23195  0 Oct26 ?        00:00:00 dnsmasq --no-hosts --no-resolv
--strict-order --bind-interfaces --interface=ns-f14c598d-98 --except-interface=lo
--pid-file=/var/lib/quantum/dhcp/88b1609c-68e0-49ca-a658-f1edff54a264/pid
--dhcp-hostsfile=/var/lib/quantum/dhcp/88b1609c-68e0-49ca-a658-f1edff54a264/host
--dhcp-optsfile=/var/lib/quantum/dhcp/88b1609c-68e0-49ca-a658-f1edff54a264/opts
--dhcp-script=/usr/bin/quantum-dhcp-agent-dnsmasq-lease-update --leasefile-ro
--dhcp-range=tag0,10.1.0.0,static,120s --conf-file= --domain=openstacklocal
```

1.3.2.5 router 服务

同样的，router 服务也运行在自己的名字空间中，可以通过如下命令查看：

```
# ip netns exec qrouter-2d214fde-293c-4d64-8062-797f80ae2d8f ip addr
66: qg-d48b49e0-aa: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether fa:16:3e:5c:a2:ac brd ff:ff:ff:ff:ff:ff
    inet 172.24.4.227/28 brd 172.24.4.239 scope global qg-d48b49e0-aa
    inet 172.24.4.228/32 brd 172.24.4.228 scope global qg-d48b49e0-aa
    inet6 fe80::f816:3eff:fe5c:a2ac/64 scope link
        valid_lft forever preferred_lft forever
68: qr-c2d7dd02-56: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP qlen 1000
    link/ether fa:16:3e:ea:64:6e brd ff:ff:ff:ff:ff:ff
    inet 10.1.0.1/24 brd 10.1.0.255 scope global qr-c2d7dd02-56
```

```
    inet6 fe80::f816:3eff:feea:646e/64 scope link
       valid_lft forever preferred_lft forever
```

可以看出，该名字空间中包括两个网络接口。

第一个接口 qg-d48b49e0-aa（即 K）是外部接口（qg=q gateway），将路由器的网关指向默认网关（通过 router-gateway-set 命令指定），这个接口连接到 br-ex 上的 tapd48b49e0-aa（即 L）。

第二个接口 qr-c2d7dd02-56（即 N，qr=q bridge）跟 br-int 上的 tapc2d7dd02-56 口（即 M）相连，将 router 进程连接到集成网桥上。

查看该名字空间中的路由表：

```
# ip netns exec qrouter-2d214fde-293c-4d64-8062-797f80ae2d8f ip route
172.24.4.224/28 dev qg-d48b49e0-aa  proto kernel  scope link  src 172.24.4.227
10.1.0.0/24 dev qr-c2d7dd02-56  proto kernel  scope link  src 10.1.0.1
default via 172.24.4.225 dev qg-d48b49e0-aa
```

其中，第一条规则是将到 172.24.4.224/28 段的访问都从网卡 qg-d48b49e0-aa（即 K）发出。

第二条规则是将到 10.1.0.0/24 段的访问都从网卡 qr-c2d7dd02-56（即 N）发出。

最后一条是默认路由，所有的通过 qg-d48b49e0-aa 网卡（即 K）发出。

floating ip 服务同样在路由器名字空间中实现，例如如果绑定了外部的 floating ip 172.24.4.228 到某个虚拟机 10.1.0.2，则 nat 表中规则为：

```
# ip netns exec qrouter-2d214fde-293c-4d64-8062-797f80ae2d8f iptables -t nat -S
-P PREROUTING ACCEPT
-P POSTROUTING ACCEPT
-P OUTPUT ACCEPT
-N quantum-l3-agent-OUTPUT
-N quantum-l3-agent-POSTROUTING
-N quantum-l3-agent-PREROUTING
-N quantum-l3-agent-float-snat
-N quantum-l3-agent-snat
-N quantum-postrouting-bottom
-A PREROUTING -j quantum-l3-agent-PREROUTING
-A POSTROUTING -j quantum-l3-agent-POSTROUTING
-A POSTROUTING -j quantum-postrouting-bottom
-A OUTPUT -j quantum-l3-agent-OUTPUT
-A quantum-l3-agent-OUTPUT -d 172.24.4.228/32 -j DNAT --to-destination 10.1.0.2
-A quantum-l3-agent-POSTROUTING ! -i qg-d48b49e0-aa ! -o qg-d48b49e0-aa -m conntrack !
--ctstate DNAT -j ACCEPT
-A quantum-l3-agent-PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j REDIRECT
--to-ports 9697
-A quantum-l3-agent-PREROUTING -d 172.24.4.228/32 -j DNAT --to-destination 10.1.0.2
-A quantum-l3-agent-float-snat -s 10.1.0.2/32 -j SNAT --to-source 172.24.4.228
-A quantum-l3-agent-snat -j quantum-l3-agent-float-snat
-A quantum-l3-agent-snat -s 10.1.0.0/24 -j SNAT --to-source 172.24.4.227
-A quantum-postrouting-bottom -j quantum-l3-agent-snat
```

其中 SNAT 和 DNAT 规则完成外部 floating ip 到内部 ip 的映射：

```
-A quantum-l3-agent-OUTPUT -d 172.24.4.228/32 -j DNAT --to-destination 10.1.0.2
-A quantum-l3-agent-PREROUTING -d 172.24.4.228/32 -j DNAT --to-destination 10.1.0.2
-A quantum-l3-agent-float-snat -s 10.1.0.2/32 -j SNAT --to-source 172.24.4.228
```

另外有一条 SNAT 规则把所有其他的内部 IP 出来的流量都映射到外部 IP 172.24.4.227。这样即使在内部虚拟机没有外部 IP 的情况下，也可以发起对外网的访问。

```
-A quantum-l3-agent-snat -s 10.1.0.0/24 -j SNAT --to-source 172.24.4.227
```

1.3.2.6 br-ex

br-ex 上直接连接到外部物理网络，一般情况下网关在物理网络中已经存在，则直接转发即可。

如果对外部网络的网关地址配置到了 br-ex（即 br-ex 作为一个网关）：

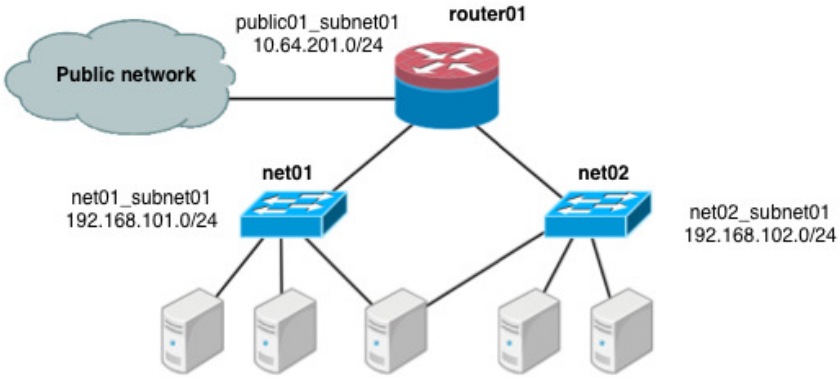
```
# ip addr add 172.24.4.225/28 dev br-ex
```

需要将内部虚拟机发出的流量进行 SNAT，之后发出。

```
# iptables -A FORWARD -d 172.24.4.224/28 -j ACCEPT
# iptables -A FORWARD -s 172.24.4.224/28 -j ACCEPT
# iptables -t nat -I POSTROUTING 1 -s 172.24.4.224/28 -j MASQUERADE
```

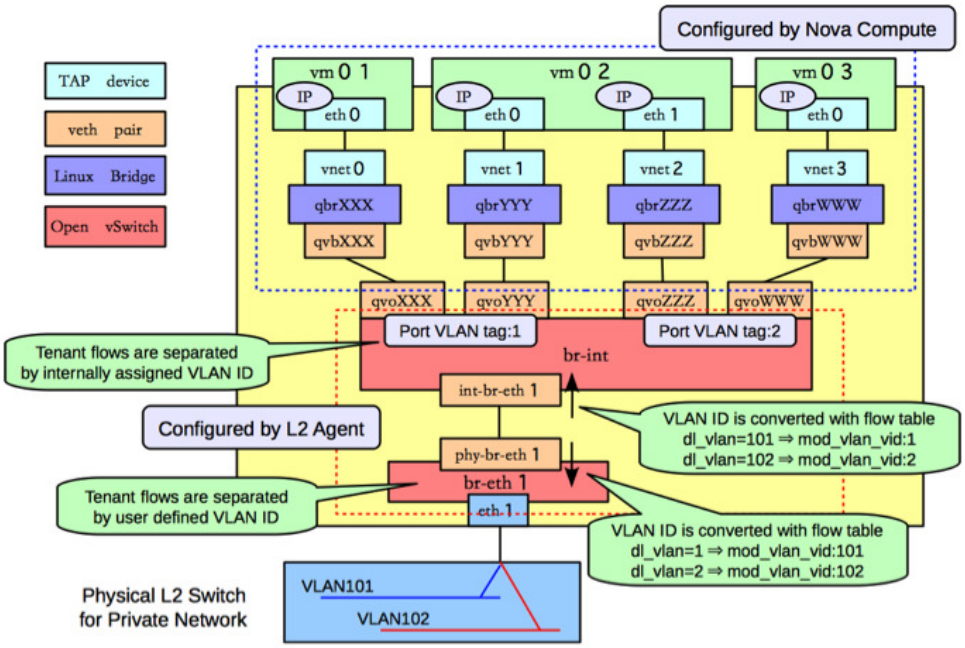
1.4 VLAN 模式

下面进行一些细节的补充讨论，以 Vlan 作为物理网络隔离的实现。假如要实现同一个租户下两个子网，如图表 2 所示：



图表 2 同一个租户的两个子网

1.4.1 Compute 节点

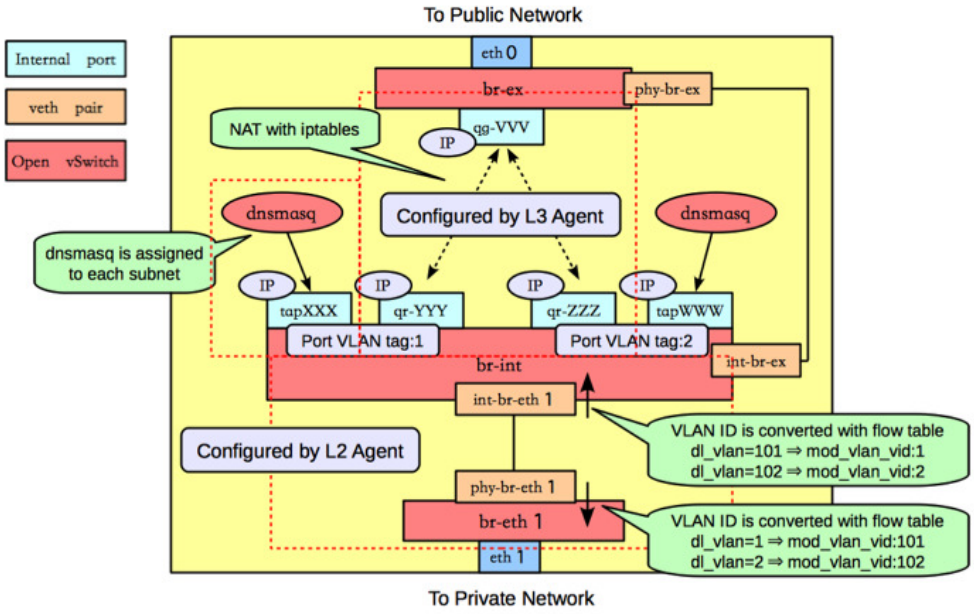


图表 3 Compute 节点网络示意

类似 GRE 模式下，其中，qbr 负责实现安全策略，br-int 负责租户隔离，br-eth1 负责跟计算节点外的网络通信。

br-int 和 br-eth1 分别对从端口 int-br-eth1 和 phy-br-eth1 上到达的网包进行 vlan tag 的处理。此处有两个网，分别带有两个 vlan tag（内部 tag1 对应外部 tag101，内部 tag2 对应外部 tag102）。

1.4.2 Network 节点



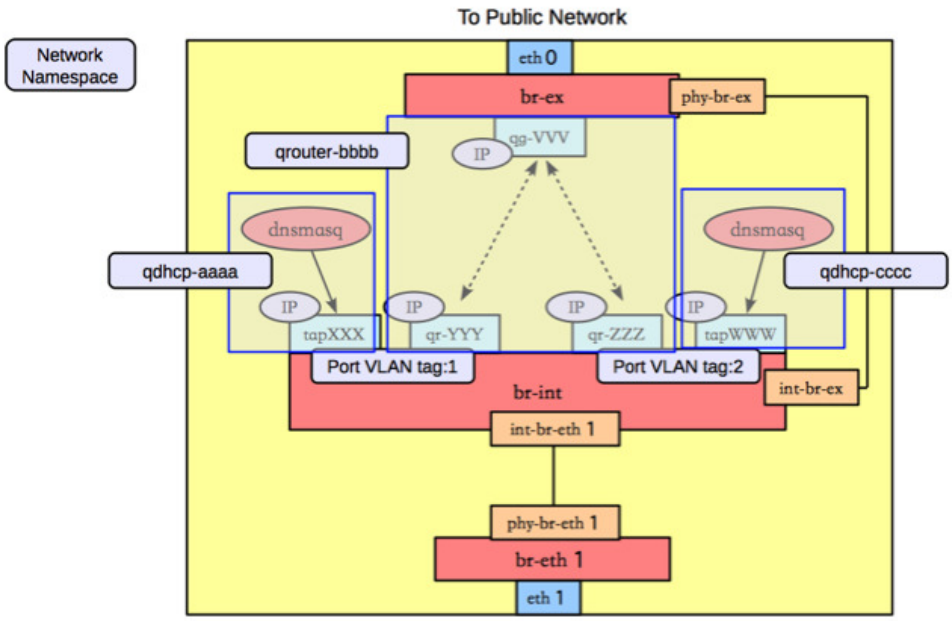
图表 4 Network 节点网络示意

类似 GRE 模式下，br-eth1 收到到达的网包，int-br-eth1 和 phy-br-eth1 上分别进行 vlan 转换，保证到达 br-int 上的网包都是带有内部 vlan tag，到达 br-eth1 上的都是带有外部 vlan tag。

同样的，dnsmasq 负责提供 DHCP 服务，绑定到某个特定的名字空间上，每个需要 DHCP 服务的租户网络有自己专属隔离的 DHCP 服务（图中的 tapXXX 和 tapWWW 上各自监听了一个 dnsmasq）。

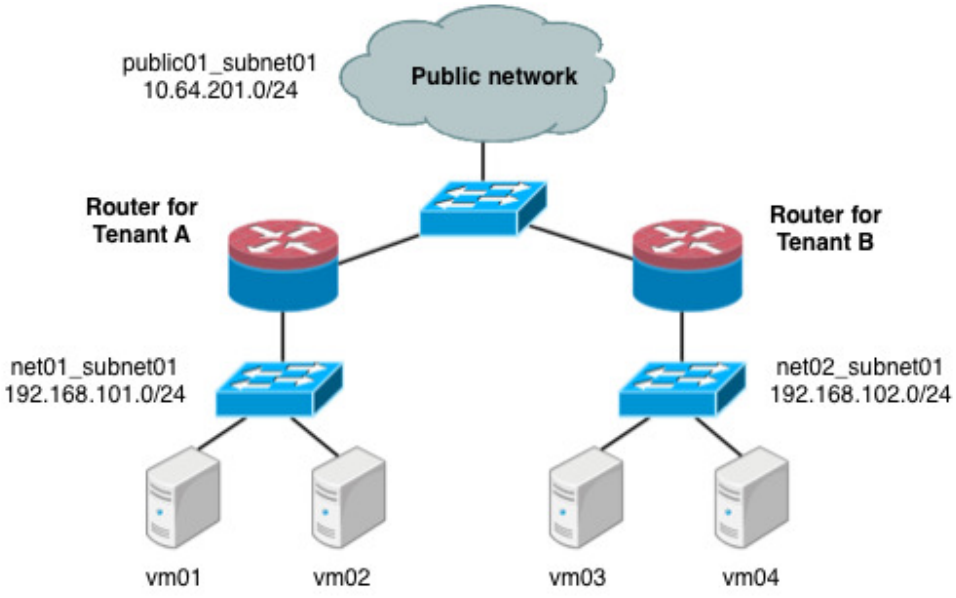
路由是 L3 agent 来实现，每个子网在 br-int 上有一个端口（qr-YYY 和 qr-ZZZ，已配置 IP，分别是各自内部子网的网关），L3 agent 绑定到上面。要访问外部的公共网络，需要通过 L3 agent 发出，而不是经过 int-br-ex 到 phy-br-ex（实际上并没有网包从这个 veth pair 传输）。如果要使用外部可见的 floating IP，L3 agent 仍然需要通过 iptables 来进行 NAT。

每个 L3 agent 或 dnsmasq 都在各自独立的名字空间中，如图表 5 所示，其中同一租户的两个子网都使用了同一个路由器。



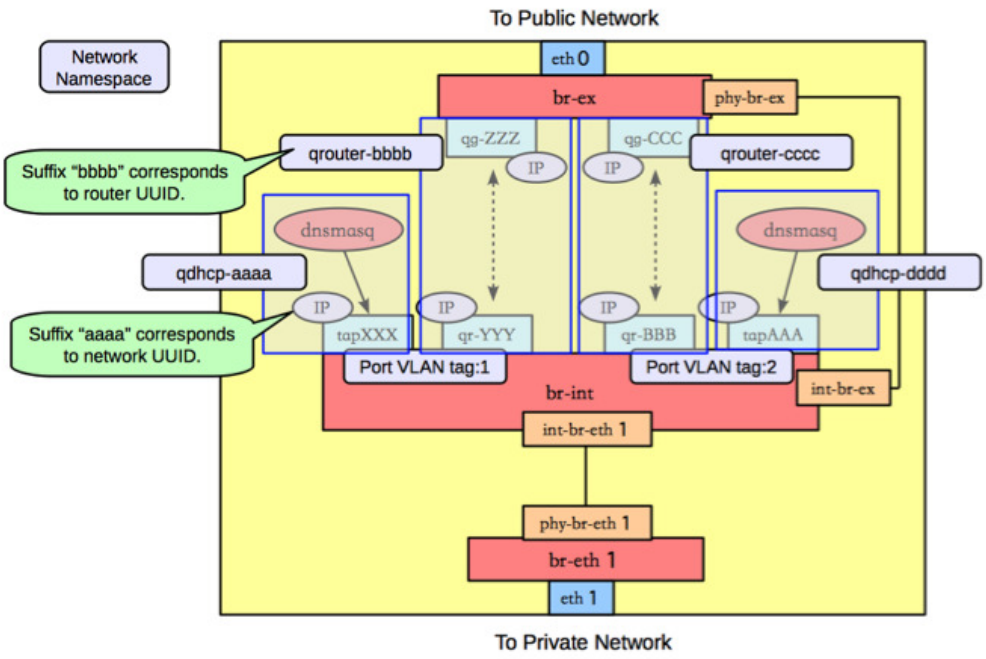
图表 5 每个网络功能进程都在自己的名字空间中

对于子网使用不同路由器的情况，多个路由器会在自己独立的名字空间中。例如要实现两个租户的两个子网的情况，如图表 6 所示。



图表 6 两个租户的两个子网的结构

这种情况下，网络节点上的名字空间如图表 7 所示。



图表 7 两个租户两个子网情况下的名字空间

1.5 参考

[1] http://openstack.redhat.com/Networking_in_too_much_detail

[2] http://masimum.inf.um.es/fjrm/2013/12/26/the-journey-of-a-packet-within-an-openstack-cloud/

[3] http://packetpushers.net/openstack-quantum-network-implementation-in-linux/

[4] http://masimum.inf.um.es/fjrm/2013/12/26/the-journey-of-a-packet-within-an-openstack-cloud/

[5] http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/

Filename: 深入理解 OpenStack 中的网络实现.doc
Directory: E:\My Documents\GitHub\tech_writing\OpenStack
Template: C:\Users\IBM_ADMIN\AppData\Roaming\Microsoft\Templates\Normal.dot
Title:
Subject:
Author: ibm
Keywords:
Comments:
Creation Date: 2013/3/21 15:54:00
Change Number: 398
Last Saved On: 2014/2/27 9:39:00
Last Saved By: ibm
Total Editing Time: 5,242 Minutes
Last Printed On: 2014/2/27 9:43:00
As of Last Complete Printing
Number of Pages: 13
Number of Words: 2,366 (approx.)
Number of Characters: 13,490 (approx.)