[**Neutron 理解（14）：Neutron ML2 + Linux bridge + VxLAN 组网**](http://www.cnblogs.com/sammyliu/p/4985907.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)

（14）[Linux bridge + VXLAN](http://www.cnblogs.com/sammyliu/p/4985907.html%20)

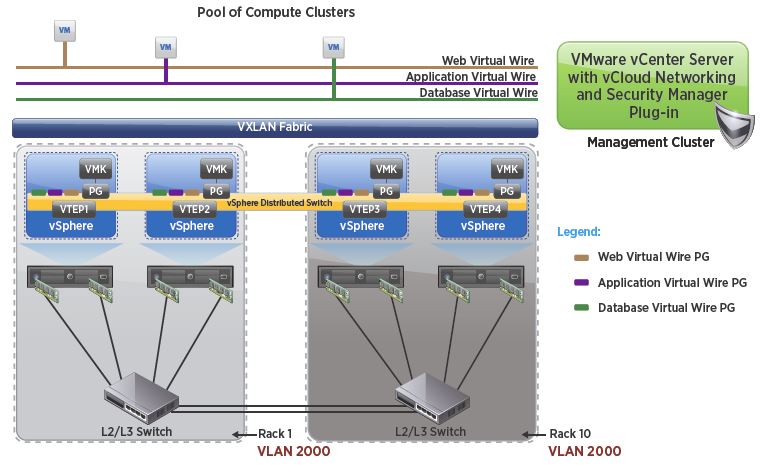
**1. 基础知识**

**1.1 VXLAN 和 Linux 以及 Linux bridge 的关系**

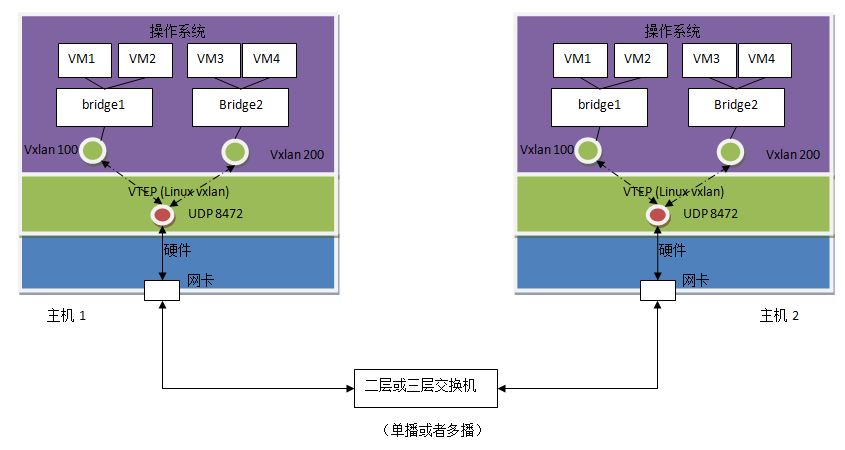
[VXLAN](https://tools.ietf.org/html/rfc7348) 是一个新兴的SDN 标准，它定义了一种新的 overlay 网络，它主要的创造者是 VMware, Cisco 和 Arista。它被设计来消除虚拟化网络世界中的 VLAN 数目的限制。VXLAN 本身是一个多播标准，但是大多数的企业既不情愿启用多播，而且许多网络设备也不支持多播。因此，许多 VXLAN 的实现，比如 Linux vxlan， 就既支持多播也支持单播。VXLAN 的两个主要概念是 VTEP 和 VXLAN ID。 其中，VTEP 负责解包和封包，以及包的传送。它可以由专有硬件来实现，也可以使用纯软件实现。目前比较成熟的软件实现的 VTEP 包括：

* 3.9 版本及以后的带 vxlan 内核模块的 Linux
* Open vSwitch
* VMware vSphere

   以 vSphere 为例，下面是它所实现的 VXLAN 虚拟二层网络的一个参考架构：



可以看出来，这和 Open vSwitch 的网络架构非常类似。对于 Linux 作为 VTEP 来说，使用 linux bridge 替代 VMware 的 VDS 或者 Open vSwitch，其它也是一样的了。



特点：

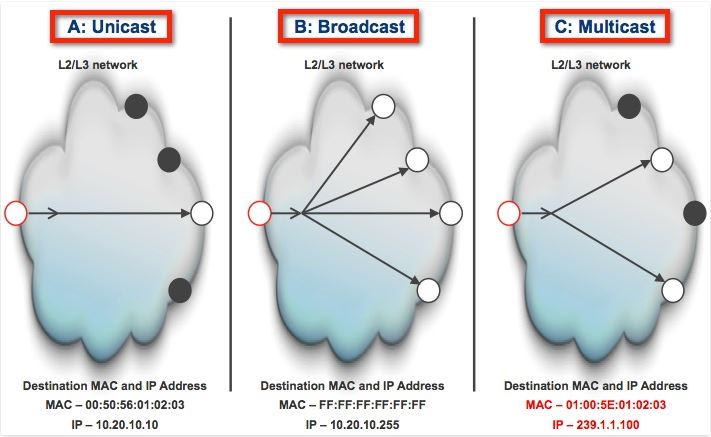
* Linux vxlan 创建一个 UDP Socket，默认在 8472 端口监听
* Linux vxlan 在 UDP socket 上接收到 vxlan 包后，解包，然后根据其中的 vxlan ID 将它转给某个 vxlan interface，然后再通过它所连接的linux bridge 转给虚机
* Linux vxlan 在收到虚机的多播或者广播帧后，将其封装为多播 UDP 包（在使用多播时）或者多个单播包（在使用单播时），在从指定网卡发出
* Linux vxlan 在虚机的单播帧后，如果 fdb 表中包含有目的虚机的 VTEP IP 条目，则将其封装为单播 UDP 包，否则，通过多播或者多个单播，发到指定的 VTEP
* Linux vxlan 在配置了 Proxy ARP 时会根据 fdb 表来响应虚机的 ARP 广播请求

因此：

|  |  |  |  |
| --- | --- | --- | --- |
| **VXLAN 组件** | **vSphere** | **Linux** | **Open vSwitch （OVS）** |
| VTEP | vSphere | Linux | OVS |
| Bridge | VDS | Linux bridge | OVS |

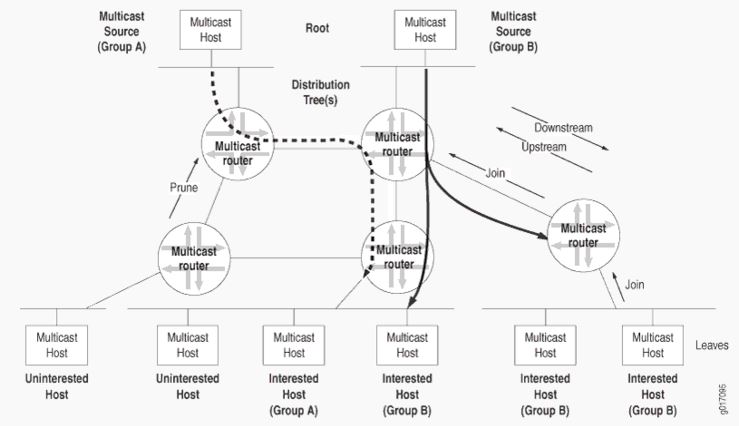
**1.2 多播和多播组**

这是单播、多播和广播的概念：



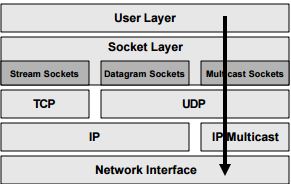
* 单播：使用单播地址来发送一个网络包到一个目的机器。
* 广播：使用广播地址来发送一个网络包到一个网段内的所有机器。
* 多播：使用多播地址来发送一个网络包到一个已经加入一个多播组内的所有成员，这些成员是可以跨网段的。

一个多播IP网络示例：



多播的一些基础知识：

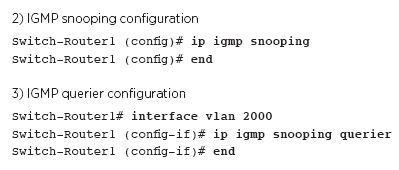
* 多播使用一个特殊的 IP 地址，来将数据从一个发送至发至多个接收者。IANA 规定的多播地址是 224/4，也就是 224.0.0.0 到 239.255.255.255 区间。
* 如果网卡上启用了多播，那么在计算器启动后，它会默认加入多播组 224.0.0.1，该组内包括该网段内的所有启用了多播的网卡。因此，此时的多播和广播就是一样的了。网卡使用 Internet Group Management Protocol (IGMP) 协议去通知本地的启用了多播的路由器它要加入一个多播组。
* IGMP 是一个 IP 层的协议，非常类似于 ICMP。多播路由器使用该协议来维护多播组及其成员，支持的功能包括加入一个多播组、查询组成员和发送组成员报告。多播路由器周期性地发送 IGMP 查询给所有的主机，主机通过响应该请求来加入一个或者多个多播组。
* 多播的路由协议包括 DVMRP （Distance Vector Multicast Routing Protocol） and PIM （Protocol-Independent Multicast），但是，许多的网络并不支持这些协议。
* 如果 XVLAN 使用多播的话，则需要物理网络包括交换机和路由器支持多播。
* 多播只支持UDP 作为上层协议，可见多播不是一种可靠的协议



要支持多播，需要物理网络上做一些设置：

* 往往对虚机使用的数据网络划分一个单独的 VALN，从而控制广播/多播域
* 配置物理交换机上的 IGMP snooping
* 配置物理路由器上所在 VLAN 的 IGMP querier，它会响应各主机的多播查询请求

比如：



**1.3 Linux vxlan**

在支持 vxlan 的 Linux 主机上，可以创建多个多播或者单播的 vxlan interface，每个interface 是一个 vxlan tunnel 的 endpoint。比如在主机1上：

创建多播 interface 1：ip link add vxlan0 type vxlan id 42 group 239.1.1.1 dev eth1 dstport 4789

创建多播 interface 2：ip link add vxlan2 type vxlan id 43 group 239.1.1.2 dev eth1 dstport 4790

创建单播 interface 3：ip link add vxlan2 type vxlan id 44 dev eth1 port 32768 61000 proxy ageing 300

然后在每一个设备启动后，如果使用的是多播地址，vxlan 就会加入相应的多播组，并且创建相应的 UDP socket。

[复制代码](javascript:void(0);)

/\* Start ageing timer and join group when device is brought up \*/

static int vxlan\_open(struct net\_device \*dev) #对每个 vxlan dev 使用

{

......

ret = vxlan\_sock\_add(vxlan); #创建 UDP socket

if (ret < 0)

return ret;

if (vxlan\_addr\_multicast(&vxlan->default\_dst.remote\_ip)) {

ret = vxlan\_igmp\_join(vxlan); #加入多播组

......

}

}

.....

}

[复制代码](javascript:void(0);)

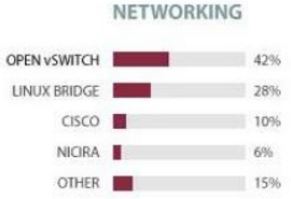
在主机2上，使用同样的 dstport， group 和 vxlan id 创建相应的 vxlan interface，即可创建多个虚拟的 vxlan tunnel。

因此，只要没有冲突，一个Linux 主机上可以加入多个多播组，创建多个 UDP socket。

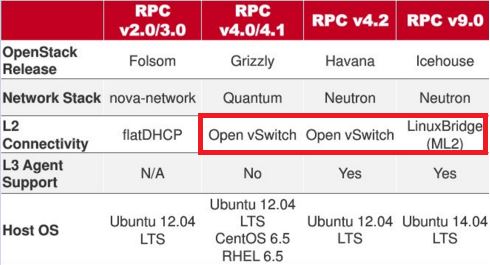
Linux vxlan 模块的代码在 <https://github.com/torvalds/linux/blob/master/drivers/net/vxlan.c>

**2. 为什么Neutron 要用 Linux bridge agent 而不使用 Open vSwitch agent？**

    我们都知道，OpenStack 社区官方的安装文档的步骤都是以 Open vSwitch 为例子的。而且从OpenStack 用户调查来看，使用 OVS 的人比使用 linux bridge 多很多。



   那还有人使用 Linux bridge 吗？答案是有的，据我所知，国内厂商比如海云捷迅就推荐私有云中的两种配置：Linux bridge + VLAN 以及 Linux bridge + VxLAN。而且，Liberty 版本之前社区官方文档都是使用 neutron-plugin-openvswitch-agent， 但是 Liberty 版本转为使用 neutron-plugin-linuxbridge-agent， 不知道这背后究竟发生了什么。国外的 Rackspace 也已经使用Linux bridge 替代 Open vSwitch，请参见下图：



本段以 Rackspace 为例，说明他们使用 Linux bridge 的理由：

（1）Open vSwitch 目前还存在不少稳定性问题，比如：

* Kernetl panics 1.10
* ovs-switched segfaults 1.11
* 广播风暴
* Data corruption 2.01

（2）为什么可以使用 Linux bridge？

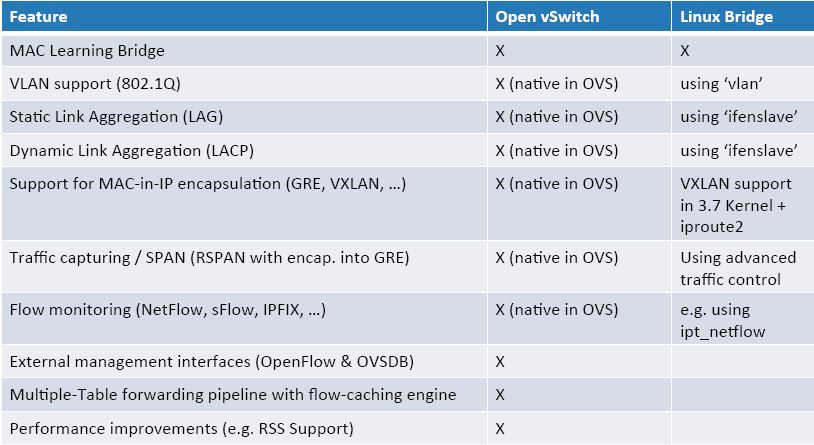
* 稳定性和可靠性要求：Linux bridge 有十几年的使用历史，非常成熟。
* 易于问题诊断 （troubleshooting）
* 社区也支持
* 还是可以使用 Overlay 网络 VxLAN 9需要 Linux 内核 3.9 版本或者以上)

（3）使用 Linux bridge 的局限性

* Neutron DVR 还不支持 Linux bridge
* 不支持 GRE
* 一些 OVS 提供但是 Neutorn 不支持的功能

（4）长期来看，随着稳定性的进一步提高，Open vSwitch 会在生产环境中成为主流。

**Linux bridge 和 Open vSwitch 的功能对比：**

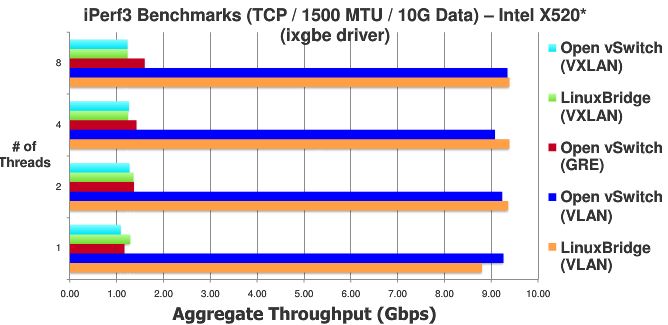
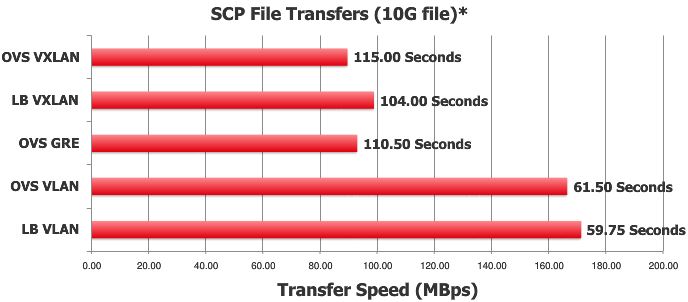


可以看出：

（1）OVS 将各种功能都原生地实现在其中，这在起始阶段不可避免地带来潜在的稳定性和可调试性问题

（2）Linux bridge 依赖各种其他模块来实现各种功能，而这些模块的发布时间往往都已经很长，稳定性较高

（3）两者在核心功能上没有什么差距，只是在集中管控和性能优化这一块OVS有一些新的功能或者优化。但是，从测试结果看，两者的性能没有明显差异：

总之，目前，除了 SDN 对集中管控的需求，Linux bridge 是个较好的选择。

**3. 使用 Linux bridge + VXLAN 组网**

**3.1 配置**

在控制、网络和计算节点上修改 /etc/neutron/plugins/ml2/ml2\_conf.ini 中的如下配置项：

[复制代码](javascript:void(0);)

[ml2]

type\_drivers = flat,vlan,gre,vxlan

tenant\_network\_types = vxlan

mechanism\_drivers = linuxbridge

[ml2\_type\_vxlan]

vni\_ranges = 1001:2000

[vxlan]

local\_ip = 10.0.0.13

enable\_vxlan = true

[securitygroup]

firewall\_driver = neutron.agent.linux.iptables\_firewall.IptablesFirewallDriver

[agent]

tunnel\_types = vxlan

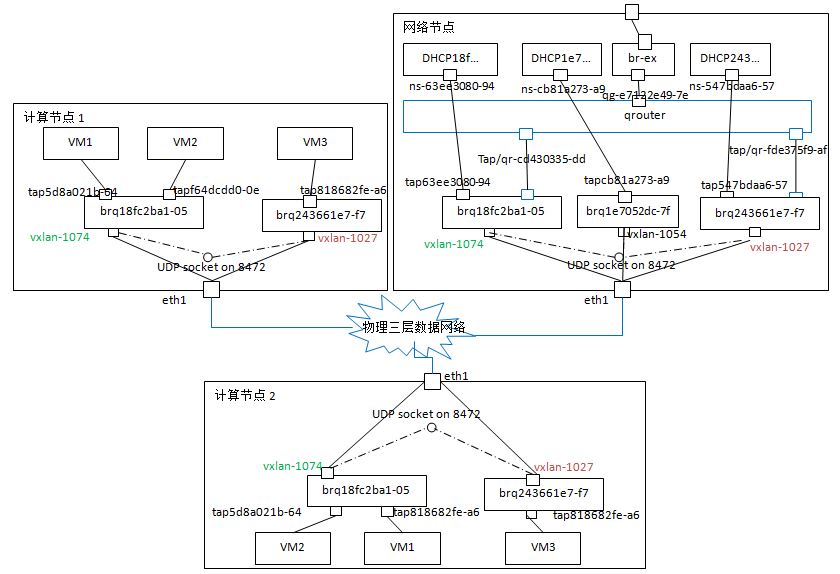
[复制代码](javascript:void(0);)

**3.2 组网**

**3.2.1 配置步骤**

1. 创建三个网络，每个网络都启动DHCP，每个网络创建一个子网
2. 创建一个 router，连接其中两个网络的若干子网
3. 创建多个虚机

**3.2.2 租户网络的构成**



计算节点1上有三个虚机，分布在两个网络上：

（1）两个 Linux bridge

[复制代码](javascript:void(0);)

root@compute1:~# brctl show

bridge name bridge id STP enabled interfaces

brq18fc2ba1-05 8000.9292780d149d no tap5d8a021b-64

tapf64dcdd0-0e

vxlan-1074

brq243661e7-f7 8000.66239c87f16c no tap818682fe-a6

vxlan-1027

[复制代码](javascript:void(0);)

（2）VXLAN 在端口 8472 上创建了一个 UDP Socket

root@compute1:~# netstat -lu

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State

udp 0 0 192.168.122.1:domain \*:\*

udp 0 0 \*:bootps \*:\*

udp 768 0 \*:8472 \*:\*

（3）创建了两个 vxlan network interface

[复制代码](javascript:void(0);)

root@compute1:~# ip -d link show dev vxlan-1074

14: vxlan-1074: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master brq18fc2ba1-05 state UNKNOWN mode DEFAULT group default

link/ether 92:92:78:0d:14:9d brd ff:ff:ff:ff:ff:ff promiscuity 1

vxlan id 1074 group 224.0.0.1 dev eth1 port 32768 61000 ageing 300

root@compute1:~# ip -d link show dev vxlan-1027

12: vxlan-1027: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master brq243661e7-f7 state UNKNOWN mode DEFAULT group default

link/ether 66:23:9c:87:f1:6c brd ff:ff:ff:ff:ff:ff promiscuity 1

vxlan id 1027 group 224.0.0.1 dev eth1 port 32768 61000 ageing 300

[复制代码](javascript:void(0);)

（4）vxlan interface 使用网卡 eth1

网络节点上：

（1）一个 qrouter network namespace

（2）三个 dhcp network namespace，每个 enable DHCP 的网络各一个

（3）三个 Linux bridge，每个 network 一个

（4）三个 vxlan network interface（ip link），每个 network 一个 bridge， 每个 linux bridge 上只有一个 vxlan interface

[复制代码](javascript:void(0);)

root@controller:~/s1# ip -d link show dev vxlan-1027

11: vxlan-1027: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master brq243661e7-f7 state UNKNOWN mode DEFAULT group default

link/ether a6:6b:39:67:60:68 brd ff:ff:ff:ff:ff:ff promiscuity 1

vxlan id 1027 group 224.0.0.1 dev eth1 port 32768 61000 ageing 300

root@controller:~/s1# ip -d link show dev vxlan-1074

5: vxlan-1074: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master brq18fc2ba1-05 state UNKNOWN mode DEFAULT group default

link/ether 8a:e2:2f:2d:e7:48 brd ff:ff:ff:ff:ff:ff promiscuity 1

vxlan id 1074 group 224.0.0.1 dev eth1 port 32768 61000 ageing 300

root@controller:~/s1# ip -d link show dev vxlan-1054

8: vxlan-1054: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master brq1e7052dc-7f state UNKNOWN mode DEFAULT group default

link/ether 5e:c7:d4:85:23:fc brd ff:ff:ff:ff:ff:ff promiscuity 1

vxlan id 1054 group 224.0.0.1 dev eth1 port 32768 61000 ageing 300

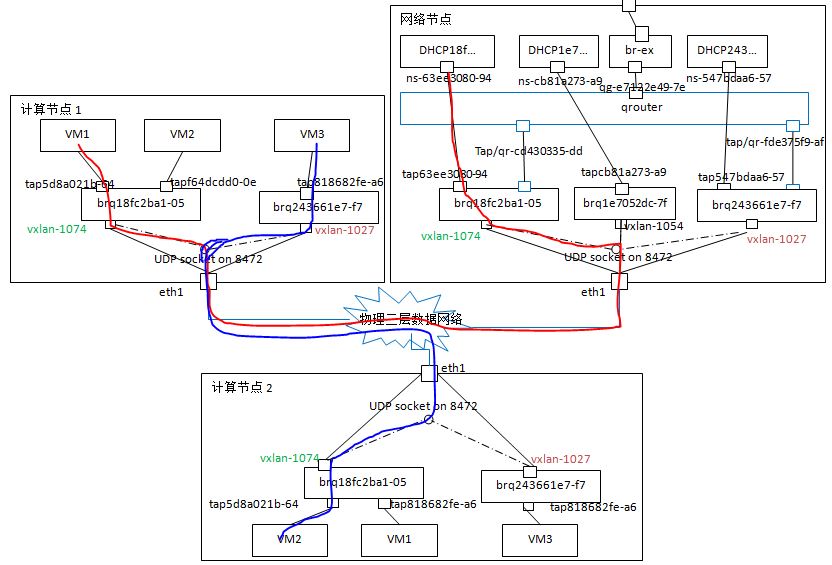
[复制代码](javascript:void(0);)

 （5）vxlan 在 8472 端口上监听 UDP 请求

 （6）vxlan interface 的广播网卡是 eth1

**3.2.3 网络流量走向**

注：红色线条为计算节点1上的vm3访问 DHCP Agent 来获取 IP 地址；蓝色线条为同一个网段的 VM3 和 VM2 互访



**4. vxlan + linux bridge 组网的实现**

**4.1 vxlan interface**

**4.1.1 创建 vxlan interface**

Neutron Linux bridge agent 使用 "ip link add type vxlan" 命令来创建 vxlan interface。这是该命令的帮助信息：

[复制代码](javascript:void(0);)

Command: ['ip link add type vxlan help']

Usage: ... vxlan id VNI [ { group | remote } ADDR ] [ local ADDR ]

[ ttl TTL ] [ tos TOS ] [ dev PHYS\_DEV ]

[ port MIN MAX ] [ [no]learning ]

[ [no]proxy ] [ [no]rsc ]

[ [no]l2miss ] [ [no]l3miss ]

Where: VNI := 0-16777215 #network 的 segemntation\_id

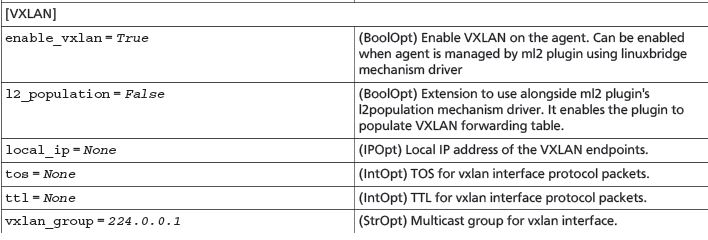
ADDR := { IP\_ADDRESS | any } #IP 地址

TOS := { NUMBER | inherit } #可以由 tos 配置项指定

TTL := { 1..255 | inherit } #可以由 ttl 配置项指定，不设置时默认为1

[复制代码](javascript:void(0);)

以 vxlan id 1027 group 224.0.0.1 dev eth1 port 32768 61000 ageing 300 为例，它使用如下的配置项：



（1）id 1027，这是 vxlan interface id，其值为 neutron network 的 provider:segmentation\_id 属性。

（2）group 224.0.0.1。表示这个是一个多播 VXLAN interface，使用 Neutron 默认的多播组 224.0.0.1，你可以使用 vxlan\_group 配置项指定其它的多播组。需要注意的是：

（a）每个 Neutron linux bridge agent 只能配置一个 vxlan\_group 地址，这意味着只能加入一个多播组。

（b）在需要广播时使用多播组地址为目的地址，比如：

17:01:28.524935 IP (tos 0x0, ttl 1, id 45252, offset 0, flags [none], proto UDP (17), length 78)

10.0.0.10.49332 > 224.0.0.1.8472: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 1074

ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 70.0.0.105 tell 70.0.0.100, length 28

（c）Neutron linux bridge agent 会优先使用单播（ucast），只有在单播不能使用的时候才会尝试多播（mcast），而该配置项只有在使用多播时候才有效。看代码：

检查是使用单播还是多播：

if self.vxlan\_ucast\_supported():

self.vxlan\_mode = lconst.VXLAN\_UCAST #使用单播

elif self.vxlan\_mcast\_supported():

self.vxlan\_mode = lconst.VXLAN\_MCAST #使用多播

else:

raise exceptions.VxlanNetworkUnsupported()

而 vxlan\_group 配置项只有在 MCAST 时候才生效：

if self.vxlan\_mode == lconst.VXLAN\_MCAST:

args['group'] = cfg.CONF.VXLAN.vxlan\_group

而使用单播的条件比较苛刻（可以看代码中的详细条件），基本上（1）如果没有配置 l2population，则肯定不使用单播 （2）如果配置了l2population，支持 iproute2，vxlan 支持 proxy 等等一系列检查，都合格是才使用单播。而使用单播时，vxlan interface 没有 group 属性，比如：

vxlan id 1074 dev eth1 port 32768 61000 proxy ageing 300

需要注意的是，目前的代码只检查 vxlan interface 的名称（name）而不检查具体的属性包括 group 等，如果你变换 l2population 的配置，导致需要切换使用单播和多播，则需要手工删除已有的 vxlan interface，然后重启 neutron linux bridge agent 让它重建 vxlan interface，否认， 你的租户网络可能会不通。

（3）dev eth1 是 UDP 出去的网卡，由配置项 local\_ip 指定。

（4）port 32768 61000 是 linux vxlan 实现的 UDP 源端口号范围，这个不可以配置。

/\* The below used as the min/max for the UDP port range \*/

>> +#define VXLAN\_SRC\_PORT\_MIN 32768

>> +#define VXLAN\_SRC\_PORT\_MAX 61000

在计算该端口的时候，可以考虑每个虚机的特定属性，来实现底层转发网络上多条转发路径上的负载均衡。

[复制代码](javascript:void(0);)

+/\* Compute source port for outgoing packet.

>> + \* Currently we use the flow hash.

>> + \*/

>> +static u16 get\_src\_port(struct sk\_buff \*skb)

>> +{

>> + unsigned int range = (VXLAN\_SRC\_PORT\_MAX - VXLAN\_SRC\_PORT\_MIN) + 1;

>> + u32 hash = OVS\_CB(skb)->flow->hash;

>> +

>> + return (\_\_force u16)(((u64) hash \* range) >> 32) + VXLAN\_SRC\_PORT\_MIN;

>> +}

[复制代码](javascript:void(0);)

（5）ageing 300，这是 unreachable-vtep-aging-timer，单位是秒。其含义是，经过学习得到的远端 VTEP 后面的虚机的 MAC 地址过期时长。

（6）目前 Neutron linux bridge agent 无法配置 VXLAN UDP 端口，只能使用linux的默认端口，具体见这个 ticket：[[Juno]: Cannot set the VXLAN UDP destination port to 4789 using Linux Bridge](https://bugs.launchpad.net/neutron/+bug/1483853)。到目前为止其状态依然是 “In progress”。一般来讲，linux 默认使用的端口号是 8472。修改配置文件 /etc/modprobe.d/vxlan-port.conf 来向 vxlan 内核模块传递端口参数可以修改该端口号，比如如下的配置会使得系统改为使用 4789 端口：

cat /etc/modprobe.d/vxlan-port.conf

options vxlan udp\_port=4789

或者在创建 vxlan interface 时指定 dstport 参数 （这是 Neutron linux bridge fix 的做法）。创建好以后，需要设置其 IP 地址或者加入一个bridge。Neutron 的方案是将它加入到一个 linux bridge 上。然后，将其设置为 up （使用 ip link set up 命令）。

**4.1.2 VXLAN interface 的功能**

该 interface 会：

（1）创建并连接到一个 UDP 端口 8472 的 socket。IANA 标准化组织规定的端口是 4789，而 Linux 内核为了后向兼容需要而使用的端口是 8472.

/\* UDP port for VXLAN traffic.

62 \* The IANA assigned port is 4789, but the Linux default is 8472

63 \* for compatibility with early adopters.

64 \*/

err = sock\_create(AF\_INET, SOCK\_DGRAM, 0, &vxlan\_port->vxlan\_rcv\_socket);

kernel\_bind(vxlan\_port->vxlan\_rcv\_socket, (struct sockaddr \*)&sin, sizeof(struct sockaddr\_in));

udp\_sk(vxlan\_port->vxlan\_rcv\_socket->sk)->encap\_rcv = vxlan\_rcv;

（2）在支持多播的情况下，加入一个多播组。可使用 netstat -g 命令查看多播组成员。

（3）虚机经过 linux bridge 由 vxlan 出去的流量，由 vxlan interface 封装 VXLAN 头，然后使用 UDP 由指定网卡发出

[复制代码](javascript:void(0);)

src\_port = udp\_flow\_src\_port(net, skb, 0, 0, true); #计算源 UDP 端口

md.vni = htonl(be64\_to\_cpu(tun\_key->tun\_id) << 8);

md.gbp = vxlan\_ext\_gbp(skb);

vxflags = vxlan\_port->exts |(tun\_key->tun\_flags & TUNNEL\_CSUM ? VXLAN\_F\_UDP\_CSUM : 0);

err = vxlan\_xmit\_skb(rt, sk, skb, fl.saddr, tun\_key->ipv4\_dst, #经过 UDP socket 发出

tun\_key->ipv4\_tos, tun\_key->ipv4\_ttl, df, src\_port, dst\_port, &md, false, vxflags);

[复制代码](javascript:void(0);)

（4）进来的 vxlan 流量，首先到达 UDP 端口，再交给 vxlan 钩子函数，由 vxlan 做解包处理后，经过 vxlan interface 通过 linux bridge 转发给虚机

/\* Called with rcu\_read\_lock and BH disabled. \*/

>> +static int vxlan\_rcv(struct sock \*sk, struct sk\_buff \*skb)

注：

（1）以上示例代码都是 OVS 对 vxlan 的实现，linux 内核的实现原理其实也差不多。

（2）其实更准确地说以上这些都是Linux vxlan 内核模块的功能，只不过都是通过 vxlan interface 来体现给用户的。

**4.1.3 不使用 l2population 情况下的 Linux vxlan interface 的 FDB 表（forwarding table）**

   在不使用 l2population 的情况下，VXLAN 通过多播学习来得到 fdb 表项即（VM-MAC, VTEP-IP）。这个成本是蛮高的。你可以使用 bridge 命令来查看 FDB，该命令由 iproute2 包提供。我们通过下面的实验来观察该过程：

（1）vxlan interface 被创建后，fdb 只有一个表项，就是所有的流量都发往多播组

root@compute1:~# bridge fdb show dev vxlan-1074

c6:a5:bf:5b:67:a3 vlan 0 permanent

00:00:00:00:00:00 dst 224.0.0.1 via eth1 self permanent

（2）连接该 vxlan interface 的 vm1 先获得ip 地址，然后 ping 另一个网段上的vm2

[复制代码](javascript:void(0);)

root@compute1:~# bridge fdb show dev vxlan-1074

fa:16:3e:19:15:fe vlan 0

fa:16:3e:32:35:ef vlan 0

c6:a5:bf:5b:67:a3 vlan 0 permanent

00:00:00:00:00:00 dst 224.0.0.1 via eth1 self permanent

fa:16:3e:32:35:ef dst 10.0.0.10 self

fa:16:3e:19:15:fe dst 10.0.0.10 self

[复制代码](javascript:void(0);)

可见，这两个过程中，vxlan VTETP 学习到了两个地址：qdhcp 的一个端口和 qrouter 的一个端口

（3）vm1 ping vm3, vm3 在 vm1 同网段，但是在不同的机器上

[复制代码](javascript:void(0);)

root@compute1:~# bridge fdb show dev vxlan-1074

fa:16:3e:c4:c8:58 vlan 0

fa:16:3e:19:15:fe vlan 0

fa:16:3e:32:35:ef vlan 0

c6:a5:bf:5b:67:a3 vlan 0 permanent

00:00:00:00:00:00 dst 224.0.0.1 via eth1 self permanent

fa:16:3e:32:35:ef dst 10.0.0.10 self

fa:16:3e:c4:c8:58 dst 10.0.0.14 self

fa:16:3e:19:15:fe dst 10.0.0.10 self

[复制代码](javascript:void(0);)

可见它学习到了另一个主机上的VTEP 信息。

（5）我们也可以看到，这些表项都不是 permanent 的，因此，一定时间后，它们就会因过期被删除。vxlan interface 的 aging 属性值决定，默认为5分钟。因此，五分钟后，这些学到的表项都会过期了，重新回到状态（1），开始新的一轮学习过程。

c6:a5:bf:5b:67:a3 vlan 0 permanent

00:00:00:00:00:00 dst 224.0.0.1 via eth1 self permanent

（6）需要注意的是，ip n 表在整个过程中都没有变化，因为这时候，vxlan 不会承担 ARP Proxy 的任务。

详细情况，可以参考 [Neutron 理解 (3): Open vSwitch + GRE/VxLAN 组网 [Netruon Open vSwitch + GRE/VxLAN Virutal Network]](http://www.cnblogs.com/sammyliu/p/4627230.html) 中的 VTEP 学习部分。

**4.2 使用 Neutorn l2population - fdb 更新**

fdb 表通过帮助帮助解决两个问题来使得 Linux VTEP 可以使用单播而不是需要使用多播：

（1）获得远端 VM IP 和 MAC 地址的映射关系，从而本地虚机不需要使用 ARP 广播来获取该地址

（2）获得远端 VM MAC 和 它的 VTEP IP 的 映射关系，从而使得本地 VTEP 不需要通过多播来获取该地址

普通情况下，这些映射关系都是 VTEP 使用多播通过地址学习获得的。

**4.2.1 Linux IP neigh 表**

    Linux bridge agent 在使用 l2population 的情况下，使用该表来保存远端 VM IP 和 MAC 地址的映射关系。要使用 l2population，需要在配置文件 /etc/neutron/plugins/ml2/ml2\_conf.ini 中做如下配置：

[复制代码](javascript:void(0);)

[ml2]

mechanism\_drivers = linuxbridge,l2population

[vxlan]

l2\_population = true

[复制代码](javascript:void(0);)

在l2pop 生效后，并且启用了 arp\_reponder 之后，新建的 vxlan interface 上就会增加 “proxy” 功能：

root@controller:~# ip -d link show dev vxlan-1027

20: vxlan-1027: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master brq243661e7-f7 state UNKNOWN mode DEFAULT group default

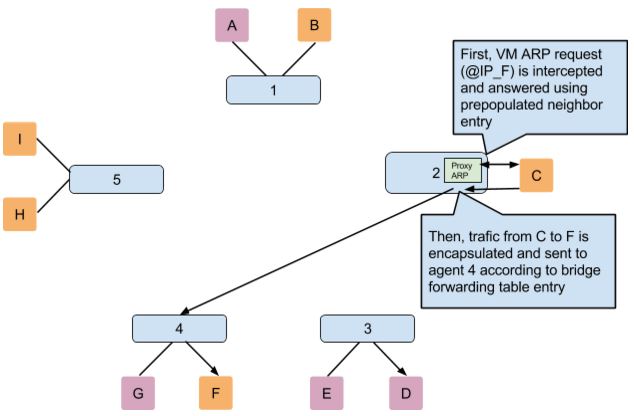
link/ether d6:40:1e:47:38:59 brd ff:ff:ff:ff:ff:ff promiscuity 1

vxlan id 1027 dev eth1 port 32768 61000 proxy ageing 300

这使得 vxlan interface 能够通过查询 ip neighbour table 来响应本地虚机对远端虚机的 ARP 请求，从而作为这些虚机的 ARP Proxy。

关于l2pop的实现原理，可参考 [Neutron 理解 (4): Neutron OVS OpenFlow 流表 和 L2 Population [Netruon OVS OpenFlow tables + L2 Population]](http://www.cnblogs.com/sammyliu/p/4633814.html)。

VXLAN Interface 实现 Proxy ARP 的原理：



linux bridge agent 的实现采用的是直接调用 ip neighbour 命令行来操作 fdb 表。该命令同样是 iproute2 包提供的命令之一。

def add\_fdb\_ip\_entry(self, mac, ip, interface):

ip\_lib.IPDevice(interface).neigh.add(ip, mac)

def remove\_fdb\_ip\_entry(self, mac, ip, interface):

ip\_lib.IPDevice(interface).neigh.delete(ip, mac)

格式：ip neighbor add *REMOTE\_VM\_IP* lladdr *REMOTE\_VM\_MAC* dev vx-NET\_ID nud permanent。注意这些entry 是 PERMANENT 即永久的，它们不受 vxlan aging 时长的限制。

以 vxlan-1074 interface示例：

root@compute1:~# ip n | grep vxlan-1074  
70.0.0.129 dev vxlan-1074 lladdr fa:16:3e:c4:c8:58 PERMANENT #在计算节点2上的虚机  
70.0.0.100 dev vxlan-1074 lladdr fa:16:3e:32:35:ef PERMANENT #qdhcp 的 ns-63ee3080-94 interface  
70.0.0.1 dev vxlan-1074 lladdr fa:16:3e:19:15:fe PERMANENT   #qrouter 的 qr-cd430335-dd interface

这样的话，vxlan interface 就可以直接向虚机的 ARP 广播请求提供 ARP 响应了。但是，这里只有远端虚机和 qroute/qdhcp Interface 的条目。对于本机上的同网络虚机，看来 Neutron 是放任 linux bridge 去做广播了，反正范围也很小。可见，这里的目的只是为了减少对外的多播。

再看看 VTEP 的 MAC 地址是如何保存的：

root@compute1:~# ip n | grep 10.0

10.0.0.10 dev eth1 lladdr 52:54:00:7c:c0:79 STALE

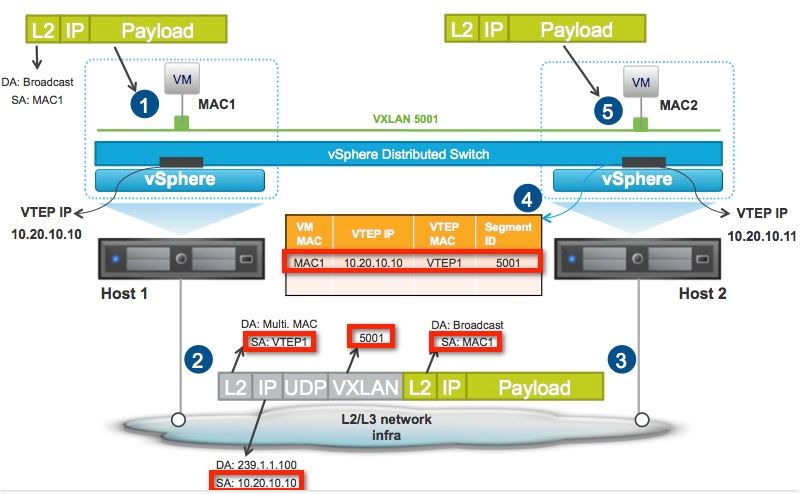
10.0.0.14 dev eth1 lladdr 52:54:00:c6:4d:42 STALE

10.0.0.100 dev eth1 lladdr fa:80:13:21:6b:56 STALE

它们的来源还待查。

**4.2.2 vxlan interface 所连接的 linux bridge 的 fdb 表**

linux 实现的这个表和 vSphere 中的 fdb 表是对应的（除了 VTEP MAC 保存在主机的 ip neigh 表中）：



这些表保存 远端 VM MAC （mac）- 远端 VTEP IP（dst） - VXLAN ID （dev） 的关系。linux-linuxbridge-agent 实现了通过 “bridge” 命令来操作 fdb 表项的各个方法（代码[在这里](https://github.com/openstack/neutron/blob/master/neutron/plugins/ml2/drivers/linuxbridge/agent/linuxbridge_neutron_agent.py)）：

[复制代码](javascript:void(0);)

def add\_fdb\_bridge\_entry(self, mac, agent\_ip, interface, operation="add"):

utils.execute(['bridge', 'fdb', operation, mac, 'dev', interface, 'dst', agent\_ip],

run\_as\_root=True, check\_exit\_code=False)

def remove\_fdb\_bridge\_entry(self, mac, agent\_ip, interface):

utils.execute(['bridge', 'fdb', 'del', mac, 'dev', interface, 'dst', agent\_ip],

run\_as\_root=True, check\_exit\_code=False)

def add\_fdb\_entries(self, agent\_ip, ports, interface):

for mac, ip in ports:

if mac != constants.FLOODING\_ENTRY[0]:

self.add\_fdb\_ip\_entry(mac, ip, interface)

self.add\_fdb\_bridge\_entry(mac, agent\_ip, interface,

operation="replace")

elif self.vxlan\_mode == lconst.VXLAN\_UCAST:

if self.fdb\_bridge\_entry\_exists(mac, interface):

self.add\_fdb\_bridge\_entry(mac, agent\_ip, interface,

"append")

else:

self.add\_fdb\_bridge\_entry(mac, agent\_ip, interface)

def remove\_fdb\_entries(self, agent\_ip, ports, interface):

for mac, ip in ports:

if mac != constants.FLOODING\_ENTRY[0]:

self.remove\_fdb\_ip\_entry(mac, ip, interface)

self.remove\_fdb\_bridge\_entry(mac, agent\_ip, interface)

elif self.vxlan\_mode == lconst.VXLAN\_UCAST:

self.remove\_fdb\_bridge\_entry(mac, agent\_ip, interface)

[复制代码](javascript:void(0);)

fdb 条目格式：bridge fdb add *REMOTE\_VM\_MAC* dev *vx-NET\_ID* dst *REMOTE\_HOST\_IP*

比如：

[复制代码](javascript:void(0);)

root@compute1:~# bridge fdb show dev vxlan-1074  
fa:16:3e:19:15:fe vlan 0  
b6:dc:2f:dd:8b:81 vlan 0 permanent  
00:00:00:00:00:00 dst 10.0.0.10 self permanent #需要多播或者广播时，比如一个虚机的MAC地址在 fdb 表不存在的时候，需要使用多次单播来模拟多播  
00:00:00:00:00:00 dst 10.0.0.14 self permanent #多播或者广播目标之二。在不使用多播的情况下，如果需要多播的功能，则通过多次发送单播的方式来模拟多播  
fa:16:3e:32:35:ef dst 10.0.0.10 self permanent # qdhcp 的 ns-63ee3080-94 interface  
fa:16:3e:c4:c8:58 dst 10.0.0.14 self permanent # 在另一个计算节点上的同网络的虚机  
fa:16:3e:19:15:fe dst 10.0.0.10 self permanent # qrouter 的 qr-cd430335-dd interface

[复制代码](javascript:void(0);)

这样的话，VTEP 就不需要通过多播来获取目的虚机的 VTEP 的 IP 地址了。

**4.2.3 问题调试一例：虚机无法获取固定IP**

问题定位步骤如下：

（1）在 tap 设备上做 tcpdump，能看到BOOTP 请求发出，但是没用响应

（2）在 vxlan interface 设备上做 tcpdump，看不到包发出

（3）查看该设备的 fdb 表，只有一条记录

root@compute1:~# bridge fdb show dev vxlan-1074

3e:c0:e5:74:f7:49 vlan 0 permanent

  可见，这时候 vxlan 是无法通过 UDP 将包发出去的，因为没用单播或者多播 fdb 表表项。

（4）重启 Neutron linux bridge agent，在查看 fdb 表

[复制代码](javascript:void(0);)

root@compute1:~# bridge fdb show dev vxlan-1074

3e:c0:e5:74:f7:49 vlan 0 permanent

00:00:00:00:00:00 dst 10.0.0.14 self permanent

00:00:00:00:00:00 dst 10.0.0.10 self permanent

fa:16:3e:32:35:ef dst 10.0.0.10 self permanent

fa:16:3e:c4:c8:58 dst 10.0.0.14 self permanent

fa:16:3e:19:15:fe dst 10.0.0.10 self permanent

[复制代码](javascript:void(0);)

   可见此时表项都正常了

（5）重新在虚机内运行 ifup eth0，正常获取固定IP

   该问题同时也说明，如果 l2population 功能不正常的话，虚机的网络可能会断；如果使用多播，这种情况应该会避免。

更深入地看一下到底是什么原因：

（1）对 linux bridge 来说，它自身有个 fdb。在转发之前，它会查这个表。如果有查到一条记录，那么就将二层帧转发到该记录对应的 bridge port；如果找不到，它就会泛洪。也就是说，linux bridge 是无论如何都会将它收到的帧转发出去的。

（2） vxlan interface 也有自己的 fdb，它由 vxlan driver 来维护。当 linux bridge 将帧发到 vxlan interface 之后，vxlan driver 在将封包发给 udp 协议栈之前，它需要查 fdb 表，从中找出目的 MAC 地址对应的 fdb 条目中的对方 VTEP 的 IP 地址。vxlan.c 中的相关代码如下：

[复制代码](javascript:void(0);)

f = vxlan\_find\_mac(vxlan, eth->h\_dest);

did\_rsc = false;

...if (f == NULL) {

f = vxlan\_find\_mac(vxlan, all\_zeros\_mac);

if (f == NULL) {

if ((vxlan->flags & VXLAN\_F\_L2MISS) &&

!is\_multicast\_ether\_addr(eth->h\_dest))

vxlan\_fdb\_miss(vxlan, eth->h\_dest);

**dev->stats.tx\_dropped++;**

kfree\_skb(skb);

return NETDEV\_TX\_OK;

}

}

[复制代码](javascript:void(0);)

这代码说明：（1）首先根据 目的 MAC 地址查表 （2）如果查不到，则根据全0的 MAC查表 （3）再查不到，则丢弃该帧。

**4.3 使用多播的一些考量**

   在不使用单播的情况下，Linux vxlan 需要使用多播。而在选择多播地址上，还是有一些讲究：

* 尽量不要使用 224.0.0.1 到 224.0.0.255 区间内的地址，这些地址往往有一些约定俗成的用途，比如用于多播路由
* 公司内网使用的话，尽量选择 239.0.0.0 到 239.255.255.255 区间内的地址，这些地址就像 10.0.0.0/8 地址一样，都是常用的公司内部地址

    Neutron Linux bridge agent的实现中，默认情况下，所有的计算和网络节点上的 VTEP 都必须在同一个多播组中，这个组的地址可以配置，不配置的话使用默认的组 224.0.0.1。需要注意的是，224.0.0.1 是 “The *All Hosts* multicast group addresses all hosts on the same network segment.”，因此，它的成员只能在一个物理网段内，也就是说，如果各节点跨网段的话，需要修改多播组的IP。另外，还不清楚 Neutron 是否支持配置多个多播组，如果支持的话，结合 Nova 的 AZ 概念，可以将特定 AZ 内的计算节点加入到一个特定的多播组，而网络节点使用多个AZ的多个多播组，这样将会有利于控制多播组内成员的数量。

一个示例：

加入多播组：

* + 主机1 上的 VTEP 发送 IGMP 加入消息 （join message）去加入多播组
  + 主机4 上的 VTEP 发送 IGMP 加入消息去加入同一个多播组

多播过程：

* 1. 虚机 VM1 发送一个广播帧
  2. 主机1 上的VM1所在网络对应的 VTEP 将该广播帧封装成 UDP 多播包，设置其目的 IP 地址为 VTEP interface 多播组的地址。
  3. 物理网络将其发到该多播组内的所有主机
  4. 主机4 上的 VTEP，收到该 IP 包后，检查其 VXLAN ID，发现其上有同样 VXLAN ID 的 vxlan interface，则将该包解包后由它通过 linux bridge 转发给该虚机。
  5. 主机2 和 3 同样收到该包，因为它们也在在多播组内，然而，它们发现没有带 VXLAN ID 的 vxlan interface，因此将包丢弃。

以上是理论部分，具体还要进一步的实践。TBD。

**4.4 Neutron 基于 Linux bridge 的防火墙 IptablesFirewallDriver**

github 上的源代码[在这里](https://github.com/openstack/neutron/blob/master/neutron/agent/linux/iptables_firewall.py)。支持 ipset。使用 iptables，作用在各个 linux bridge 上。

在 /etc/neutron/plugins/ml2/ml2\_conf.ini 中的配置：

[securitygroup]

# enable\_ipset = True

enable\_security\_group = True

enable\_ipset = True

firewall\_driver = neutron.agent.linux.iptables\_firewall.IptablesFirewallDriver

从实现上看，OVSHybridIptablesFirewallDriver 类是从IptablesFirewallDriver 继承过来的，只有小幅的改动。因此，可以参考 [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)，不同的部分的细节待将来再进一步的分析。

**5. Linux bridge agent 源代码分析**

github 上的源代码[在这里](https://github.com/openstack/neutron/blob/master/neutron/plugins/ml2/drivers/linuxbridge/agent/linuxbridge_neutron_agent.py)。其代码逻辑并不复杂，主要实现了如下的逻辑：

（1）main（）函数。在启动 neutron-plugin-linuxbridge-agent 服务时被调用，它初始化一个 LinuxBridgeNeutronAgentRPC 实例，并调用其 start 方法启动它。

（2）LinuxBridgeNeutronAgentRPC 类是主类，它初始化SecurityGroupAgentRpc 类的实例用来处理安全组；设置并启动该Rpc (setup\_rpc)；启动一个循环来不断地 scan tap devices （scan\_devices）来获取待处理的 tap 设备，并在设备列表有变化时进行相应的处理（包括 add 或者 remove interface）

（3）对于 Rpc，LinuxBridgeNeutronAgentRPC  会启动一个 LinuxBridgeRpcCallbacks 实例，默认情况下，会处理 PORT\_UPDATE，NETWORK\_DELETE 和 SG\_UPDATE RPC 消息；在设置了 l2pop 的情况下，会增加处理 L2POP\_UPDATE 消息；并且还会启动一个循环来不断调用 \_report\_state 来想 neutron server 报告其状态。

consumers = [[topics.PORT, topics.UPDATE],[topics.NETWORK, topics.DELETE],[topics.SECURITY\_GROUP, topics.UPDATE]]

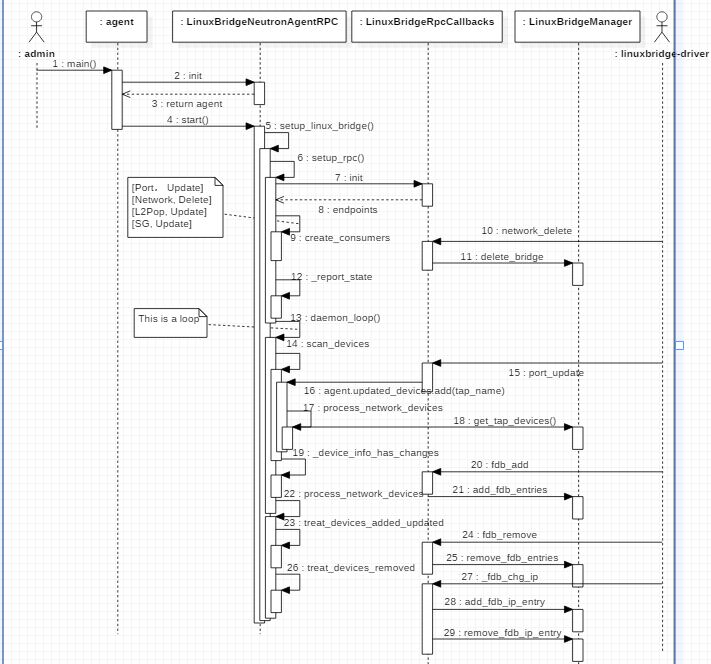
if cfg.CONF.VXLAN.l2\_population:

consumers.append([topics.L2POPULATION, topics.UPDATE])

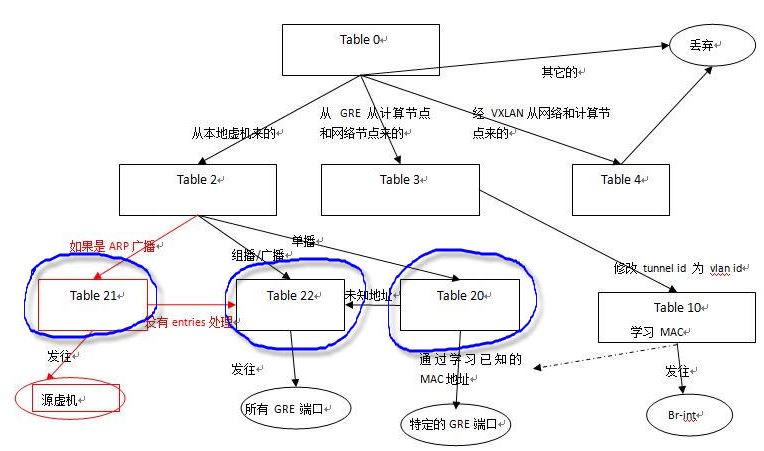
（4）RPC 处理

* network\_delete：在 network 被删除后，将相应的 linux bridge 删除
* port\_update：将 port 的 tap 名称放到 updated\_devices 中，待进一步处理
* fdb\_add：增加 fdb 表项
* fdb\_remove：删除 fdb 表项
* \_fdb\_chg\_ip：修改 ip neighbor fdb 表项
* 直接使用 sg\_rpc.SecurityGroupAgentRpcCallbackMixin 的方法来处理安全组通知消息

画了一个大致的流程图：



**6. 与 OVS 的对比**



* OVS 通过发往多个隧道端口的方式来模拟多播/广播
* OVS 2.1 及以后版本自带 ARP Responder 功能
* 没有 l2pop 时，计算节点之间以及计算节点与网络节点之间都会建立隧道
* 有 l2pop 时，只有有虚机的计算节点和网络节点之间，以及有虚机在同一个网络的计算节点之间建立隧道

详细内容，可参考 [Neutron 理解 (4): Neutron OVS OpenFlow 流表 和 L2 Population](http://www.cnblogs.com/sammyliu/p/4633814.html)

**7. VXLAN Hardware Offload**

  现在，越来越多的网卡设备支持 offload 特性，来提升网络收/发性能。offload 是将本来该操作系统进行的一些数据包处理（如分片、重组等）放到网卡硬件中去做，降低系统 CPU 消耗的同时，提高处理的性能。包括 LSO/LRO、GSO/GRO、TSO/UFO 等。

  诸于 VXLAN 的网络虚拟化技术给服务器的CPU带来了额外的负担，比如封包、解包和校验等。因此，一些中高端网卡已经添加了 Hardware Offload 的能力，将原本需要服务器CPU处理的事情交给网卡自己来处理。

**7.1 常见的 offload 技术**

**LSO/LRO：Large Segment Offload 和 Large Receive Offload**

分别对应到发送和接收两个方向

首先来看 LSO。我们知道计算机网络上传输的数据基本单位是离散的网包，既然是网包，就有大小限制，这个限制就是 MTU（Maximum Transmission Unit）的大小，一般是1518字节。比如我们想发送很多数据出去，经过os协议栈的时候，会自动帮你拆分成几个不超过MTU的网包。然而，这个拆分是比较费计算资源的（比如很多时候还要计算分别的checksum），由 CPU 来做的话，往往会造成使用率过高。那可不可以把这些简单重复的操作 offload 到网卡上呢？

于是就有了 LSO，在发送数据超过 MTU 限制的时候（太容易发生了），OS 只需要提交一次传输请求给网卡，网卡会自动的把数据拿过来，然后进行切，并封包发出，发出的网包不超过 MTU 限制。

接下来看 LRO，当网卡收到很多碎片包的时候，LRO 可以辅助自动组合成一段较大的数据，一次性提交给 OS处理。

一般的，LSO 和 LRO 主要面向 TCP 报文。

**GSO/GRO： Generic Segmentation Offload 和 Generic Receive Offload**

分别比 LSO 和 LRO 更通用，自动检测网卡支持特性，支持分包则直接发给网卡，否则先分包后发给网卡。新的驱动一般用 GSO/GRO。

GRO 是在内核 2.6.29 之后合并进去的，作者是一个华裔Herbert Xu ,GRO的简介可以看这里：<http://lwn.net/Articles/358910/>

GSO 是在内核 Linux 2.6.18 之后合并进去的。

先来描述一下GRO的作用，GRO 是针对网络接受包的处理的，并且只是针对 NAPI 类型的驱动，因此如果要支持 GRO，不仅要内核支持，而且驱动也必须调用相应的接口，用 ethtool -K gro on 来设置，如果报错就说明网卡驱动本身就不支持GRO。

GRO 类似 TSO，可是 TSO 只支持发送数据包，这样你 tcp 层大的段会在网卡被切包，然后再传递给对端，而如果没有gro，则小的段会被一个个送到协议栈，有了gro之后，就会在接收端做一个反向的操作(想对于tso).也就是将tso切好的数据包组合成大包再传递给协议栈。

详细信息及代码分析，请阅读 [linux kernel 网络协议栈之GRO(Generic receive offload)](http://www.pagefault.info/?p=159)

**TSO/UFO：TCP Segmentation Offload 和 UDP fragmentation offload**

分别对应 TCP 报文和 UDP 报文。

TSO 将 TCP 协议的一些处理下放到网卡完成以减轻协议栈处理占用 CPU 的负载。通常以太网的 MTU 是1500Bytes，除去 IP 头（标准情况下20Bytes）、TCP头（标准情况下20Bytes），TCP的MSS (Max Segment Size)大小是1460Bytes。当应用层下发的数据超过 MSS 时，协议栈会对这样的 payload 进行分片，保证生成的报文长度不超过MTU的大小。但是对于支持 TSO/GSO 的网卡而言，就没这个必要了，可以把最多 64K 大小的 payload 直接往下传给协议栈，此时 IP 层也不会进行分片，一直会传给网卡驱动，支持TSO/GSO的网卡会自己生成TCP/IP包头和帧头，这样可以offload很多协议栈上的内存操作，checksum计算等原本靠CPU来做的工作都移给了网卡。

VMware 产品中实现了 TSO 和 LRO 来提高网络性能。

**RSS：Receive Side Scaling**

多核服务器中的网卡还推荐考虑 RSS，将网流分配到多个 RSS 队列上，多个队列绑定到不同的核心上，分散负载。

|  |  |  |
| --- | --- | --- |
| 发送端 | 接收端 | 注释 |
| LSO/TSO/UFO | LRO | LRO 是 linux 2.6.24 中引入的。 |
| GSO | GRO | GRO 是 linux 2.6.29 中引入的。  GSO 和 GRO 更加通用，G\*O 是对 T\*O 的增强，新的驱动都是用G\*O。  注意两个可以同时使用，所以要禁止offload的话需要检查是否都被禁止了 |

**7.2 网卡offload 能力检测、打开和关闭**

 检查你的网卡是否具备Offload 能力：

root@hkg02kvm001ccz023:~# ethtool -k eth0 | grep tx-udp

tx-udp\_tnl-segmentation: off [fixed] （悲剧的是我的网卡不具备）

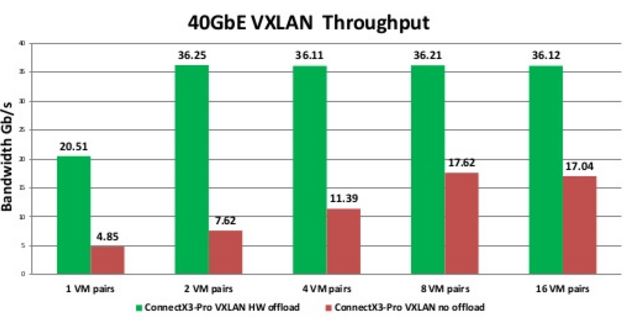
你可以打开或者关闭该功能：

ethtool -K ethX tx-udp\_tnl-segmentation [off|on]

**7.3 Offload 带来的网络性能优化**

而从一些网卡提供商的报告中可以看出来Offload 带来的好处 （[资料来源](http://www.slideshare.net/mellanox/vxlan-vm-world-02-theatre)）：

（1）带宽提高了



（2）服务器CPU压力下降了



参考

<http://www.slideshare.net/JamesDenton1/2014-openstack-summit-neutron-ovs-to-linuxbridge-migration>

<http://blogs.vmware.com/vsphere/2013/05/vxlan-series-how-vtep-learns-and-creates-forwarding-table-part-5.html>

<https://blogs.vmware.com/vsphere/2013/05/vxlan-series-multicast-basics-part-2.html>

<http://blogs.vmware.com/vsphere/2013/05/vxlan-series-multicast-usage-in-vxlan-part-3.html>

<https://github.com/torvalds/linux/blob/master/drivers/net/vxlan.c>

<https://software.intel.com/en-us/blogs/2015/01/29/optimizing-the-virtual-networks-with-vxlan-overlay-offloading>

<https://docs.google.com/document/d/1sUrvOQ9GIl9IWMGg3qbx2mX0DdXvMiyvCw2Lm6snaWQ/edit>

<http://www.enterprisenetworkingplanet.com/netsp/article.php/10953_3623181_2/Networking-101--Understanding-Multicast-Routing.htm>

<http://www.cs.virginia.edu/~cs458/slides/module21-mcast.pdf>

<http://www.juniper.net/documentation/en_US/junos13.2/topics/concept/multicast-ip-overview.html>

<http://blog.csdn.net/yeasy/article/details/19204639>

<http://seitran.com/2015/04/13/01-gso-gro-lro/>

分类: [Neutron](http://www.cnblogs.com/sammyliu/category/677824.html),[基础知识](http://www.cnblogs.com/sammyliu/category/692759.html),[原理](http://www.cnblogs.com/sammyliu/category/750167.html)