基本网络

« (intro.html) » (intro-network-components.html) (https://bugs.launchpad.net/neutron/+filebug?field.title=Basic%20networking%20in%20Neutron&field.comment=%0A%0A%0AThis bug tracker is for errors with the documentation, use the following as a template and remove or add fields as you see fit. Convert [ ] into [x] to check boxes:%0A%0A- [ ] This doc is inaccurate in this way: _____%0A- [ ] This is a doc addition request.%0A- [ ] I have a fix to the document that I can paste below including example: input and output. %0A%0AIf you have a troubleshooting or support issue, use the following resources:%0A%0A - Ask OpenStack: http://ask.openstack.org%0A - The mailing list: http://lists.openstack.org%0A - IRC: 'openstack' channel on Freenode%0A%0A----------%0ARelease:%2012.0.1.dev11%20on%202018-03-07%2021:05%0ASHA:%2043df2709acbdce86686a40b75fd34e96880427d0%0ASource:%20https://git.openstack.org/cgit/openstack/neutron/tree/doc/source/admin/intro-basic-networking.rst%0AURL: https://docs.openstack.org/neutron/queens/admin/intro-basic-networking.html&field.tags=doc)

更新日期：2018-03-07 21:05

以太网¶

以太网是一种网络协议，由IEEE 802.3标准规定。大多数有线网络接口卡（NIC）使用以太网进行通信。

在网络协议的OSI模型 (https://en.wikipedia.org/wiki/OSI_model)中，以太网占据了第二层，即所谓的数据链路层。在讨论以太网，你经常会听到诸如 *本地网络*，*层2*，*L2*，*链路层*和*数据链路层*。

在以太网中，连接到网络的主机通过交换*帧进行*通信。以太网上的每台主机都被称为媒体访问控制（MAC）地址的地址唯一标识。特别是，OpenStack环境中的每个虚拟机实例都有唯一的MAC地址，这与计算主机的MAC地址不同。MAC地址有48位，通常表示为十六进制字符串，例如**08:00:27:b9:88:74**。虽然现代NIC允许您以编程方式更改MAC地址，但MAC地址由制造商硬编码到NIC中。在Linux中，您可以使用**ip**命令检索NIC的MAC地址：

```
$ ip link show eth0
 2: eth0: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500 qdisc pfifo_fast state UP模式默认组默认qlen 1000
     link / ether 08: 00: 27: b9: 88: 74 brd ff: ff: ff : FF: FF: FF
```

从概念上讲，您可以将以太网视为每个网络主机连接到的单个总线。在早期的实现中，一个以太网网络由一个单一的同轴电缆组成，主机将接入网络。但是，现代以太网中的网络主机直接连接到称为*交换机*的网络设备 。尽管如此，这个概念模型还是很有用的，在网络图中（包括由OpenStack仪表板生成的），以太网通常被描述为单个总线。您有时会听到以太网称为*第2层网段*。

在以太网中，网络上的每台主机都可以直接向其他主机发送帧。以太网也支持广播，因此一台主机可以通过发送到特殊的MAC地址来向网络中的每台主机发送一帧 **ff:ff:ff:ff:ff:ff**。ARP和DHCP 是使用以太网广播的两个值得注意的协议。由于以太网支持广播，有时您会听到称为*广播域*的以太网。

当NIC收到以太网帧时，默认情况下，NIC会检查目标MAC地址是否与NIC的地址（或广播地址）匹配，并且如果MAC地址不匹配，则丢弃以太网帧。对于计算主机，这种行为是不可取的，因为该帧可能是针对其中一个实例的。即使MAC地址不匹配，也可以将NIC配置为*混合模式*，即将所有以太网帧传送到操作系统。计算主机应始终将适当的NIC配置为混杂模式。

如前所述，现代以太网使用交换机来连接网络主机。交换机是一个网络硬件盒，具有大量端口，可将以太网帧从一台连接的主机转发到另一台主机。当主机首先通过交换机发送帧时，交换机不知道哪个MAC地址与哪个端口相关联。如果以太网帧的目的地是未知的MAC地址，则交换机将该帧广播到所有端口。交换机通过观察流量了解哪些MAC地址位于哪个端口。一旦知道哪个MAC地址与端口相关联，它就可以将以太网帧发送到正确的端口而不是广播。交换机维护MAC地址映射到交换机端口，称为*转发表*或*转发信息库*（FIB）。交换机可以以菊花链方式连接在一起，交换机和主机的连接就像一个网络。

VLAN的¶

VLAN是一种网络技术，使单个交换机可以像多个独立交换机一样工作。具体而言，连接到同一交换机但位于不同VLAN的两台主机不会看到彼此的流量。OpenStack能够利用VLAN隔离不同项目的流量，即使这些项目碰巧在同一个计算主机上运行实例。每个VLAN都有一个相关的数字ID，介于1到4095之间。我们说"VLAN 15"是指数字ID为15的VLAN。

为了理解VLAN如何工作，我们考虑传统IT环境中的VLAN应用，其中物理主机连接到物理交换机，并且不涉及虚拟化。想象一下你想要三个隔离网络的场景，但你只有一个物理交换机。网络管理员将选择三个VLAN ID（例如10,11和12），并将交换机配置为将交换端口与VLAN ID相关联。例如，switchport 2可能与VLAN 10关联，switchport 3可能与VLAN 11关联，等等。当为一个特定的VLAN配置一个交换端口时，它被称为*访问端口*。交换机负责确保VLAN间的网络流量隔离。

现在考虑第一个交换机中的所有交换机端口都被占用的情况，因此组织会购买第二台交换机并将其连接到第一台交换机以扩展可用的交换机端口数量。第二台交换机还配置为支持VLAN ID 10,11和12.现在，想象在为VLAN ID 10配置的端口上连接到交换机1的主机A发送了一个以太网帧，该帧用于连接到交换机2的主机B，该端口配置为VLAN ID 10.当交换机1将以太网帧转发到交换机2时，它必须通知该帧与VLAN ID 10相关联。

如果要将两台交换机连接在一起，并且将交换机配置为使用VLAN，则必须将用于交叉连接交换机的交换机端口配置为允许将来自任何VLAN的以太网帧转发到其他交换机。此外，发送交换机必须在每个以太网帧中标记VLAN ID，以便接收交换机可以确保只有匹配VLAN上的主机才有资格接收该帧。

配置为从所有VLAN传递帧并使用VLAN ID标记它们的交换机端口称为 *中继端口*。IEEE 802.1Q是描述在使用中继时如何在以太网帧中编码VLAN标签的网络标准。

请注意，如果您在物理交换机上使用VLAN来在OpenStack云中实施项目隔离，则必须确保所有交换机端口都配置为中继端口。

选择当前网络基础设施未使用的VLAN范围很重要。例如，如果您估计您的云必须最多支持100个项目，请选择该值以外的VLAN范围，例如VLAN 200-299。OpenStack以及处理项目网络的所有物理网络基础架构都必须支持此VLAN范围。

中继用于连接不同的交换机。每个中继使用标记来标识哪个VLAN正在使用。这可以确保同一VLAN上的交换机可以通信。

子网和ARP¶

虽然NIC使用MAC地址来寻址网络主机，但TCP / IP应用程序使用IP地址。地址解析协议（ARP）通过将IP地址转换为MAC地址来弥合以太网和IP之间的差距。

IP地址分为两部分：*网络号*和*主机标识符*。如果两台主机具有相同的网络号，它们位于同一*子网中*。回想一下，如果两台主机位于同一本地网络上，则只能通过以太网直接进行通信。ARP假定位于同一子网中的所有机器位于同一本地网络上。网络管理员在为主机分配IP地址和网络掩码时必须小心，以使同一子网中的任何两台主机位于同一本地网络上，否则ARP将无法正常工作。

要计算IP地址的网络号码，*您必须*知道 与该地址关联的*网络掩码*。网络掩码表示32位IP地址中有多少位构成网络号。

有两种表达网络掩码的语法：

- 虚线四
- 无类域间路由（CIDR）

考虑一个IP地址192.0.2.5，其中地址的前24位是网络号。以点分四分法表示，网络掩码将写为**255.255.255.0**。CIDR表示法包含IP地址和网络掩码，并且此示例将写为**192.0.2.5/24**。

> **注意**
>
> 在OpenStack环境中不能使用包括多播地址或环回地址的CIDR子网。例如，使用**224.0.0.0/16**或**127.0.1.0/24**不支持创建子网。

有时我们想引用子网，但不是子网上的任何特定IP地址。常见的约定是将主机标识符设置为全零以引用子网。例如，如果主机的IP地址是**192.0.2.24/24**，那么我们会说这个子网是**192.0.2.0/24**。

要了解ARP如何将IP地址转换为MAC地址，请考虑以下示例。假设主机*A*的IP地址为 **192.0.2.5/24**，MAC地址为**fc:99:47:49:d4:a0**，并且想要向IP地址为的 主机*B*发送数据包**192.0.2.7**。需要注意的是网络号是两台主机是相同的，所以主机*一个*是能够直接发送帧到主机*乙*。

主机*A*第一次尝试与主机*B*通信时，目标MAC地址未知。主机*A*向本地网络发出ARP请求。该请求是一个带有这样的消息的广播：

致：大家（ff：ff：ff：ff：ff：ff）。我正在寻找IP地址为192.0.2.7的计算机。签名：MAC地址fc：99：47：49：d4：a0。

主机*B*回应这样的回应：

收件人：fc：99：47：49：d4：a0。我有IP地址192.0.2.7。签名：MAC地址54：78：1a：86：00：a5。

主机*甲*然后发送以太网帧到主机*乙*。

您可以使用**arping**命令手动启动ARP请求。例如，要发送一个ARP请求到IP地址**192.0.2.132**：

```
$ arping -I eth0 192 .0.2.132
 ARPING 192.0.2.132 from 192.0.2.131 eth0
单播回复来自192.0.2.132 [54：78：1A：86：1C：0B] 0.670ms
来自192.0.2.132的单播回复[54:78 ： 1A：86：1C：0B] 0.722ms
单播回复自192.0.2.132 [54：78：1A：86：1C：0B] 0.723ms
发送3个探测器（1个广播）
收到3个回复
```

为了减少ARP请求的数量，操作系统维护一个ARP缓存，其中包含IP地址到MAC地址的映射。在Linux机器上，您可以使用**arp** 命令查看ARP缓存的内容：

```
$ arp -n
 地址HWtype HWaddress标志掩码Iface
192.0.2.3 ether 52：54：00：12：35：03 C eth0
192.0.2.2 ether 52：54：00：12：35：02 C eth0
```

DHCP ¶

连接到网络的主机使用动态主机配置协议（DHCP）动态获取IP地址。DHCP服务器将IP地址分发给作为DHCP客户端的网络主机。

DHCP客户端通过从端口68 发送UDP数据包到端口67上的地址来定位DHCP服务器**255.255.255.255**。地址 **255.255.255.255**是本地网络广播地址：本地网络上的所有主机都可以看到发送到此地址的UDP数据包。但是，这样的数据包不会被转发到其他网络。因此，DHCP服务器必须与客户端在同一本地网络上，否则服务器将不会收到广播。DHCP服务器通过从端口67向客户端上的端口68发送UDP数据包进行响应。交易所看起来像这样：

1. 客户端发送一个discover（“我是MAC地址的客户端 **08:00:27:b9:88:74**，我需要IP地址”）
2. 服务器发送报价（“确定**08:00:27:b9:88:74**，我提供IP地址**192.0.2.112**”）
3. 客户端发送一个请求（“服务器**192.0.2.131**，我想知道IP **192.0.2.112**”）
4. 服务器发送确认（“确定**08:00:27:b9:88:74**，IP **192.0.2.112**是你的”）

OpenStack使用名为dnsmasq (http://www.thekelleys.org.uk/dnsmasq/doc.html)的第三方程序 来实现DHCP服务器。Dnsmasq写入系统日志，您可以在其中观察DHCP请求并回复：

```
年04月 23  15 ： 53 ： 46  C100 - 1  的dhcpd ：  DHCPDISCOVER  从 08 ： 00 ： 27 ： B9 ： 88 ： 74  通过 的eth2
年04月 23  15 ： 53 ： 46  C100 - 1  的dhcpd ：  DHCPOFFER  上 192.0 。2.112  至 08 ： 00 ： 27 ： b9 ： 88 ： 74 经由 的eth2
年04月 23  15 ： 53 ： 48  C100 - 1  的dhcpd ：  DHCPREQUEST  为 192.0 。2.112  （192.0 。2.131 ） 从 08 ： 00 ： 27 ： B9 ： 88 ： 74  通过 的eth2
年04月 23  15 ： 53 ： 48  C100 - 1  的dhcpd ：  DHCPACK  上 192.0 。2.112 到 08 ： 00 ： 27 ： B9 ： 88 ： 74  通过 的eth2
```

在对通过网络无法访问的实例进行故障诊断时，检查此日志以验证DHCP协议的所有四个步骤是否针对相关实例执行会很有帮助。

IP ¶

Internet协议（IP）指定如何在连接到不同本地网络的主机之间路由数据包。IP依靠称为*路由器*或*网关的*特殊网络主机。路由器是连接到至少两个本地网络的主机，可以将IP数据包从一个本地网络转发到另一个本地网络。路由器有多个IP地址：每个网络连接一个网络。

在网络协议的OSI模型中，IP占据了第三层，即网络层。在讨论IP时，您经常会听到*第3层*，*L3*和*网络层*等术语。

向IP地址发送数据包的主机会查询其*路由表* 以确定数据包应发送到本地网络上的哪台计算机。路由表维护与主机直接连接的每个本地网络关联的子网列表，以及这些本地网络上的路由器列表。

在Linux机器上，以下任一命令都会显示路由表：

```
$ ip route show
 $ route -n
 $ netstat -rn
```

以下是**ip route show**的输出示例：

```
$ ip route show show
 default via 192.0.2.2 dev eth0
192.0.2.0/24 dev eth0 proto kernel scope链接src 192.0.2.15
198.51.100.0/25 dev eth1 proto kernel scope链接src 198.51.100.100
198.51.100.192/26 dev virbr0 proto kernel范围链接src 198.51.100.193
```

输出的第1行指定默认路由的位置，如果其他规则均不匹配，则为有效路由规则。与默认路由相关的路由器（**192.0.2.2**在上面的例子中）有时被称为*默认网关*。一个DHCP服务器通常与客户端的IP地址和子网掩码一起发送默认网关设置为DHCP客户端的IP地址。

输出的第2行指定**192.0.2.0/24**子网中的IP位于与网络接口eth0关联的本地网络上。

输出的第3行指定**198.51.100.0/25**子网中的IP位于与网络接口eth1关联的本地网络上。

输出的第4行指定**198.51.100.192/26**子网中的IP位于与网络接口virbr0关联的本地网络上。

路由-n和**netstat -rn**命令的输出格式稍有不同。这个例子显示了如何使用这些命令格式化相同的路由：

```
$ route -n
 内核IP路由表
目的网关Genmask标志MSS窗口irtt Iface
0.0.0.0 192.0.2.2 0.0.0.0 UG 0 0 0 eth0
192.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
198.51.100.0 0.0.0.0 255.255.255.128 U 0 0 0 eth1
198.51.100.192 0.0.0.0 255.255.255.192 U 0 0 0 virbr0
```

在**IP路由得到**命令输出的目的地IP地址的路由。从以下示例中，目标IP地址**192.0.2.14**位于eth0的本地网络上，并将直接发送：

```
$ ip route get 192 .0.2.14
 192.0.2.14 dev eth0 src 192.0.2.15
```

目标IP地址**203.0.113.34**不在任何连接的本地网络上，并将在以下位置转发到默认网关**192.0.2.2**：

```
$ ip route获取203 .0.113.34
 203.0.113.34通过192.0.2.2 dev eth0 src 192.0.2.15
```

数据包通过多个路由器跳转到达最终目的地是很常见的。在Linux机器上，**traceroute**更新的**mtr** 程序打印出每个路由器的IP地址，IP数据包沿其路径到达目的地。

TCP / UDP / ICMP¶

对于通过IP网络进行通信的联网软件应用程序，他们必须使用IP层以上的协议。这些协议占用OSI模型的第四层，称为*传输层*或*第4层*。请参阅 由互联网号码分配机构（IANA）维护的 协议号码 (http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml)网页，以获取IP层及其关联号码上的协议列表。

的*传输控制协议*（TCP）是在网络应用中最常用的4层协议。TCP是 *面向连接的*协议：它使用客户端 - 服务器模型，其中客户端连接到服务器，其中*服务器*指的是接收连接的应用程序。基于TCP的应用程序中的典型交互过程如下所示：

1. 客户端连接到服务器。
2. 客户端和服务器交换数据。
3. 客户端或服务器断开连接。

由于网络主机可能有多个基于TCP的应用程序在运行，TCP使用称为*端口*的寻址方案来唯一标识基于TCP的应用程序。TCP端口与1-65535范围内的数字相关联，并且主机上的一个应用程序一次只能与一个TCP端口相关联，这是操作系统强制执行的限制。

据说TCP服务器在端口上*侦听*。例如，SSH服务器通常监听端口22.对于客户端使用TCP连接到服务器，客户端必须知道服务器主机的IP地址和服务器的TCP端口。

TCP客户端应用程序的操作系统自动将端口号分配给客户端。客户端拥有此端口号，直到TCP连接终止，之后操作系统回收端口号。这些类型的端口被称为*临时端口*。

IANA 为许多基于TCP的服务以及使用其他使用端口的第4层协议的服务维护一个端口号 (http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml)的注册表 (http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml)。注册TCP端口号不是必需的，但注册端口号有助于避免与其他服务冲突。请参阅 OpenStack安装指南中的防火墙和默认端口 (https://docs.openstack.org/install-guide/firewalls-default-ports.html)，了解OpenStack部署中涉及的各种服务使用的默认TCP端口。

用于编写基于TCP的应用程序的最常见的应用程序编程接口（API）称为*Berkeley套接字*，也称为*BSD套接字*，或简称为*套接字*。套接字API暴露了一个*面向流*编写TCP应用程序的接口。从程序员的角度来看，通过TCP连接发送数据类似于将一串字节写入文件。操作系统的TCP / IP实现负责将数据流分解为IP数据包。操作系统还负责自动转发丢弃的数据包，并负责处理流量控制，以确保传输的数据不会超出发送方的数据缓冲区，接收方的数据缓冲区和网络容量。最后，操作系统负责将数据包以正确的顺序重新组装成接收方的数据流。由于TCP检测并重新传输丢失的数据包，因此它被认为是*可靠的*协议。

的*用户数据报协议*（UDP）是另一种层4协议，是的几个众所周知的网络协议的基础。UDP是无 *连接*协议：通过UDP进行通信的两个应用程序在交换数据之前不需要建立连接。UDP也是一种*不可靠的*协议。操作系统不会尝试重新发送或甚至检测丢失的UDP数据包。操作系统也不能保证接收应用程序按照它们发送的相同顺序看到UDP数据包。

与TCP一样，UDP使用端口的概念来区分在同一系统上运行的不同应用程序。但请注意，操作系统将UDP端口与TCP端口分开处理。例如，可以将一个应用程序与TCP端口16543关联，并将单独的应用程序与UDP端口16543关联。

与TCP一样，套接字API是编写基于UDP的应用程序的最常用API。套接字API 为编写UDP应用程序提供了*面向消息的*接口：程序员通过发送固定大小的消息通过UDP发送数据。如果应用程序要求重新传输丢失的数据包或明确定义接收数据包的顺序，程序员将负责在应用程序代码中实现此功能。

DHCP, the Domain Name System (DNS), the Network Time Protocol (NTP), and Virtual extensible local area network (VXLAN) (intro-overlay-protocols.html#vxlan) are examples of UDP-based protocols used in OpenStack deployments.

UDP支持一对多通信：将一个数据包发送到多个主机。应用程序可以通过将接收方IP地址设置为特殊IP广播地址，将UDP数据包广播到本地网络上的所有网络主机 **255.255.255.255**。应用程序还可以使用*IP多播*将UDP数据包发送到一组接收器。预期的接收者应用通过将UDP套接字绑定到作为有效多播组地址之一的特殊IP地址来加入多播组。接收主机不必与发送方位于同一本地网络中，但中间路由器必须配置为支持IP多播路由。VXLAN是使用IP多播的基于UDP协议的示例。

的*因特网控制消息协议*（ICMP）是用于通过IP网络发送控制消息的协议。例如，如果路由器的路由表中没有与目标地址相对应的路由（ICMP代码1，目标主机不可达）或者IP数据包为空，则接收IP数据包的路由器可能会将ICMP数据包发送回源路由器处理太大（ICMP代码4，需要分片并设置"不分片"标记）。

在**平安**和**地铁** Linux的命令行工具的使用ICMP网络工具的两个例子。

« (intro.html) » (intro-network-components.html) (https://bugs.launchpad.net/neutron/+filebug?field.title=Basic%20networking%20in%20Neutron&field.comment=%0A%0A%0AThis bug tracker is for errors with the documentation, use the following as a template and remove or add fields as you see fit. Convert [ ] into [x] to check boxes:%0A%0A- [ ] This doc is inaccurate in this way: _____%0A- [ ] This is a doc addition request.%0A- [ ] I have a fix to the document that I can paste below including example: input and output. %0A%0AIf you have a troubleshooting or support issue, use the following resources:%0A%0A - Ask OpenStack: http://ask.openstack.org%0A - The mailing list: http://lists.openstack.org%0A - IRC: 'openstack' channel on Freenode%0A%0A----------%0ARelease:%2012.0.1.dev11%20on%202018-03-07%2021:05%0ASHA:%2043df2709acbdce86686a40b75fd34e96880427d0%0ASource:%20https://git.openstack.org/cgit/openstack/neutron/tree/doc/source/admin/intro-basic-networking.rst%0AURL: https://docs.openstack.org/neutron/queens/admin/intro-basic-networking.html&field.tags=doc)

更新日期：2018-03-07 21:05

OpenStack文档

Neutron 12.0.1

(../index.html)

安装指南 (../install/index.html)

OpenStack网络指南 (index.html)
- 介绍 (intro.html)
- 组态 (config.html)
- 部署示例 (deploy.html)
- 操作 (ops.html)
- 移民 (migration.html)
- 杂 (misc.html)
- 存档的内容 (archives/index.html)

中子配置选项 (../configuration/index.html)

命令行界面参考 (../cli/index.html)

中子特征分类 (../feature_classification/index.html)

贡献者指南 (../contributor/index.html)

页面内容

以太网络
VLAN的
子网和ARP
DHCP
IP
TCP / UDP / ICMP

OpenStack的

- 项目 (http://openstack.org/projects/)
- OpenStack安全 (http://openstack.org/projects/openstack-security/)
- 常见问题 (http://openstack.org/projects/openstack-faq/)
- 博客 (http://openstack.org/blog/)
- 新闻 (http://openstack.org/news/)

社区

- 用户组 (http://openstack.org/community/)
- 活动 (http://openstack.org/community/events/)
- 工作 (http://openstack.org/community/jobs/)
- 公司 (http://openstack.org/foundation/companies/)
- 有助于 (http://docs.openstack.org/infra/manual/developers.html)

文档

- OpenStack手册 (http://docs.openstack.org)
- 入门 (http://openstack.org/software/start/)
- API文档 (http://developer.openstack.org)
- 维基 (https://wiki.openstack.org)

品牌与法律

- 标志和指南 (http://openstack.org/brand/)
- 商标政策 (http://openstack.org/brand/openstack-trademark-policy/)
- 隐私政策 (http://openstack.org/privacy/)
- OpenStack CLA (https://wiki.openstack.org/wiki/How_To_Contribute#Contributor_License_Agreement)

保持联系

(https://t(https://www.facebook.com/OpenStackFoundation)