002-Linux 虚拟网络设备

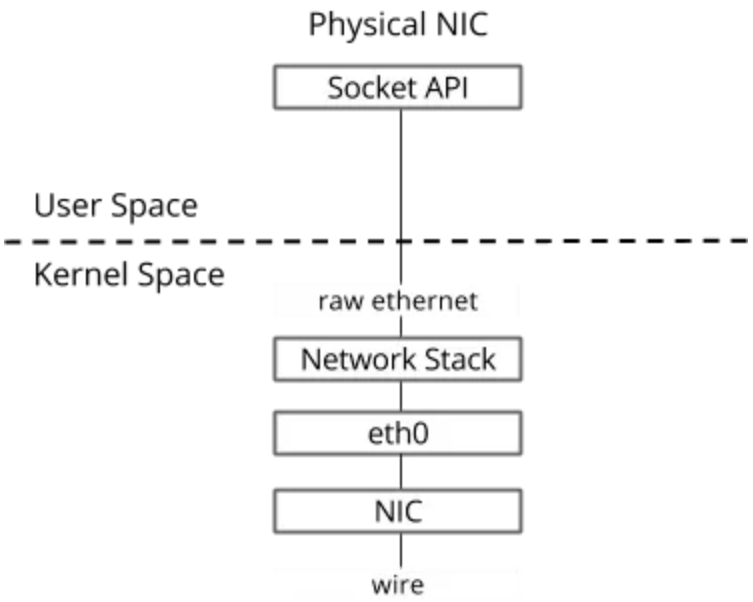
随着容器逐步取代虚拟机，成为现在云基础架构的标准，这些容器的网络管理模块都离不开 Linux 虚拟网络设备。事实上，了解常用的 Linux 虚拟网络设备对于我们理解容器网络以及其他依赖于容器的基础网络架构实现都大有裨益。现在开始，我们就来看看常见的 Linux 虚拟网络设备有哪些以及它们的典型使用场景。

1. 网络设备

网络设备的驱动程序并不直接与内核中的协议栈交互，而是通过内核的网络设备管理模块作为中间桥梁。这样做的好处就是，驱动程序不需要了解网络协议栈的细节，协议栈也无需针对特定网络设备驱动处理数据包。

对于内核网络设备模块来说，虚拟设备与物理设备没有区别，都是网络设备，都能配置IP地址，都类似于管道，从一端接收到的数据将从另外一端发送出去，比如**物理网卡的两端分别是协议栈与外面的物理网络**，从外面物理网络接收到的数据包会转发给协议栈，相反，应用程序通过协议栈发送过来的数据包会通过物理网卡发送到外面的物理网络。但是对于具体将数据包发送到哪里，怎么发送，不同的网络设备有不同的驱动实现，与内核设备管理模块以及协议栈没关系。

总的来说，**虚拟网络设备与物理网络设备没有什么区别，他们的一端连接着内核协议栈，而另一端的行为是什么取决于不同网络设备的驱动实现。**



2. Tun/Tap设备

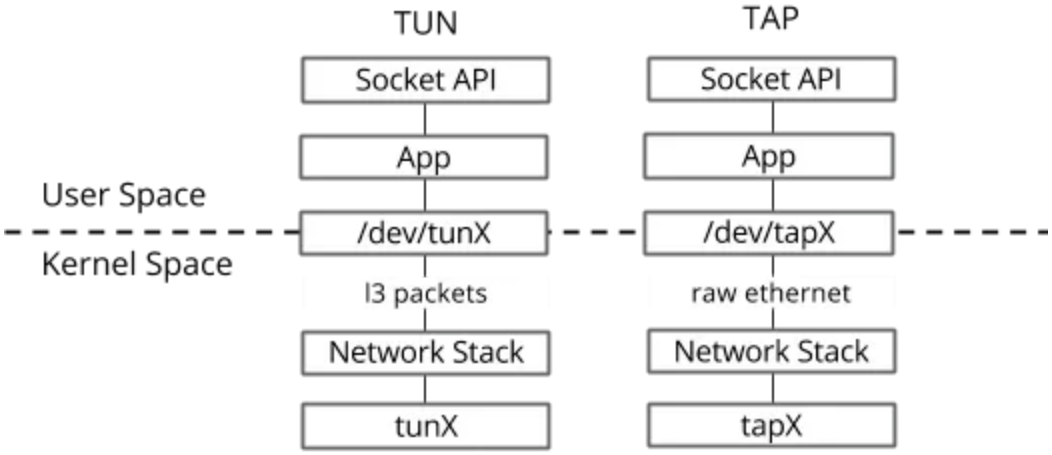
2.1 认识Tun/Tap

- **Tun：虚拟的是点对点设备；Tap：虚拟的是以太网设备；**tun与tap是操作系统内核中的虚拟网络设备。不同于普通靠硬件网络适配器实现的设备，这些虚拟的网络设备全部用软件实现，并向运行于操作系统上的软件提供与硬件的网络设备完全相同的功能。
- **tun/tap设备到底是什么？**从Linux文件系统的角度看，它是用户可以用文件句柄操作的字符设备；从网络虚拟化角度看，它是虚拟网卡，一端连着**网络协议栈**，另一端连着**用户态程序。**
- **tun/tap设备有什么作用呢？tun是网络层的虚拟网络设备**，可以收发第三层数据报文包，如IP封包，因此常用于一些点对点IP隧道，例如OpenVPN，IPSec等。**tap是链路层的虚拟网络设备**，等同于一个以太网设备，它可以收发第二层数据报文包，如以太网数据帧。Tap最常见的用途就是做为虚拟机的网卡，因为它和普通的物理网卡更加相近，也经常用作普通机器的虚拟网卡。

OSI 7层模型

2.2 Tap/Tun工作原理

Linux Tun/Tap驱动程序为应用程序提供了两种交互方式：**虚拟网络接口和字符设备/dev/net/tun。写入字符设备/dev/net/tun的数据会发送到虚拟网络接口中；发送到虚拟网络接口中的数据也会出现在该字符设备上。**



应用程序可以通过标准的Socket API向Tun/Tap接口发送IP数据包，就好像对一个真实的网卡进行操作一样。除了应用程序以外，操作系统也会根据TCP/IP协议栈的处理向Tun/Tap接口发送IP数据包或者以

太网数据包，例如ARP或者ICMP数据包。Tun/Tap驱动程序会将Tun/Tap接口收到的数据包原样写入到/dev/net/tun字符设备上，处理Tun/Tap数据的应用程序如VPN程序可以从该设备上读取到数据包，以进行相应处理。

应用程序也可以通过/dev/net/tun字符设备写入数据包，这种情况下该字符设备上写入的数据包会被发送到Tun/Tap虚拟接口上，进入操作系统的TCP/IP协议栈进行相应处理，就像从物理网卡进入操作系统的数据一样。

Tun虚拟设备和物理网卡的区别是Tun虚拟设备是IP层设备，从/dev/net/tun字符设备上读取的是IP数据包，写入的也只能是IP数据包，因此不能进行二层操作，如发送ARP请求和以太网广播。与之相对的是，Tap虚拟设备是以太网设备，处理的是二层以太网数据帧，从/dev/net/tun字符设备上读取的是以太网数据帧，写入的也只能是以太网数据帧。从这点来看，Tap虚拟设备和真实的物理网卡的能力更接近。

2.3 Tun/Tap区别

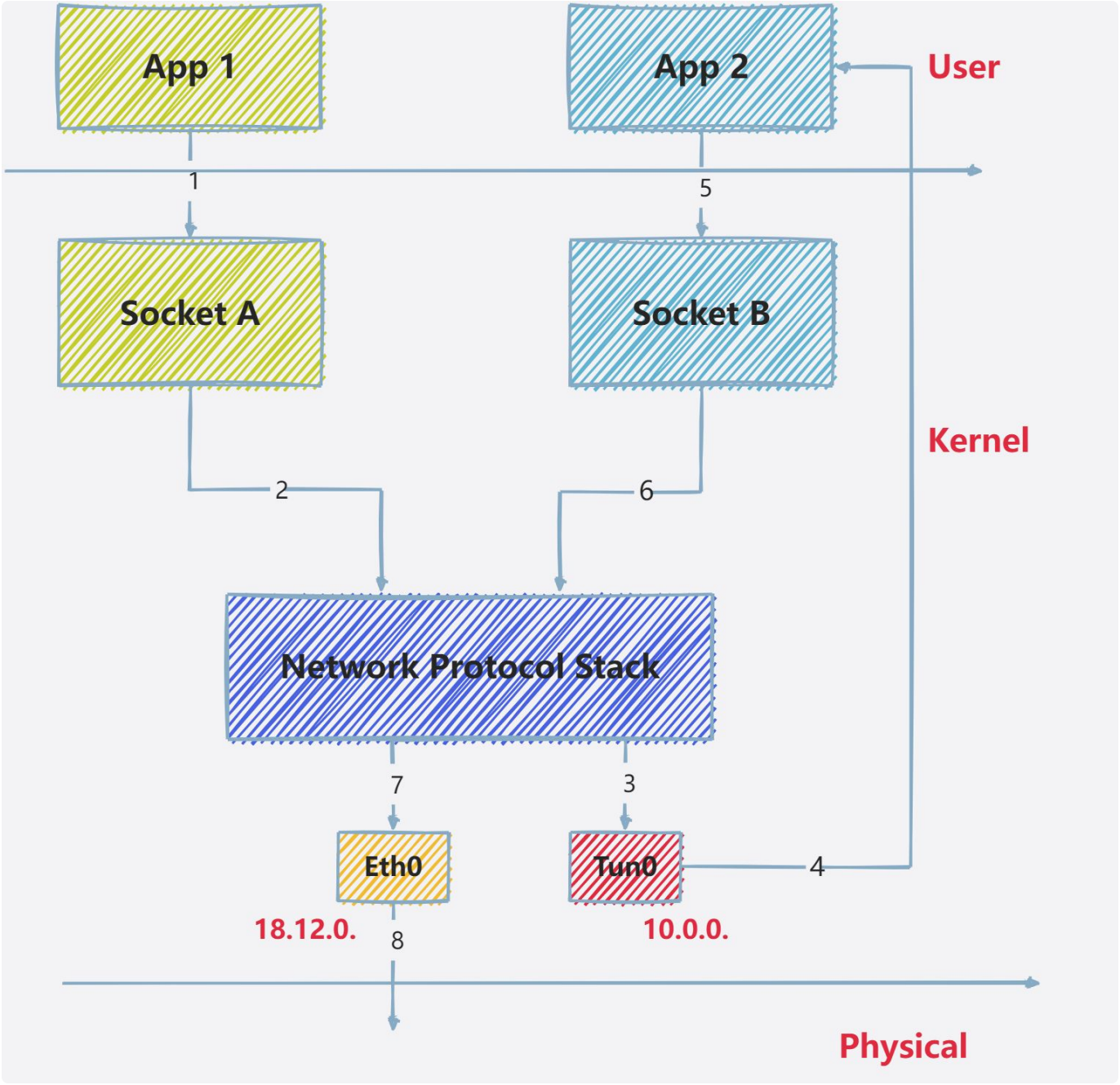
tun/tap 虽然工作原理一致，但是工作的层次不一样。

tun是三层网络设备，收发的是IP层数据包，无法处理以太网数据帧，例如OpenVPN的路由模式就是使用了tun网络设备，OpenVPN Server重新规划了一个网段，所有的客户端都会获取到该网段下的一个IP，并且会添加对应的路由规则，而客户端与目标机器产生的数据报文都要经过OpenVPN网关才能转发。

tap是二层网络设备，收发以太网数据帧，拥有MAC层的功能，可以和物理网卡通过网桥相连，组成一个二层网络。例如OpenVPN的桥接模式可以从外部打一条隧道到本地网络。进来的机器就像本地的机器一样参与通讯，丝毫看不出这些机器是在远程。如果你有使用过虚拟机的经验，桥接模式也是一种十分常见的网络方案，虚拟机会分配到和宿主机器同网段的IP，其他同网段的机器也可以通过网络访问到这台虚拟机。

tun/tap设备的用处是将协议栈中的部分数据包转发给用户空间的应用程序，给用户空间的程序一个处理数据包的机会。常见的tun/tap设备使用场景有数据压缩、加密等，最常见的是VPN，包括tunnel及应用层的IPSec等。

一个典型的使用TUN/TAP网络设备的例子如下所示



上图中我们配置了一个物理网卡，IP 为18.12.0.92，而 Tun0 为一个 TUN设备，IP 配置为10.0.0.12。数据包的流向为：

1. App1是一个普通的程序，通过Socket API发送了一个数据包，假设这个数据包的目的IP地址是**10.0.0.22**（和tun0在同一个网段）。
2. socket A 将这个数据包丢给网络协议栈
3. App1的数据包到达网络协议栈后，协议栈根据数据包的目的IP地址匹配到这个数据包应该由tun0网口出去，于是将数据包发送给tun0网卡。
4. tun0网卡收到数据包之后，发现网卡的另一端被App2打开了（这也是tun/tap设备的特点，**一端连着协议栈，另一端连着用户态程序**），于是将数据包发送给App2。
5. App2收到数据包之后，通过报文封装构造出一个新的数据包。

> Note: 新数据包的源地址变成了 tun0 的地址，而目的 IP 地址则变成了另外一个地址 18.13.0.91.

6. sApp2通过同样的socket B将数据包发送给协议栈。
7. 协议栈根据本地路由规则和数据包的目的 IP，决定将这个数据包要通过设备 eth0 发送出去，于是将数据包转发给设备 eth0，最后eth0通过物理网络将数据包发送给VPN的对端
8. 设备 eth0 通过物理网络将数据包发送出去

我们看到发送给10.0.0.22的网络数据包通过在用户空间的应用程序B，利用18.12.0.92发送到远端网络18.12.0.91，网络包到达18.12.0.91后，读取里面的原始数据包，在转发给本地的10.0.0.22。这就是 VPN 的基本实现原理。

VPN网络的报文真正从物理网卡出去要经过网络协议栈两次，因此会有一定的性能损耗。另外，经过用户态程序的处理，数据包可能已经加密，包头进行了封装，所以第二次通过网络栈内核看到的是截然不同的网络包。

综上所述，TUN 和 TAP 设备的区别在于，**TUN 设备是一个虚拟的端到端 IP 层设备**，也就是说用户空间的应用程序通过 TUN 设备只能读写 IP 网络数据包（三层），而 **TAP 设备是一个虚拟的链路层设备**，通过 TAP 设备能读写链路层数据包（二层）。

> 如果使用 Linux 网络工具包 iproute2 来创建网络设备 TUN/TAP 设备 则需要指定 --dev tun 和 --dev tap 来区分。

2.4 Tun/Tap使用

Linux 提供了一些命令行程序方便我们来创建持久化的tun/tap设备，但是如果没有应用程序打开对应的文件描述符，tun/tap的状态一直会是DOWN，但这并不会影响我们把它当作普通网卡去使用。

使用`ip tuntap help`查看使用帮助

Shell | 复制代码

```
[root@euler-x86-70 ~]# ip tuntap help
Usage: ip tuntap { add | del | show | list | lst | help } [ dev PHYS_DEV ]
   [ mode { tun | tap } ] [ user USER ] [ group GROUP ]
   [ one_queue ] [ pi ] [ vnet_hdr ] [ multi_queue ] [ name NAME ]

Where:  USER  := { STRING | NUMBER }
   GROUP := { STRING | NUMBER }
[root@euler-x86-70 ~]#

```

2.4.1 示例

```shell
# 创建 tap
ip tuntap add dev tap0 mode tap
# 创建 tun
ip tuntap add dev tun0 mode tun
# 删除 tap
ip tuntap del dev tap0 mode tap
# 删除 tun
ip tuntap del dev tun0 mode tun
```

tun/tap 设备创建成功后可以当作普通的网卡一样使用，因此我们也可以通过`ip link`命令来操作它。

```shell
# 例如使用ip link命令也可以删除tun/tap设备
ip link del tap0
ip link del tun0
```

2.4.2 实验

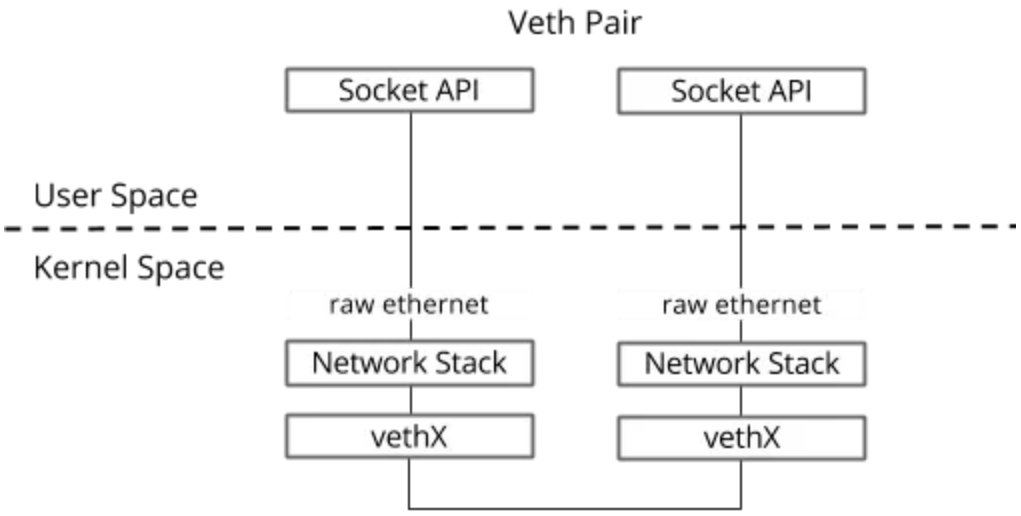
```shell
[root@cmss ~]# # 创建 tap
[root@cmss ~]# ip tuntap add dev tap0 mode tap
[root@cmss ~]# # 创建 tun
[root@cmss ~]# ip tuntap add dev tun0 mode tun
[root@cmss ~]#
[root@cmss ~]# ip a  | grep tap0 -C 3
    link/ether aa:ad:8d:f2:19:cc brd ff:ff:ff:ff:ff:ff link-netnsid 7
    inet6 fe80::a8ad:8dff:fef2:19cc/64 scope link
       valid_lft forever preferred_lft forever
103: tap0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group defa
ult qlen 1000
    link/ether ae:d9:dd:66:2a:05 brd ff:ff:ff:ff:ff:ff  <具有mac地址，可当做
普通的网卡来使用>
104: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN gr
oup default qlen 500
    link/none    <无mac地址>
[root@cmss ~]
```

3. veth设备

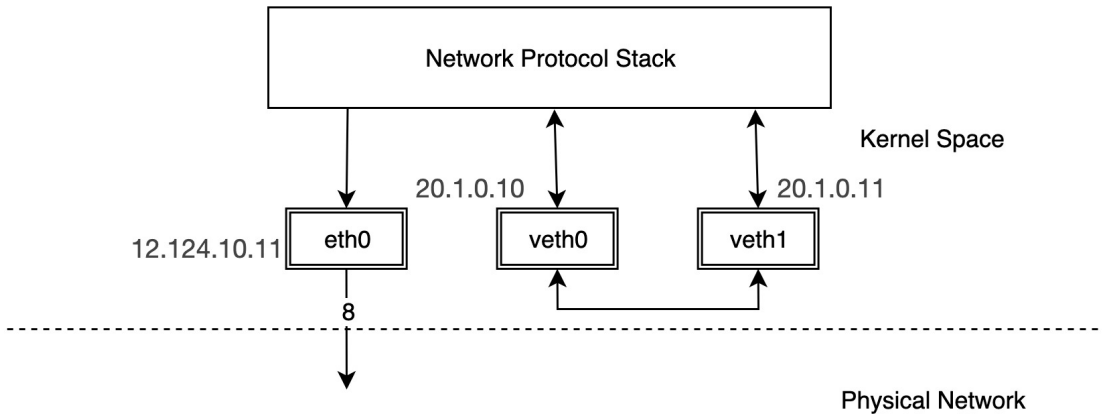
3.1 认识veth设备

veth-pair 是成对出现的一种虚拟网络设备，一端连接着协议栈，一端连接着彼此，数据从一端进，从另一端出。

它的这个特性常常用来连接不同的虚拟网络组件，构建大规模的虚拟网络拓扑，比如连接 Linux Bridge、OVS、LXC 容器等。每个veth设备都可以配置ip地址，并参与三层ip网络的路由过程。



3.2 veth应用



我们配置物理网卡 eth0 的 IP 地址为12.124.10.11，这里 veth 设备对分别为 veth0 和 veth1，它们的 IP 分别是20.1.0.10和20.1.0.11：

Shell | 复制代码

```
# ip link add veth0 type veth peer name veth1
# ip addr add 20.1.0.10/24 dev veth0
# ip addr add 20.1.0.11/24 dev veth1
# ip link set veth0 up
# ip link set veth1 up
```

然后尝试从veth0设备ping另外一个设备veth1

Shell | 复制代码

```
# ping -c 2 20.1.0.11 -I veth0
PING 20.1.0.11 (20.1.0.11) from 20.1.0.11 veth0: 28(42) bytes of data.
64 bytes from 20.1.0.11: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 20.1.0.11: icmp_seq=2 ttl=64 time=0.052 ms

--- 20.1.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1500ms
```

> Note: 在有些版本的 Ubuntu 中有可能会 ping 不通，原因是默认情况下内核网络配置导致 veth 设备对无法返回 ARP 包，解决办法是配置 veth 设备可以返回 ARP 包：

```shell
# echo 1 > /proc/sys/net/ipv4/conf/veth1/accept_local
# echo 1 > /proc/sys/net/ipv4/conf/veth0/accept_local
# echo 0 > /proc/sys/net/ipv4/conf/veth0/rp_filter
# echo 0 > /proc/sys/net/ipv4/conf/veth1/rp_filter
# echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
```

可以尝试使用 tcpdump 命令看看在 veth 设备对上的请求包：

```shell
# tcpdump -n -i veth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth1, link-type EN10MB (Ethernet), capture size 458122 bytes
20:24:12.220002 ARP, Request who-has 20.1.0.11 tell 20.1.0.10, length 28
20:24:12.220198 ARP, Request who-has 20.1.0.11 tell 20.1.0.10, length 28
20:24:12.221372 IP 20.1.0.10 > 20.1.0.11: ICMP echo request, id 18174, seq 1, length 64
20:24:13.222089 IP 20.1.0.10 > 20.1.0.11: ICMP echo request, id 18174, seq 2, length 64
```

可以看到在 veth1 上面只有 ICMP echo 的请求包，但是没有应答包。仔细想一下，veth1 收到 ICMP echo请求包后，转交给另一端的协议栈，但是协议栈检查当前的设备列表，发现本地有20.1.0.10，于是构造 ICMP echo 应答包，并转发给 lo 设备，lo 设备收到数据包之后直接交给协议栈，紧接着给交给用户空间的 ping 进程。

我们可以尝试使用 tcpdump 抓取 lo 设备上的数据：

```shell
# tcpdump -n -i lo
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 458122 bytes
20:25:49.486019 IP IP 20.1.0.11 > 20.1.0.10: ICMP echo reply, id 24177, seq 1, length 64
20:25:50.4861228 IP IP 20.1.0.11 > 20.1.0.10: ICMP echo reply, id 24177, seq 2, length 64
```

由此可见，对于成对出现的 veth 设备对，从一个设备出去的数据包会直接发给另外一个设备。在实际的应用场景中，比如容器网络中，成对的 veth 设备对处于不同的网络命名空间中，数据包的转发在不同网络命名空间之间进行，后续在介绍容器网络的时候会详细说明。

4. bridge设备

4.1 认识bridge设备

Bridge一般叫做“网桥”，也是一种虚拟网络设备，所以具有虚拟网络设备的特征，可以配置IP、MAC地址等。bridge与其他网络设备不同的是，他是一个虚拟交换机，和物理交换机有类似的功能，bridge一端连接着协议栈，另外一端有多个端口，数据在各个端口间转发数据包是基于MAC地址。

Bridge可以工作在二层（链路层），也可以工作在三层（IP网络层）。默认情况下，其工作在二层，可以在同一子网内的不同主机转发以太网报文；当给bridge分配了ip地址，也就是开启了该bridge的三层工作模式。在Linux下，您可以使用 iproute2 或 brctl 命令对 bridge 进行管理。

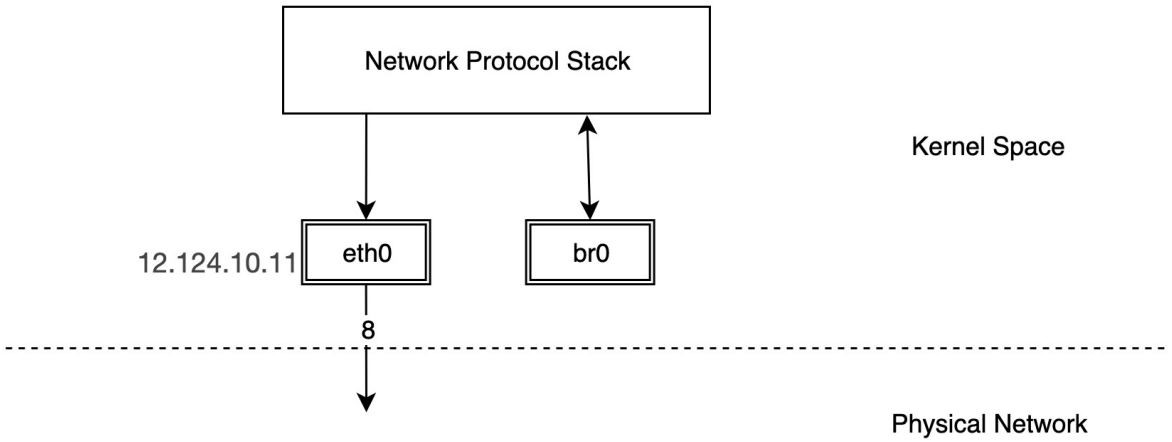
4.2 bridge应用

4.2.1 veth0的变化

创建 bridge 与创建其他虚拟网络设备类似，只需要指定 type 参数为 bridge：

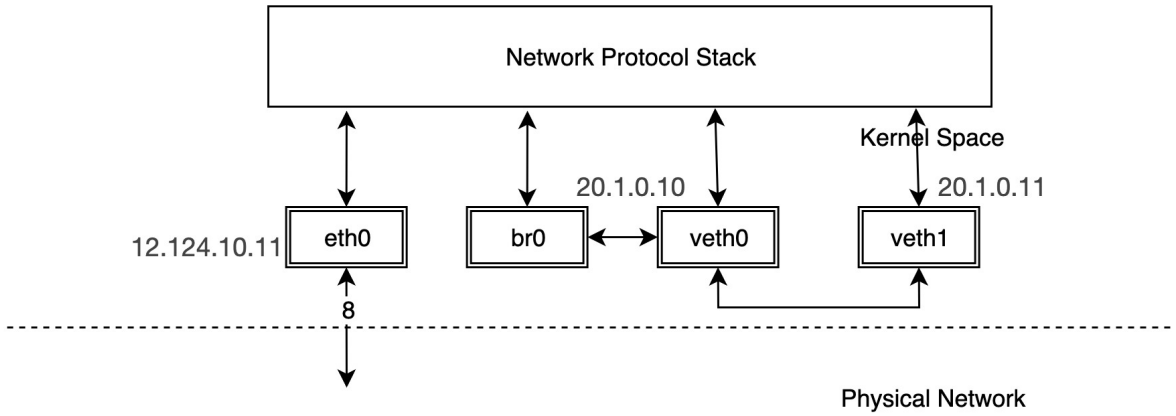
Shell | 复制代码

```
# ip link add name br0 type bridge
# ip link set br0 up
```



但是这样创建出来的 bridge 一端连接着协议栈，其他端口什么也没有连接，因此我们需要将其他设备连接到该 bridge 才能有实际的功能：

```shell
# ip link add veth0 type veth peer name veth1
# ip addr add 20.1.0.10/24 dev veth0
# ip addr add 20.1.0.11/24 dev veth1
# ip link set veth0 up
# ip link set veth1 up
# 将 veth0 连接到 br0
# ip link set dev veth0 master br0
# 通过 bridge link 命令可以看到 bridge 上连接了哪些设备
# bridge link
6: veth0 state UP : <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0
state forwarding priority 32 cost 2
```



事实上，一旦 br0 和 veth0 连接之后，它们之间将变成双向通道，但是内核协议栈和 veth0 之间变成了单通道，协议栈能发数据给 veth0，但 veth0 从外面收到的数据不会转发给协议栈，同时 br0 的 MAC 地址变成了 veth0 的 MAC 地址。我们可以验证一下：

```shell
# ping -c 1 -I veth0 20.1.0.11
PING 20.1.0.11 (20.1.0.11) from 20.1.0.10 veth0: 56(84) bytes of data.
From 20.1.0.10 icmp_seq=1 Destination Host Unreachable

--- 20.1.0.11 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

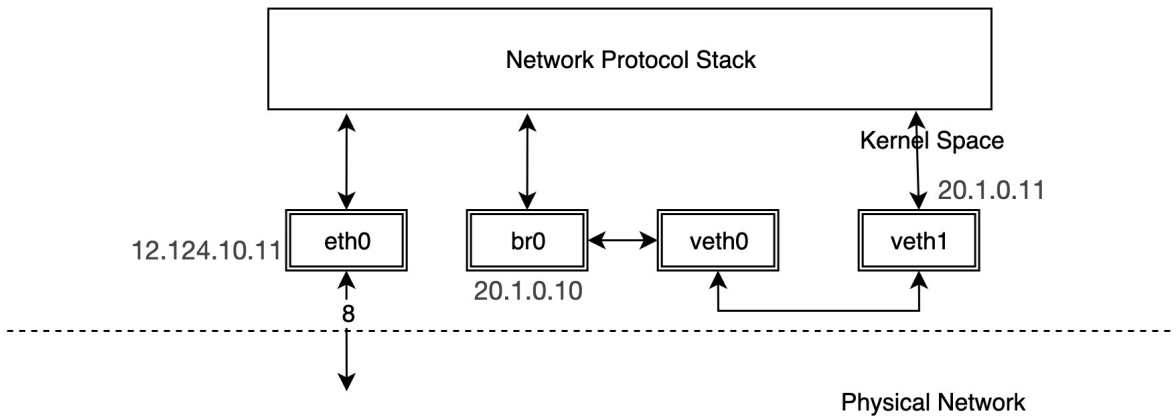
如果我们使用 tcpdump 在 br0 上抓包就会发现：

```shell
# tcpdump -n -i br0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on br0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:45:48.225459 ARP, Reply 20.1.0.10 is-at a2:85:26:b3:72:6c, length 28
```

可以看到 veth0 收到应答包后没有给协议栈，而是直接转发给 br0，这样协议栈得不到 veth1 的 MAC 地址，从而 ping 不通。br0 在 veth0 和协议栈之间将数据包给拦截了。但是如果我们给 br0 配置 IP，会怎么样呢?

```shell
# ip addr del 20.1.0.10/24 dev veth0
# ip addr add 20.1.0.10/24 dev br0
```

这样，网络结构就变成了下面这样：



这时候再通过 br0 来 ping 一下 veth1，会发现结果可以通：

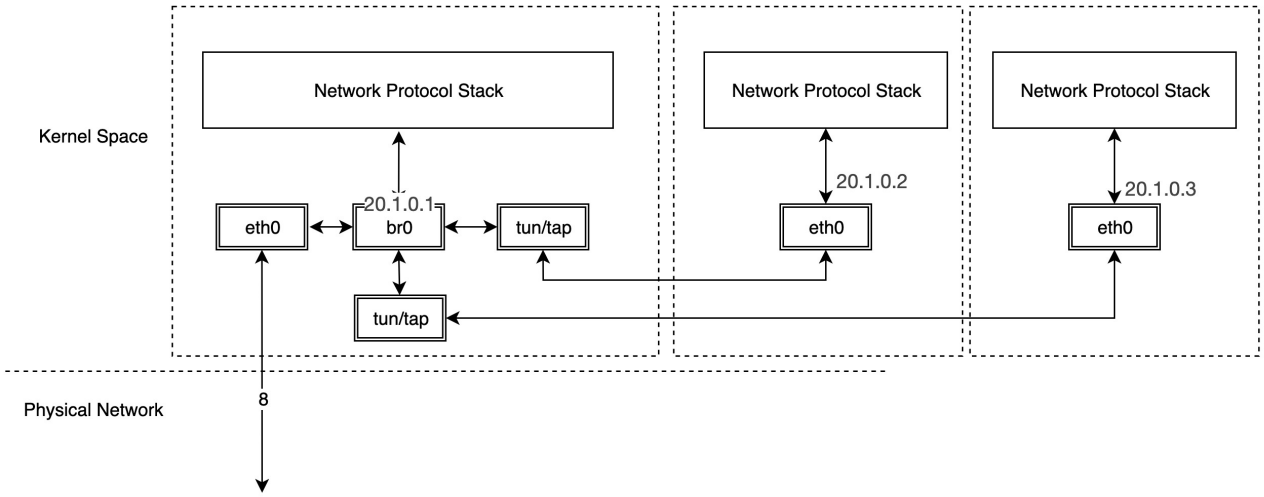
```shell
# ping -c 1 -I br0 20.1.0.11
PING 20.1.0.11 (20.1.0.11) from 20.1.0.10 br0: 56(84) bytes of data.
64 bytes from 20.1.0.11: icmp_seq=1 ttl=64 time=0.121 ms

--- 20.1.0.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.121/0.121/0.121/0.000 ms
```

其实当去掉veth0的ip，而给br0配置了ip之后，协议栈在路由的时候不会将数据包发给veth0，为了表达更直观，我们协议栈和 veth0 之间的连接线去掉，这时候的 veth0 相当于一根网线。

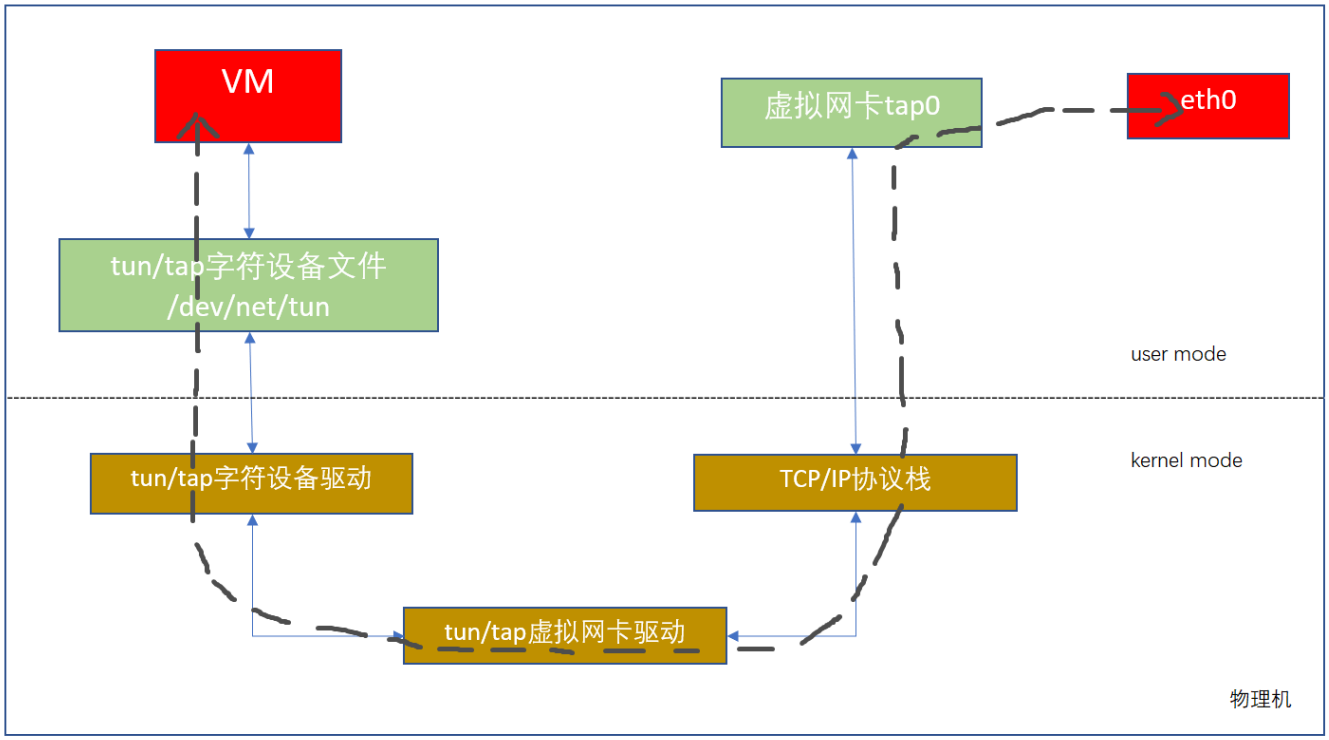
4.2.2 虚拟机

典型的虚拟机网络实现就是通过 TUN/TAP 将虚拟机内的网卡同宿主机的 br0 连接起来，这时 br0 和物理交换机的效果类似，虚拟机发出去的数据包先到达 br0，然后由 br0 交给 eth0 发送出去，这样做数据包都不需要经过宿主机的协议栈，运行效率非常高。



tap在libvirt中的应用

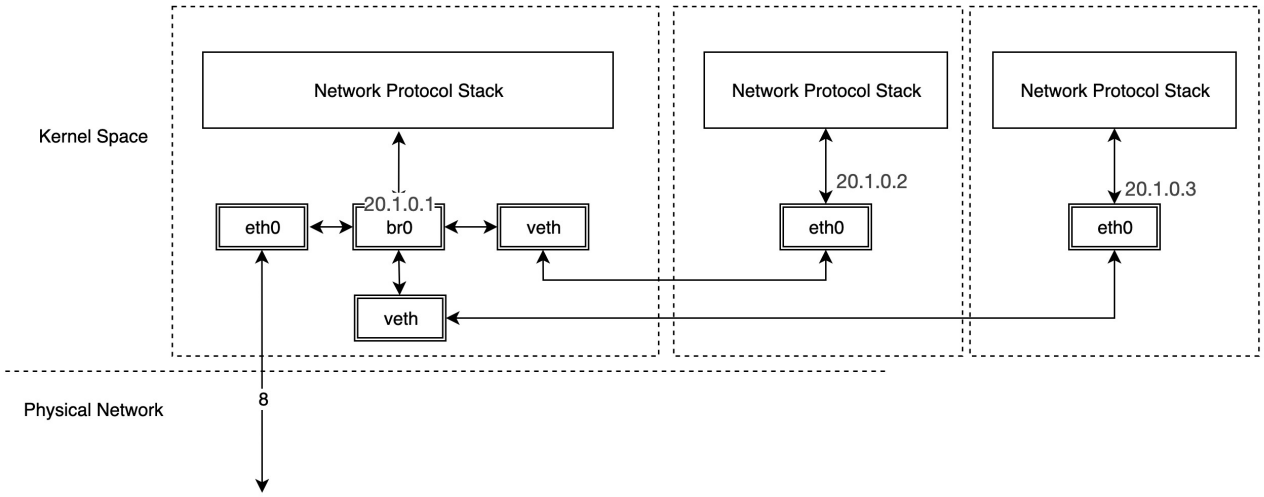
- 将guest system的网络和hostsystem的网络连在一起。
- 通过TUN/TAP adapter，会生成一个在host system上的虚拟网卡tap
- tun建立了point to point的网络设备，使得guest system的网卡和tap虚拟网卡成为一对
- 从而guest system的所有网络包，host system都能收到。



- 虚拟机将网络包通过字符设备写入/dev/net/tun（Host上）；
- 字符设备驱动将数据包写入虚拟网卡驱动；
- 虚拟网卡驱动将包通过TCP/IP协议栈写给Host上的虚拟网卡tap0；
- 在HOST上通过路由规则（通过网桥（东西向流量）、网桥和路由器（南北向流量）），包从eth0出去。

4.2.3 容器

而对于容器网络来说，每个容器的网络设备单独的网络命名空间中。



5.总结

最后，总结一下，我们提到几种网络设备，eth0、tap、tun、veth-pair，这些都构成了如今云网络必不可少的元素。