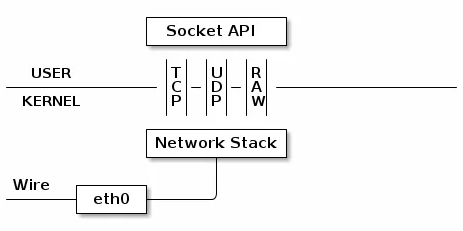
# 概述

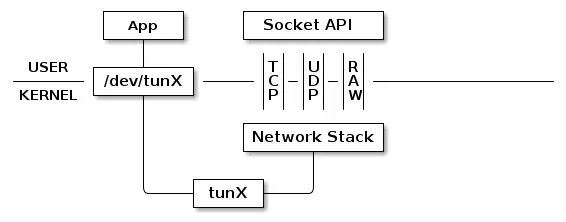
KVM网络优化方案，就是让虚拟机访问物理机网卡的层数更少，中间环节减少了，效率自然提高了，最理想的状态就是虚拟机独占物理网卡，和物理机一样使用物理网卡，这是网卡透传方案，除此外，还有好几种方案。

以往虚拟机数据包到达物理机的路径是：虚拟机🡪QEMU虚拟网卡🡪虚拟化层🡪内核网桥🡪物理网卡

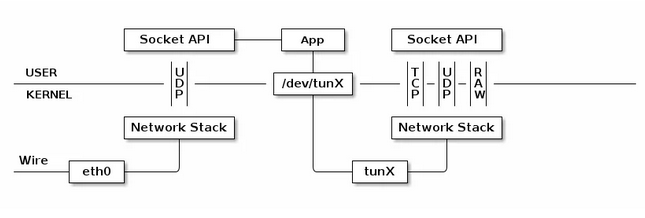
# 虚拟网络设备

## TUN 设备

TUN 设备是一种虚拟网络设备，通过此设备，程序可以方便得模拟网络行为。先来看看物理设备是如何工作的：

所有物理网卡收到的包会交给内核的 Network Stack 处理，然后通过 Socket API 通知给用户程序。下面看看 TUN 的工作方式：

普通的网卡通过网线收发数据包，但是 TUN 设备通过一个文件收发数据包。所有对这个文件的写操作会通过 TUN 设备转换成一个数据包送给内核；当内核发送一个包给 TUN 设备时，通过读这个文件可以拿到包的内容。

如果我们使用 TUN 设备搭建一个基于 UDP VPN，那么整个处理过程就是这样：

数据包会通过内核网络栈两次。但是经过 App 的处理后，数据包可能已经加密，并且原有的 ip 头被封装在 udp 内部，所以第二次通过网络栈内核看到的是截然不同的网络包。

## TAP 设备

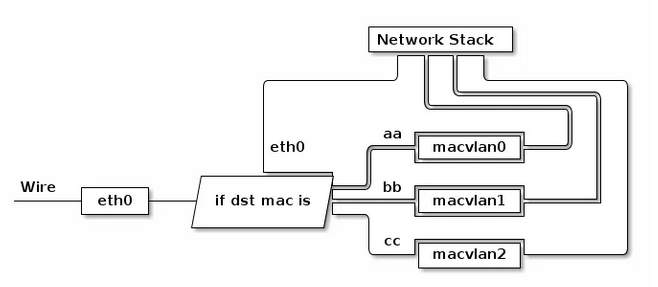
TAP 设备与 TUN 设备工作方式完全相同，区别在于：

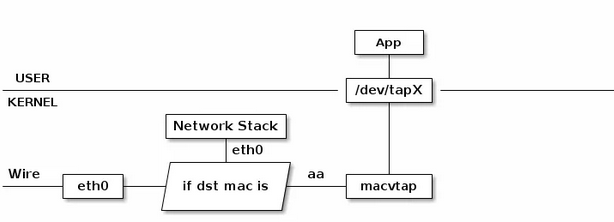
TUN 设备的 /dev/tunX 文件收发的是 IP 层数据包，只能工作在 IP 层，无法与物理网卡做 bridge，但是可以通过三层交换（如 ip\_forward）与物理网卡连通。

TAP 设备的 /dev/tapX 文件收发的是 MAC 层数据包，拥有 MAC 层功能，可以与物理网卡做 bridge，支持 MAC 层广播

## MACVLAN

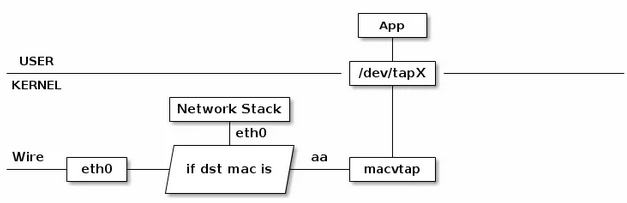
有时我们可能需要一块物理网卡绑定多个 IP 以及多个 MAC 地址，虽然绑定多个 IP 很容易，但是这些 IP 会共享物理网卡的 MAC 地址，可能无法满足我们的设计需求，所以有了 MACVLAN 设备，其工作方式如下：



MACVLAN 会根据收到包的目的 MAC 地址判断这个包需要交给哪个虚拟网卡。单独使用 MACVLAN 好像毫无意义，但是配合之前介绍的 network namespace 使用，我们可以构建这样的网络：

由于 macvlan 与 eth0 处于不同的 namespace，拥有不同的 network stack，这样使用可以不需要建立 bridge 在 virtual namespace 里面使用网络。

## MACVTAP

MACVTAP 是对 MACVLAN的改进，把 MACVLAN 与 TAP 设备的特点综合一下，使用 MACVLAN 的方式收发数据包，但是收到的包不交给 network stack 处理，而是生成一个 /dev/tapX 文件，交给这个文件：

由于 MACVLAN 是工作在 MAC 层的，所以 MACVTAP 也只能工作在 MAC 层，不会有 MACVTUN 这样的设备。

# 虚拟化技术

## 全虚拟化网卡

在KVM中，默认情况下网络设备是由QEMU在Linux的用户空间模拟出来并提供给虚拟机使用的。其优点在于通过模拟可以提供给虚拟机各种类型的网卡，但是由于网络I/O过程需要虚拟化引擎的参数，因此产生了大量的vm exit、vm entry，效率低下。

## 半虚拟化网卡

（1）概述

在实际应用中，使用较多的是半虚拟化技术，即virtio技术。Virtio驱动因为改造了虚拟机操作系统，让虚拟机可以直接和虚拟化层通信，从而大大提高了虚拟机的性能。

目前，Linux内核默认集成Virtio驱动，大部分Linux发行版可以直接使用Virtio驱动；Windows虚拟机一直都是使用全虚拟化方案，现在Virtio驱动不断改善，已经可以很好地支持Windows操作系统了，需要注意的是Windows虚拟机需要额外安装Virtio的驱动。

（2）配置

有两种方法配置半虚拟网卡，一种是基于命令行的，一种是基于config.xml配置文件的。

1. 在虚拟机启动命令中加入virtio-net-pci参数
2. 使用libvirt管理的KVM虚拟机，可以修改xml配置文件

## MacVTap技术

MacVTap是跳过内核网桥，虚拟化层直接和物理网卡进行通信。

## vhost-net技术

vhost\_net技术使虚拟机的网络通信绕过用户空间的虚拟化层，可以直接和内核通信，使用vhost\_net，必须使用Virtio半虚拟化网卡。vhost\_net是对Virtio优化，

## passthrough技术

### 概述

如果虚拟机对网络的要求比较高，通过PCI passthrough技术将物理网卡直接给虚拟机使用，虚拟机单独使用物理网卡，可以达到几乎和物理网卡一样的性能。

### 配置

1）使用lspci查看网卡硬件设备信息（bus:slot:function）

2）使用virsh dumpxml获取xml配置信息

3）编辑虚拟机xml配置文件，加入PCI信息

### 应用

PCI passthrough是性能最好的虚拟化解决方案，但是由于是独占网卡，所以对于宿主机的网卡数量有要求。可以使用1~2个网卡透传给网络压力比较大的虚拟机，其余网络压力较小的采用其余虚拟化方案。如果都是作为普通的办公使用，没必要采用这种奢侈的虚拟化方案了。

## SR-IOV技术

### 概述

SR-IOV是比较好的硬件解决方案，使用时需要特定的硬件支持，缺点是还不支持虚拟机的在线迁移。

SR-IOV（Single Root I/O Virtualization）是单根虚拟化，是一个PCI-E设备共享给虚拟机使用的标准，在网卡和显卡中有应用。SR-IOV提供了一种从硬件上染过系统和虚拟化层，并且每个虚拟机都能有单独的内存、终端、DMA流。

### 配置

1. 加载SR-IOV内核模块

modprobe igb

1. 配置文件写入

### 应用

# 操作

## 创建虚拟网络环境

使用命令

$ ip netns add net0

可以创建一个完全隔离的新网络环境，这个环境包括一个独立的网卡空间，路由表，ARP表，ip地址表，iptables，ebtables，等等。总之，与网络有关的组件都是独立的。

使用命令

$ ip netns list

net0

可以看到我们刚才创建的网络环境

## 进入虚拟网络环境

使用命令

$ ip netns exec net0

`command`

我们可以在 net0 虚拟环境中运行任何命令：

$ ip netns exec net0 bash

$ ip ad

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

这样我们可以在新的网络环境中打开一个shell，可以看到，新的网络环境里面只有一个lo设备，并且这个lo设备与外面的lo设备是不同的，之间不能互相通讯。

## 连接两个网络环境

新的网络环境里面没有任何网络设备，并且也无法和外部通讯，就是一个孤岛，通过下面介绍的这个方法可以把两个网络环境连起来，简单的说，就是在两个网络环境之间拉一根网线。

$ ip netns add net1

先创建另一个网络环境net1，我们的目标是把net0与net1连起来。

$ ip link add type veth

$ ip ad

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

81: veth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000

link/ether 12:39:09:81:3a:dd brd ff:ff:ff:ff:ff:ff

82: veth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000

link/ether 32:4f:fd:cc:79:1b brd ff:ff:ff:ff:ff:ff

这里创建连一对veth虚拟网卡，类似pipe，发给veth0的数据包veth1那边会收到，发给veth1的数据包veth0会收到。就相当于给机器安装了两个网卡，并且之间用网线连接起来了。

$ ip link set veth0 netns net0

$ ip link set veth1 netns net1

这两条命令的意思就是把veth0移动到net0环境里面，把veth1移动到net1环境里面，我们看看结果。

$ ip ad

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

$ ip netns exec net0 ip ad

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

81: veth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000

link/ether 12:39:09:81:3a:dd brd ff:ff:ff:ff:ff:ff

$ ip netns exec net1 ip ad

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

82: veth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000

link/ether 32:4f:fd:cc:79:1b brd ff:ff:ff:ff:ff:ff

veth0 veth1已经在我们的环境里面消失了，并且分别出现在net0与net1里面。下面我们简单测试一下net0与net1的联通性。

$ ip netns exec net0 ip link set veth0 up

$ ip netns exec net0 ip address add 10.0.1.1/24 dev veth0

$ ip netns exec net1 ip link set veth1 up

$ ip netns exec net1 ip address add 10.0.1.2/24 dev veth1

分别配置好两个设备，然后用ping测试一下联通性：

$ ip netns exec net0 ping -c 3 10.0.1.2

PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.

64 bytes from 10.0.1.2: icmp\_req=1 ttl=64 time=0.101 ms

64 bytes from 10.0.1.2: icmp\_req=2 ttl=64 time=0.057 ms

64 bytes from 10.0.1.2: icmp\_req=3 ttl=64 time=0.048 ms

--- 10.0.1.2 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 1999ms

rtt min/avg/max/mdev = 0.048/0.068/0.101/0.025 ms

# Open vSwitch