

Linux 上的基础网络设备详解

抽象网络设备的原理及使用

网络虚拟化是 Cloud 中的一个重要部分。作为基础知识，本文详细讲述 Linux 抽象出来的各种网络设备的原理、用法、数据流向。您通过此文，能够知道如何使用 Linux 的基础网络设备进行配置以达到特定的目的，分析出 Linux 可能的网络故障原因。

夏文超，软件工程师，现从事 OpenVirtualization 方面的工作。您可以通过 developerWorks 社区与[夏文超](#)进行交流。

2013 年 10 月 28 日

Linux 抽象网络设备简介

和磁盘设备类似，Linux 用户想要使用网络功能，不能通过直接操作硬件完成，而需要直接或间接的操作一个 Linux 为我们抽象出来的设备，既通用的 Linux 网络设备来完成。一个常见的情况是，系统里装有一个硬件网卡，Linux 会在系统里为其生成一个网络设备实例，如 eth0，用户需要对 eth0 发出命令以配置或使用它了。更多的硬件会带来更多的设备实例，虚拟的硬件也会带来更多的设备实例。随着网络技术，虚拟化技术的发展，更多的高级网络设备被加入了到了 Linux 中，使得情况变得更加复杂。在以下章节中，将一一分析在虚拟化技术中经常使用的几种 Linux 网络设备抽象类型：Bridge、802.1q VLAN device、VETH、TAP，详细解释如何用它们配合 Linux 中的 Route table、IP table 简单的创建出本地虚拟网络。



在 IBM Bluemix 云平台上开发并部署您的下一个应用。

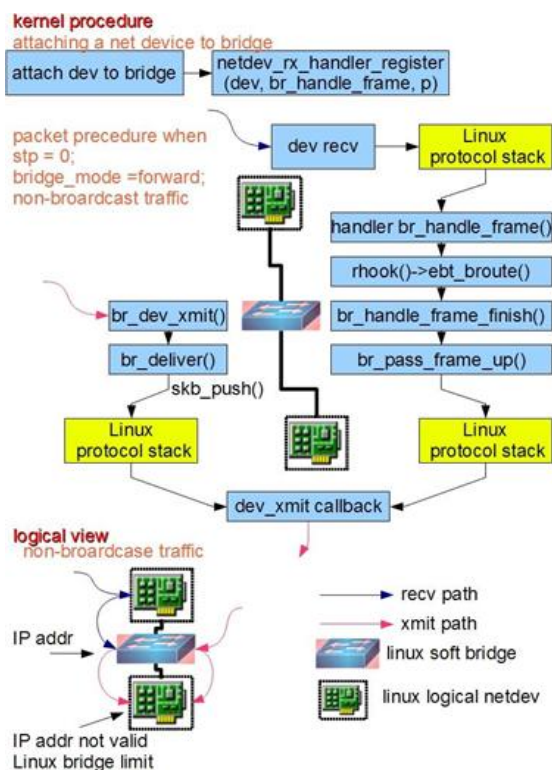
开始您的试用

相关网络设备工作原理

Bridge

Bridge（桥）是 Linux 上用来做 TCP/IP 二层协议交换的设备，与现实世界中的交换机功能相似。Bridge 设备实例可以和 Linux 上其他网络设备实例连接，既 attach 一个从设备，类似于在现实世界中的交换机和一个用户终端之间连接一根网线。当有数据到达时，Bridge 会根据报文中的 MAC 信息进行广播、转发、丢弃处理。

图 1. Bridge 设备工作过程



如图所示，**Bridge** 的功能主要在内核里实现。当一个从设备被 **attach** 到 **Bridge** 上时，相当于现实世界里交换机的端口被插入了一根连有终端的网线。这时在内核程序里，**netdev_rx_handler_register()** 被调用，一个用于接受数据的回调函数被注册。以后每当这个从设备收到数据时都会调用这个函数可以把数据转发到 **Bridge** 上。当 **Bridge** 接收到此数据时，**br_handle_frame()** 被调用，进行一个和现实世界中的交换机类似的处理过程：判断包的类别（广播/单点），查找内部 **MAC** 端口映射表，定位目标端口号，将数据转发到目标端口或丢弃，自动更新内部 **MAC** 端口映射表以自我学习。

Bridge 和现实世界中的二层交换机有一个区别，图中左侧画出了这种情况：**数据被直接发到 Bridge 上，而不是从一个端口接受。**这种情况可以看做 **Bridge** 自己有一个 **MAC** 可以主动发送报文，或者说 **Bridge** 自带了一个隐藏端口和寄主 **Linux** 系统自动连接，**Linux** 上的程序可以直接从这个端口向 **Bridge** 上的其他端口发数据。所以当有一个 **Bridge** 拥有一个网络设备时，如 **bridge0** 加入了 **eth0** 时，实际上 **bridge0** 拥有两个有效 **MAC** 地址，一个是 **bridge0** 的，一个是 **eth0** 的，他们之间可以通讯。由此带来一个有意思的事情是，**Bridge** 可以设置 **IP** 地址。通常来说 **IP** 地址是三层协议的内容，不应该出现在二层设备 **Bridge** 上。但是 **Linux** 里 **Bridge** 是通用网络设备抽象的一种，只要是网络设备就能够设定 **IP** 地址。当一个 **bridge0** 拥有 **IP** 后，**Linux** 便可以通过路由表或者 **IP** 表规则在三层定位 **bridge0**，此时相当于 **Linux** 拥有了另外一个隐藏的虚拟网卡和 **Bridge** 的隐藏端口相连，这个网卡就是名为 **bridge0** 的通用网络设备，**IP** 可以看成是这个网卡的。当有符合此 **IP** 的数据到达 **bridge0** 时，内核协议栈认为收到了一包目标为本机的数据，此时应用程序可以通过 **Socket** 接收到它。一个更好的对比例子是现实世界中的带路由的交换机设备，它也拥有一个隐藏的 **MAC** 地址，供设备中的三层协议处理程序和管理程序使用。设备里的三层协议处理程序，

对应名为 **bridge0** 的通用网络设备的三层协议处理程序，即寄主 Linux 系统内核协议栈程序。设备里的管理程序，对应 **bridge0** 寄主 Linux 系统里的应用程序。

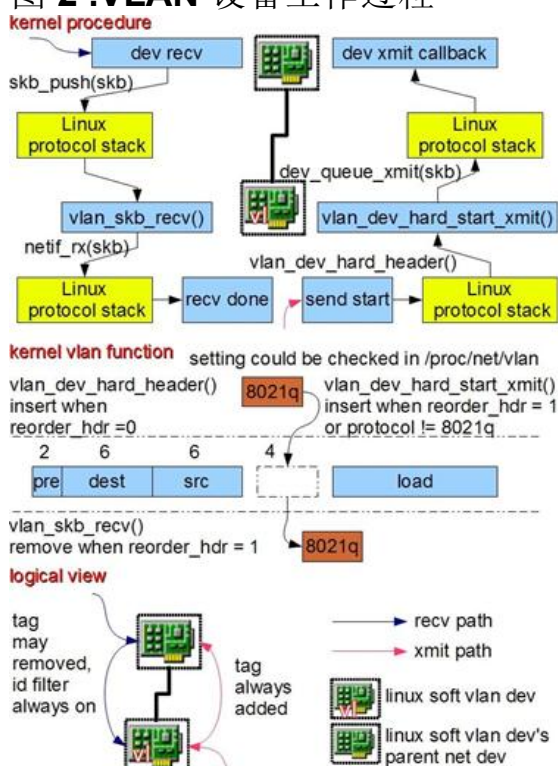
Bridge 的实现当前有一个限制：当一个设备被 **attach** 到 Bridge 上时，那个设备的 IP 会变的无效，Linux 不再使用那个 IP 在三层接受数据。举例如下：如果 **eth0** 本来的 IP 是 **192.168.1.2**，此时如果收到一个目标地址是 **192.168.1.2** 的数据，Linux 的应用程序能通过 **Socket** 操作接受到它。而当 **eth0** 被 **attach** 到一个 **bridge0** 时，尽管 **eth0** 的 IP 还在，但应用程序是无法接受到上述数据的。此时应该把 IP **192.168.1.2** 赋予 **bridge0**。

另外需要注意的是数据流的方向。对于一个被 **attach** 到 Bridge 上的设备来说，只有它收到数据时，此包数据才会被转发到 Bridge 上，进而完成查表广播等后续操作。当请求是发送类型时，数据是不会被转发到 Bridge 上的，它会寻找下一个发送出口。用户在配置网络时经常忽略这一点从而造成网络故障。

VLAN device for 802.1.q

VLAN 又称虚拟网络，是一个被广泛使用的概念，有些应用程序把自己的内部网络也称为 VLAN。此处主要说的是在物理世界中存在的，需要协议支持的 VLAN。它的种类很多，按照协议原理一般分为：**MACVLAN**、**802.1.q VLAN**、**802.1.qbg VLAN**、**802.1.qbh VLAN**。其中出现较早，应用广泛并且比较成熟的是 **802.1.q VLAN**，其基本原理是在二层协议里插入额外的 VLAN 协议数据（称为 **802.1.q VLAN Tag**），同时保持和传统二层设备的兼容性。Linux 里的 VLAN 设备是对 **802.1.q** 协议的一种内部软件实现，模拟现实世界中的 **802.1.q** 交换机。

图 2 .VLAN 设备工作过程



如图所示，Linux 里 **802.1.q VLAN** 设备是以母子关系成对出现的，母设备相当于现实世界中的交换机 **TRUNK** 口，用于连接上级网络，子设备相当于普通接口用于连接

下级网络。当数据在母子设备间传递时，内核将会根据 802.1.q VLAN Tag 进行对应操作。母子设备之间是一对多的关系，一个母设备可以有多个子设备，一个子设备只有一个母设备。当一个子设备有一包数据需要发送时，数据将被加入 VLAN Tag 然后从母设备发送出去。当母设备收到一包数据时，它将会分析其中的 VLAN Tag，如果有对应的子设备存在，则把数据转发到那个子设备上并根据设置移除 VLAN Tag，否则丢弃该数据。在某些设置下，VLAN Tag 可以不被移除以满足某些监听程序的需要，如 DHCP 服务程序。举例说明如下：eth0 作为母设备创建一个 ID 为 100 的子设备 eth0.100。此时如果有程序要求从 eth0.100 发送一包数据，数据将被打上 VLAN 100 的 Tag 从 eth0 发送出去。如果 eth0 收到一包数据，VLAN Tag 是 100，数据将被转发到 eth0.100 上，并根据设置决定是否移除 VLAN Tag。如果 eth0 收到一包包含 VLAN Tag 101 的数据，其将被丢弃。上述过程隐含以下事实：对于寄主 Linux 系统来说，母设备只能用来收数据，子设备只能用来发送数据。和 Bridge 一样，母子设备的数据也是有方向的，子设备收到的数据不会进入母设备，同样母设备上请求发送的数据不会被转到子设备上。可以把 VLAN 母子设备作为一个整体想象为现实世界中的 802.1.q 交换机，下级接口通过子设备连接到寄主 Linux 系统网络里，上级接口同过主设备连接到上级网络，当母设备是物理网卡时上级网络是外界真实网络，当母设备是另外一个 Linux 虚拟网络设备时上级网络仍然是寄主 Linux 系统网络。

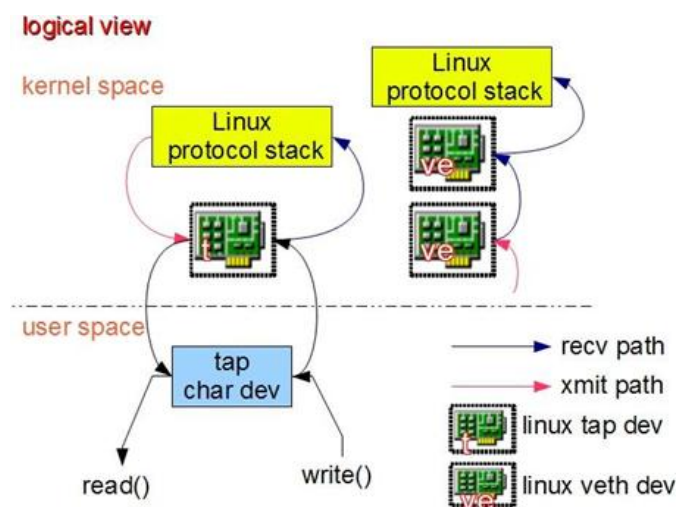
需要注意的是母子 VLAN 设备拥有相同的 MAC 地址，可以把它当成现实世界中 802.1.q 交换机的 MAC，因此多个 VLAN 设备会共享一个 MAC。当一个母设备拥有多个 VLAN 子设备时，子设备之间是隔离的，不存在 Bridge 那样的交换转发关系，原因如下：802.1.q VLAN 协议的主要目的是从逻辑上隔离子网。现实世界中的 802.1.q 交换机存在多个 VLAN，每个 VLAN 拥有多个端口，同一 VLAN 端口之间可以交换转发，不同 VLAN 端口之间隔离，所以其包含两层功能：交换与隔离。Linux VLAN device 实现的是隔离功能，没有交换功能。一个 VLAN 母设备不可能拥有两个相同 ID 的 VLAN 子设备，因此也就不可能出现数据交换情况。如果想让一个 VLAN 里接多个设备，就需要交换功能。在 Linux 里 Bridge 专门实现交换功能，因此将 VLAN 子设备 attach 到一个 Bridge 上就能完成后续的交换功能。总结起来，Bridge 加 VLAN device 能在功能层面完整模拟现实世界里的 802.1.q 交换机。

Linux 支持 VLAN 硬件加速，在安装有特定硬件情况下，图中所述内核处理过程可以被放到物理设备上完成。

TAP 设备与 VETH 设备

TUN/TAP 设备是一种让用户态程序向内核协议栈注入数据的设备，一个工作在三层，一个工作在二层，使用较多的是 TAP 设备。VETH 设备出现较早，它的作用是反转通讯数据的方向，需要发送的数据会被转换成需要收到的数据重新送入内核网络层进行处理，从而间接的完成数据的注入。

图 3 .TAP 设备和 VETH 设备工作过程



如图所示，当一个 TAP 设备被创建时，在 Linux 设备文件目录下将会生成一个对应 char 设备，用户程序可以像打开普通文件一样打开这个文件进行读写。当执行 `write()` 操作时，数据进入 TAP 设备，此时对于 Linux 网络层来说，相当于 TAP 设备收到了一包数据，请求内核接受它，如同普通的物理网卡从外界收到一包数据一样，不同的是其实数据来自 Linux 上的一个用户程序。Linux 收到此数据后将根据网络配置进行后续处理，从而完成了用户程序向 Linux 内核网络层注入数据的功能。当用户程序执行 `read()` 请求时，相当于向内核查询 TAP 设备上是否有需要被发送出去的数据，有的话取出到用户程序里，完成 TAP 设备的发送数据功能。针对 TAP 设备的一个形象的比喻是：使用 TAP 设备的应用程序相当于另外一台计算机，TAP 设备是本机的一个网卡，他们之间相互连接。应用程序通过 `read()/write()` 操作，和本机网络核心进行通讯。

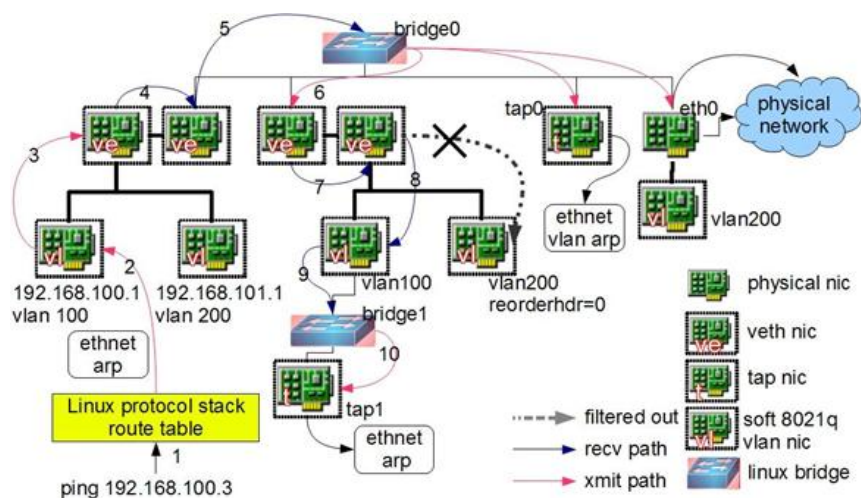
VETH 设备总是成对出现，送到一端请求发送的数据总是从另一端以请求接受的形式出现。该设备不能被用户程序直接操作，但使用起来比较简单。创建并配置正确后，向其一端输入数据，VETH 会改变数据的方向并将其送入内核网络核心，完成数据的注入。在另一端能读到此数据。

网络设置举例说明

为了更好的说明 Linux 网络设备的用法，下面将用一系列的例子，说明在一个复杂的 Linux 网络元素组合出的虚拟网络里，数据的流向。网络设置简介如下：一个中心 Bridge: `bridge0` 下 attach 了 4 个网络设备，包括 2 个 VETH 设备，1 个 TAP 设备 `tap0`，1 个物理网卡 `eth0`。在 VETH 的另外一端又创建了 VLAN 子设备。Linux 上共存在 2 个 VLAN 网络，既 `vlan100` 与 `vlan200`。物理网卡和外部网络相连，并且在它之下创建了一个 VLAN ID 为 200 的 VLAN 子设备。

从 `vlan100` 子设备发送 ARP 报文

图 4 .ARP from `vlan100` child device

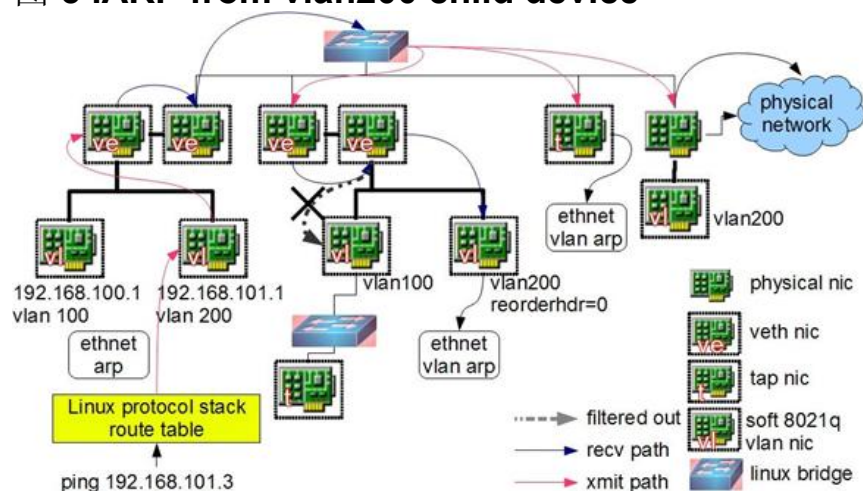


如图所示，当用户尝试 ping 192.168.100.3 时，Linux 将会根据路由表，从 vlan100 子设备发出 ARP 报文，具体过程如下：

- 1) 用户 ping 192.168.100.3
- 2) Linux 向 vlan100 子设备发送 ARP 信息。
- 3) ARP 报文被打上 VLAN ID 100 的 Tag 成为 ARP@vlan100，转发到母设备上。
- 4) VETH 设备将这一发送请求转变方向，成为一个需要接受处理的报文送入内核网络模块。
- 5) 由于对端的 VETH 设备被加入到了 bridge0 上，并且内核发现它收到一个报文，于是报文被转发到 bridge0 上。
- 6) bridge0 处理此 ARP@vlan100 信息，根据 TCP/IP 二层协议发现是一个广播请求，于是向它所知道的所有端口广播此报文，其中一路进入另一对 VETH 设备的一端，一路进入 TAP 设备 tap0，一路进入物理网卡设备 eth0。此时在 tap0 上，用户程序可以通过 read()操作读到 ARP@vlan100，eth0 将会向外界发送 ARP@vlan100，但 eth0 的 VLAN 子设备不会收到它，因为此数据方向为请求发送而不是请求接收。
- 7) VETH 将请求方向转换，此时在另一端得到请求接受的 ARP@vlan100 报文。
- 8) 对端 VETH 设备发现有数据需要接受，并且自己有两个 VLAN 子设备，于是执行 VLAN 处理逻辑。其中一个子设备是 vlan100，与 ARP@vlan100 吻合，于是去除 VLAN ID 100 的 Tag 转发到这个子设备上，重新成为标准的以太网 ARP 报文。另一个子设备由于 ID 不吻合，不会得到此报文。
- 9) 此 VLAN 子设备又被 attach 到另一个桥 bridge1 上，于是转发自己收到的 ARP 报文。
- 10) bridge1 广播 ARP 报文。
- 11) 最终另外一个 TAP 设备 tap1 收到此请求发送报文，用户程序通过 read()可以得到它。

从 vlan200 子设备发送 ARP 报文

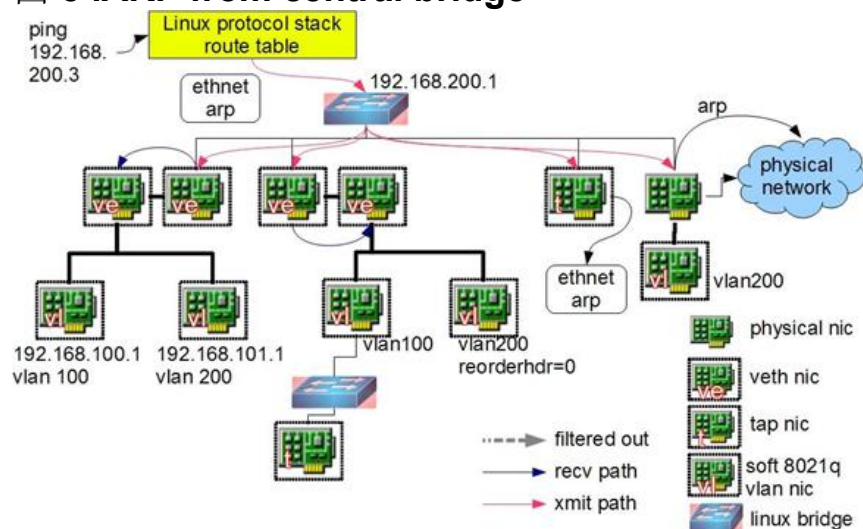
图 5 .ARP from vlan200 child device



和前面情况类似，区别是 VLAN ID 是 200，对端的 vlan200 子设备设置为 `reorder_hdr = 0`，表示此设备被要求保留收到的报文中的 VLAN Tag。此时子设备会收到 ARP 报文，但是带了 VLAN ID 200 的 Tag，既 `ARP@vlan200`。

从中心 bridge 发送 ARP 报文

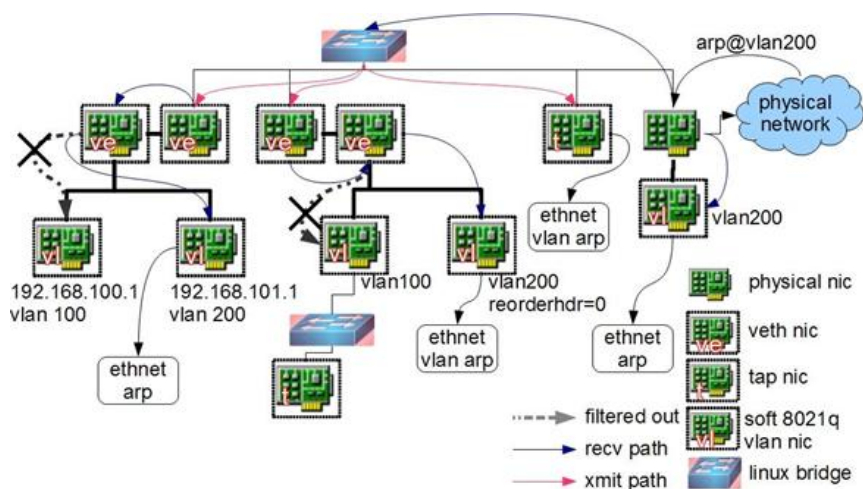
图 5 .ARP from central bridge



当 `bridge0` 拥有 IP 时，通过 Linux 路由表用户程序可以直接将 ARP 报文发向 `bridge0`。这时 `tap0` 和外部网络都能收到 ARP，但 VLAN 子设备由于 VLAN ID 过滤的原因，将收不到 ARP 信息。

从外部网络向物理网卡发送 ARP@vlan200 报文

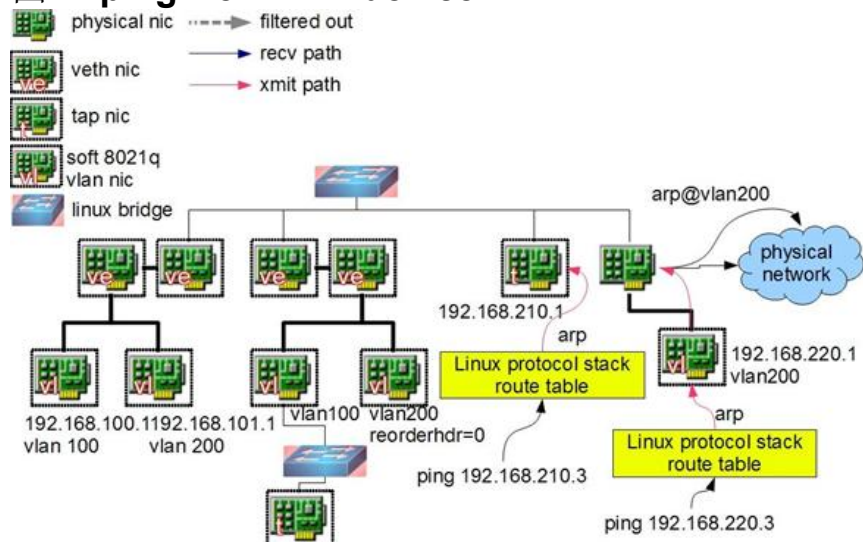
图 6 .ARP from external network



当外部网络连接在一个支持 VLAN 并且对应端口为 vlan200 时，此情况会发生。此时所有的 VLAN ID 为 200 的 VLAN 子设备都将接受到报文，如果设置 reorder_hdr=0 则会收到带 Tag 的 ARP@vlan200。

从 TAP 设备以 ping 方式发送 ARP

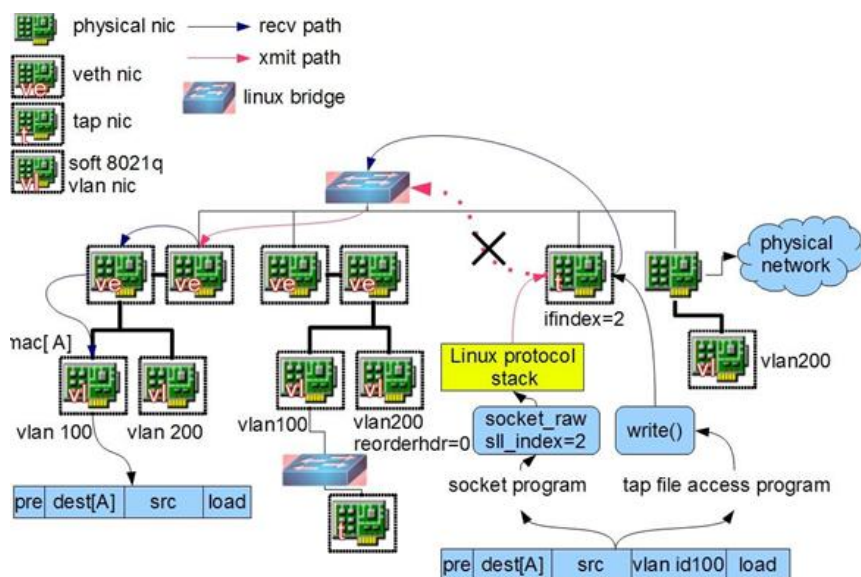
图 7 .ping from TAP device



给 tap0 赋予 IP 并加入路由，此时再 Ping 其对应网段的未知 IP 会产生 ARP 发送请求。需要注意的是此时由于 tap0 上存在的是发送而不是接收请求，因此 ARP 报文不会被转发到桥上，从而什么也不会发生。图中右边画了一个类似情况：从 vlan200 子设备发送 ARP 请求。由于缺少 VETH 设备反转请求方向，因此报文也不会被转发到桥上，而是直接通过物理网卡发往外部网络。

以文件操作方式从 TAP 设备发送报文

图 8 .file operation on TAP device



用户程序指定 **tap0** 设备发送报文有两种方式：**socket** 和 **file operation**。当用 **socket_raw** 标志新建 **socket** 并指定设备编号时，可以要求内核将报文从 **tap0** 发送。但和前面的 **ping from tap0** 情况类似，由于报文方向问题，消息并不会被转发到 **bridge0** 上。当用 **open()** 方式打开 **tap** 设备文件时，情况有所不同。当执行 **write()** 操作时，内核认为 **tap0** 收到了报文，从而会触发转发动作，**bridge0** 将收到它。如果发送的报文如图所示，是一个以 **A** 为目的地的携带 **VLAN ID 100 Tag** 的单点报文，**bridge0** 将会找到对应的设备进行转发，对应的 **VLAN** 子设备将收到没有 **VLAN ID 100 Tag** 的报文。

Linux 上配置网络设备命令举例

以 Redhat6.2 红帽 Linux 发行版为例，如果已安装 **VLAN** 内核模块和管理工具 **vconfig**，TAP/TUN 设备管理工具 **tunctl**，那么可以用以下命令设置前述网络设备：

- 创建 Bridge: `brctl addbr [BRIDGE NAME]`
- 删除 Bridge: `brctl delbr [BRIDGE NAME]`
- attach 设备到 Bridge: `brctl addif [BRIDGE NAME] [DEVICE NAME]`
- 从 Bridge detach 设备: `brctl delif [BRIDGE NAME] [DEVICE NAME]`
- 查询 Bridge 情况: `brctl show`
- 创建 VLAN 设备: `vconfig add [PARENT DEVICE NAME] [VLAN ID]`
- 删除 VLAN 设备: `vconfig rem [VLAN DEVICE NAME]`
- 设置 VLAN 设备 flag: `vconfig set_flag [VLAN DEVICE NAME] [FLAG] [VALUE]`
- 设置 VLAN 设备 qos:
 - `vconfig set_egress_map [VLAN DEVICE NAME] [SKB_PRIORITY] [VLAN_QOS]`
 - `vconfig set_ingress_map [VLAN DEVICE NAME] [SKB_PRIORITY] [VLAN_QOS]`
- 查询 VLAN 设备情况: `cat /proc/net/vlan/[VLAN DEVICE NAME]`

- 创建 VETH 设备: `ip link add link [DEVICE NAME] type veth`
- 创建 TAP 设备: `tunctl -p [TAP DEVICE NAME]`
- 删除 TAP 设备: `tunctl -d [TAP DEVICE NAME]`
- 查询系统里所有二层设备, 包括 VETH/TAP 设备: `ip link show`
- 删除普通二层设备: `ip link delete [DEVICE NAME] type [TYPE]`

小结

综上所述, Linux 已经提供一套基本工具供用户创建出各种内部网络, 利用这些工具可以方便的创建出特定网络给应用程序使用, 包括云计算中的初级内部虚拟网络。

参考资料

学习

- [Vconfig Man Page](#), vconfig 工具帮助文档。
- [802.1Q VLAN implementation for Linux](#), Linux 中 VLAN 模块如何实现的文档说明。
- [IPROUTE2 Utility Suite Howto](#), Linux 里的 IP 工具使用说明。
- [Virtual networking in Linux](#), 以虚拟化应用为中心讲述主流的虚拟网络技术, 主要以 openvswitch 为例。
- [Linux BRIDGE-STP-HOWTO](#), Linux 中的 bridge 设备使用说明。
- [Linux Kernel Networking \(Network Overview\) by Rami Rosen](#), Linux 内核里的各种网络概念的含义, 目的及用法简单介绍。
- 在 [developerWorks Linux 专区](#) 寻找为 Linux 开发人员 (包括 [Linux 新手入门](#)) 准备的更多参考资料。

讨论

- 加入 [developerWorks 中文社区](#)。查看开发人员推动的博客、论坛、组和维基, 并与其他 developerWorks 用户交流。



IBM Bluemix 资源中心
文章、教程、演示, 帮助您构建、部署和管理云应用。



developerWorks 中文社区
立即加入来自 IBM 的专业 IT 社交网络。



IBM 软件资源中心
免费下载、试用软件产品, 构建应用并提升技能。