基于 LKM 的网络设备接口编程实现

董阴晴

（电子科技大学通信抗干扰技术国家级重点实验室， 四川 成都 611731）

摘 要： 本文实现了一个可动态加载至 Linux 内核的网络设备接口模块。该模块利用 Netfilter 框架注册 VNI 发送函数，将 IP 报文封装在 VNI 数据帧中并调用以太网口发送。相应地，该模块在内核中注册 VNI 接收函数，去除 VNI 头部，将数据包上交给 IP 模块。

关键词： Linux 内核模块编程；sk_buff；Netfilter；网络设备接口

1 引言

如今，TCP/IP 协议栈已经普遍运行在了大多数的计算机设备之上。然而由于科研、生产的特殊需求，在 TCP/IP 协议栈的基础之上添加定制特定功能的协议栈的需求也日渐流行。

在 Linux 系统中，TCP/IP 协议栈位于内核空间中[1]。因此若要在协议栈中添加新协议必须对内核进行修改。然而，由于 Linux 内核是一种宏内核，不同协议功能之间相互耦合，内核的可扩展性较差。为了克服上述缺点，Linux 引入了内核模块机制，其全称为动态可加载内核模块(Loadable Kernel Module，LKM)[2]。通过动态加载或卸载外部模块，开发者可以轻松地在内核中添加或删除特定功能。因此编写外部协议功能模块是在 linux 系统上开发新协议的有效手段之一。

本文的主要工作为实现了一个虚拟网络设备接口(Virtual Network Device Interface，VNI)模块。其主要功能为在 IP 实体与 MAC 实体之间添加了一个 VNI 实体。发送方在 IP 包前封装一个 VNI 头部，并用 MAC 帧（802.3 帧）承载发送，帧格式如图 1-1，其中 VNI 头部的 seq 字段从零开始递增。接收方去掉 VNI 头部上交给 IP 模块。

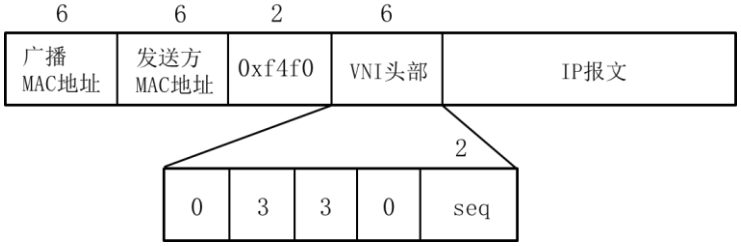


图 1-1 帧格式

本文剩余部分安排如下：第二节实现流程详细介绍了网络设备接口模块的设计原理、方案以及软件实现过程，第三节测试方法与结果分析介绍了网络设备接口模块的使用测试方法、测试环境并对测试结果进行详细地分析。

2 实现流程

2.1 设计原理

在 Linux 系统中，内核模块无法单独运行，但当其被载入内核后其代码与事先编译进内核的代码没有区别[3]。因此内核模块可以使用内核中已有的变量、数据结构、函数等。因此 VNI 模块的开发涉及许多内核中的数据结构与框架，主要包括：sk_buff、Netfilter 框架、内核收发数据包流程等。下文将逐一介绍这些数据结构与框架的基本原理。

sk_buff

sk_buff（以下简称 skb）是 Linux 网络子系统中的核心数据结构，贯穿了整个网络协议栈[4]。其由报文数据与管理数据两部分组成，其中报文数据保存了实际在网络中传输的数据；管理数据是内核中协议之间交换的控制信息。head、end、data、tail 指针是 skb 中的核心字

段。这四个指针的指向位置关系如下图：

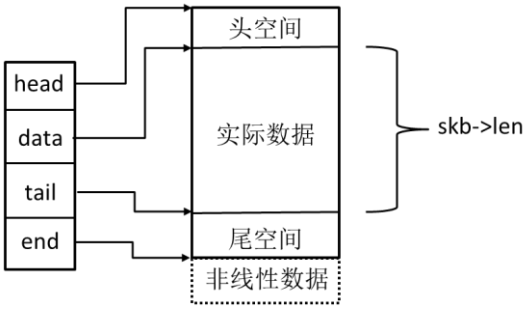


图 2-1 head、end、data、tail 指针

其中 data 与 tail 指向实际在网络中传输的数据包的首尾；由 head 与 end 界定的区域被称为线性数据区，当实际传输的数据包的大小大于线性数据区时，超出的部分将会被存储在非线性区。len 是指实际传输的数据包的大小，当非线性数据区大小为零时，len 即为 data-tail。

数据包在协议间的交换与这四个指针关系密切。在数据包穿越协议栈的过程中，内核不会反复拷贝 skb，而仅仅是移动上述四个指针[4]。从用户进程通过 sockt（内核为用户空间应用程序预留的 API）将数据递交给内核开始到完成对数据的层层封装准备发送为止，上述四个指针的变化如下[5]：

首先，当内核从 socket 获取用户的数据后，会调用 alloc_skb 函数，为数据分配一个 skb。此时 head 指针、data 指针、tail 指针都是指向内存中的同一个地方，如下图

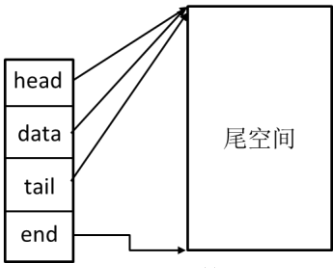


图 2-2 alloc_skb 函数返时的 skb

接着，内核调用 skb_reserve 函数，使得 data 指针和 tail 指针同时向下移动，为各层协议头的添加预留空间，如下图

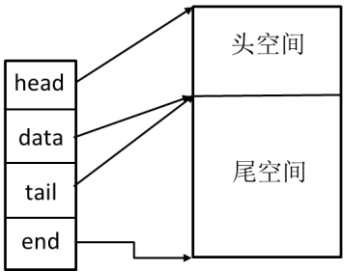


图 2-3 skb_reserve 函数返时的 skb

然后，内核调用 skb_put 函数，使得 tail 指针向下移动，用以存放用户数据，如下图

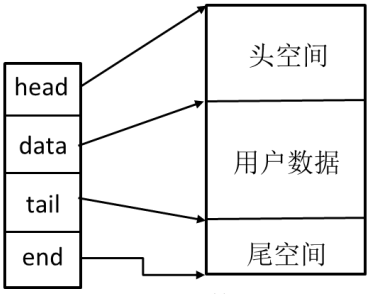


图 2-4 skb_put 函数返时的 skb

最后，内核调用 skb_push 函数，使得 data 指针向上移动，用以添加各层协议头，如下

图：

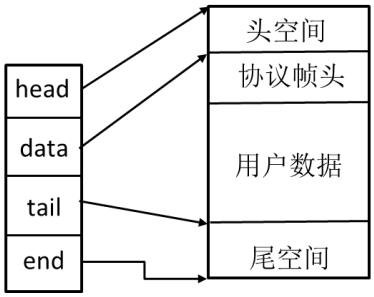


图 2-5 skb_push 函数返时的 skb

相应得，在接收方，内核会调用 skb_pull 函数来使得 data 指针向下移动去掉协议头。

除了 head、end、data、tail 指针以及实际数据包长度 len 外，skb 中较为重要且与本文工作密切相关的字段被罗列如下[6–8]：

1. dev 字段：用来指示发送或接收数据包的网络设备，通常为本机网卡。
2. csum 字段：用于记录用户空间传递的数据包即 TCP 或 UDP 载荷的校验和。
3. ip_summed 字段：该字段的取值决定了协议栈与网卡驱动计算数据包的校验和的方式。在发送过程中，若该字段被置为 0(CHECKSUM_NONE)表示协议栈已经完成了数据包，包括伪头部与载荷的校验和计算。硬件无须计算校验和。在接收过程中，若该字段被置为 0(CHECKSUM_NONE)表示硬件未对数据包进行校验，需要上层协议重新校验数据包。
4. pkt_type 字段：用于指示数据包的类型，包括 PACKET_HOST-表示该数据包为发给本机的数据包，PACKET_BOARDCAST-表示该包为广播数据包等等。
5. protocol 字段：用于指示数据包的协议类型如 IP、ARP 等。

Netfilter

Netfilter 是 Linux 内核中的包过滤框架。Linux 系统的 NAT、防火墙等功能都是通过 Netfilter 框架实现的。其提供了五个钩子(hook)以便内核模块可以在网络协议栈内部的不同位置上注册回调函数。当数据包经过检测点(hook)时，内核将会调用相应的回调函数用以处理数据包[9]。Netfilter 提供的五个检测点(hook)在协议栈中的位置与名字如下图中红字所示：

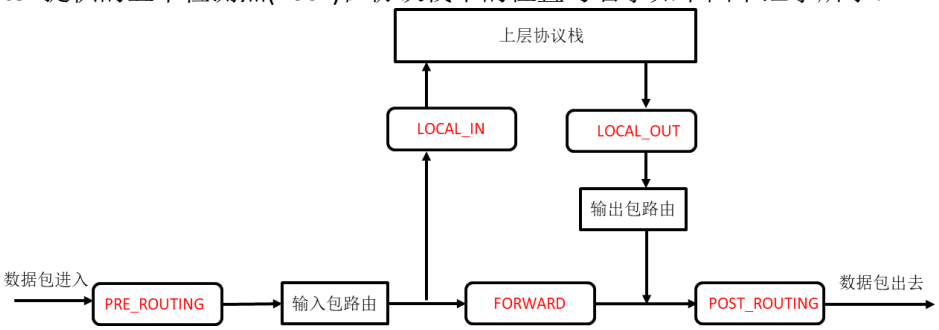


图 2-6 Netfilter hook 点位置

其中上层协议栈是指网络层及以上的协议，如 TCP、UDP、ICMP 协议等。其中上层协议递交给 IPv4 的数据包封装 IP 头部后会经过 LOCAl_OUT 检测点。IP 包经路由确定出端口后会经过 POST_ROUTING 检测点。

内核对经回调函数处理后的数据包的处理包括丢弃或继续正常传输或将数据包提交给用户空间等[10]，其中最常见的三种处理方式以及其标识如下：

1. 丢弃数据包，释放为其分配的任何资源，不再继续传输，用 NF_DROP 标识。
2. 继续正常传输报文，用 NF_ACCEPT 标识。
3. 不再继续传输数据包，但不会释放数据包占用的资源，用 NF_STOLEN 标识。

ptype_base

ptype_base 是一个较为重要的结构体。ptype_base 是一个哈希表，主要用于为数据包匹配合适的接收句柄函数[11]。当网卡驱动通过软中断函数将数据包传递给内核后，内核会调用 netif_receive_skb()函数，在 ptype_base 中查找用于处理数据包的相关函数，例如 ip_rcv() arp_rcv() ipv6_rcv()等函数，如下图，然后将数据包上交至对应的协议，若查找不到对应的函

数，则直接丢弃数据包。在内核中，开发者可以使用 dec_add_pack()函数在 ptype_base 中注册新类型协议的处理函数[12]。

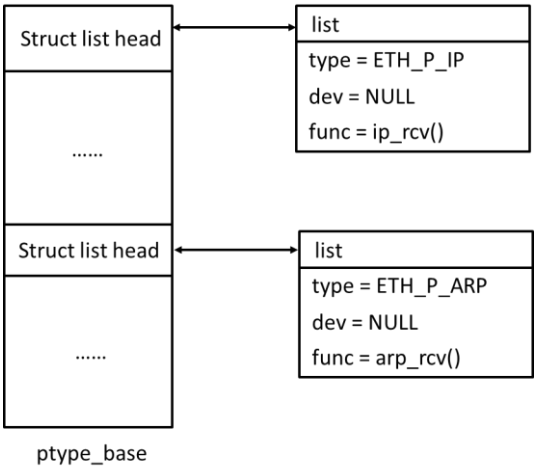


图 2-7 ptye_base 示意图

2.2 设计方案

VNI 模块包含三个核心功能：

1. 发送功能：将 IP 数据包封装在 VNI 帧中，并调用以太网络设备（以下简称 eth0）发送数据帧。
2. 接收功能：将 eth0 接收到的数据帧解封后上交给内核 IP 模块。
3. 统计功能：统计 vni 的工作状态信息，例如发送与接收的数据帧数目等。

本文将 VNI 模块划分为三个子模块——发送子模块、接收子模块与统计子模块分别实现了上述功能。三个模块的具体工作流程如下：

发送子模块

本文利用 Netfilter 框架实现了 VNI 模块的发送功能。在发送方，VN 模块需要截获内核发出的所有 IP 数据包并将其封装在 VNI 帧中。而 Netfilter 恰好满足了这样的需求。VNI 发送子模块的具体流程如下图，其中

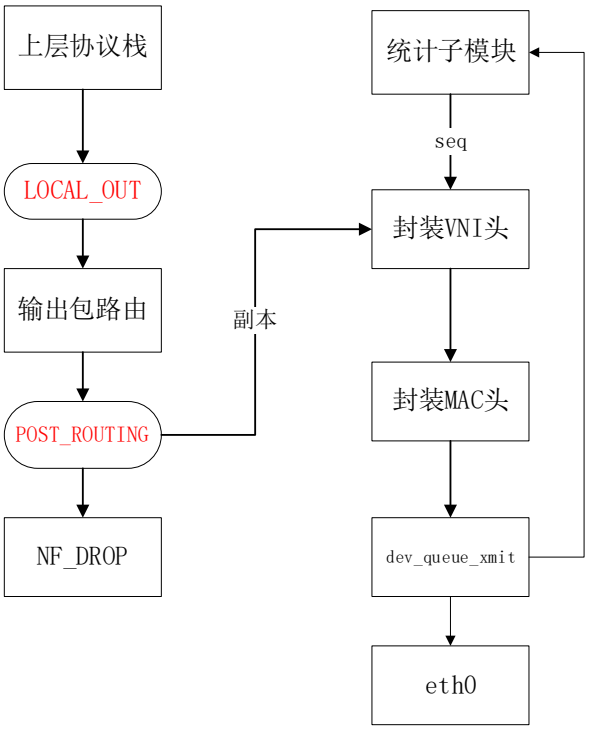


图 2-8 VNI 发送子模块

1. 通过在 POST_ROUTING 检测点注册回调函数，VNI 发送子模块会拷贝一份即将发送的 IP 数据包。由于此时内核已经完成了对 IP 数据包的路由，因此 skb 中的 dev 字段

已经被指定为实际发送数据包的设备。

2. 当数据包通过 POST_ROUTING 检测点时，内核还未将其封装在 MAC 帧中，skb 的 data 指针指向 IP 头部。因此获得数据包的副本后，VNI 发送子模块移动 data 指针在 IP 数据包前添加 VNI 头部。同时，统计子模块会将当前已发送的分组数目传递给 VNI 发送子模块，实现 VNI 头部中序号 seq 字段的填充。
3. 在添加 VNI 头部之后，VNI 发送子模块需要用 MAC 帧封装 VNI 帧。其中，MAC 帧头部的长度类型域的取值为 0xf4f0；源 MAC 地址为本机网卡即 eth0 的 MAC 地址，目的 MAC 地址为广播 MAC 地址。
4. 封装 MAC 帧头之后，发送子模块将数据包加入到网卡的发送队列[13–15]。
5. 在原先 IP 数据包通过 POST_ROUTING 检测点后，回调函数返回 NF_DROP 通知内核将原数据包丢弃，以免接收方收到重复的 IP 报文。

接收子模块

本文通过在 ptype_base 中注册句柄 vni_rcv()函数处理网卡接收到的 vni 帧[16,17]。vni 帧的处理过程如下图，其中

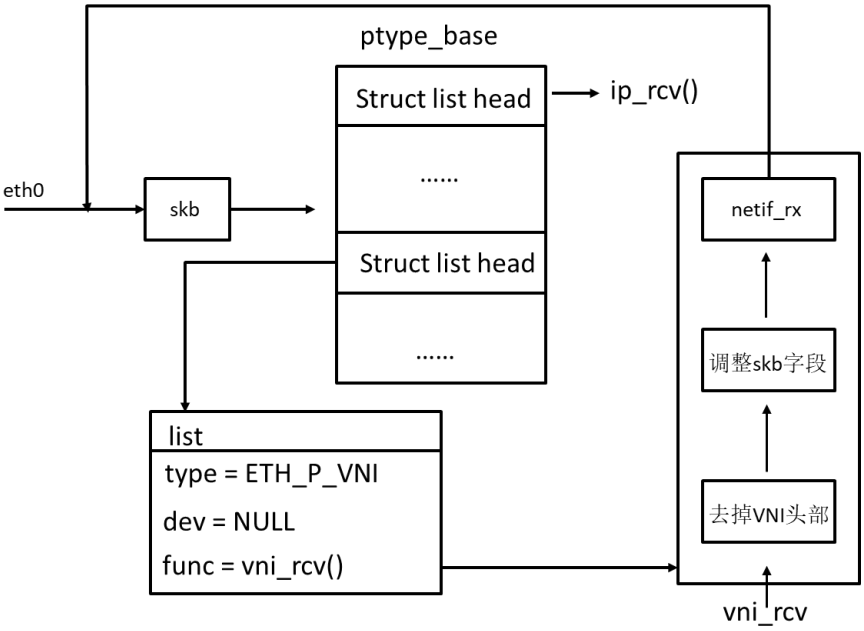


图 2-9 VNI 接收子模块

1. 当内核将 skb 上交给 vni 模块时，skb 的 data 指针指向 vni 头部。接收子模块移动 data 指针去掉 vni 头部。
2. 去除 vni 头部后，接收子模块调整 skb 中的控制字段信息，例如将协议类型由 VNI 修改为 IP，根据 IP 头部的目的 IP 地址将数据帧类型由广播帧修改为发往本机数据帧或发往其他主机的数据帧等。
3. 处理完成后，接收子模块将 skb 添加到接收队列 input_pkt_queue，通知内核将数据包上交给 IP 模块[18]。

统计子模块

VNI 模块中还包含一个简易的统计子模块。除了统计 VNI 模块已成功发送或接收的数据帧数目外，统计子模块还计算了 VNI 模块的发送、接收速率同时还会定时打印上述信息。

2.3 软件实现

站在软件实现的角度，VNI 模块可以被划分为如下几个部分：

1. 初始化：完成将 VNI 模块加载到内核中所必须的工作，包括发送回调函数的注册、接收句柄的注册以及统计子模块的初始化
2. 实际功能：上述三个模块功能的实现，包括发送、接收子模块的入口与出口、数据包处理的实现，以及统计子模块速率统计定时打印功能的实现。
3. 注销：完成从内核中卸载 VNI 模块所必须的工作。

下面将逐一介绍各个部分的软件实现原理。

初始化

当 VNI 加载到内核时，其会完成以下工作

1. 初始化一个 nf_hook_ops 结构体，包括其 hook、hooknum、pf、priority 字段[19]，其中 hook 字段用于指示待注册的回调函数即发送子模块；hooknum 用于指示注册回调函数的检测点，如图 2-8 本文选择 NF_INET_POST_ROUTING 为回调函数的检测点；pf 用于指示协议族，由于本文中网络层协议为 IPv4，故 pf 取值为 PF_INET；priority 用于指示该结构体的优先级，决定其在 nf_hooks[pf][hooknum]链表上的位置[20]，本文中 priority 取 NF_IP_PRI_FIRST。完成 nf_hook_ops 结构体的初始化后调用 nf_register_net_hook()函数，完成回调函数的注册。
2. 初始化一个 packet_type 结构体，包括.type 与.func，用于在 ptype_base 中注册 vni_rcv 函数。其中.type 用于指示数据帧协议类型，本文中取 0xf4f0 即 VNI 协议编号，赋值时需要将数据转为网络字节序。.func 用于指示处理数据帧的句柄，即 vni_rcv()函数；然后调用 dev_add_pack()函数将句柄注册到 ptype_base 表中。
3. 初始化一个 timer_list 结构体，为统计子模块分配一个定时器。本文中使用 timer_setup()函数初始化定时器。初始化时，定时回调函数将作为参数传入 timer_setup()函数，用于指示定时器超时后被调用的函数[21]。timer_list 的 expires 字段指示定时器的定时时间，通常用内核全局变量 jiffies、HZ 初始化，其中 jiffies 是自开机起系统的滴答数，HZ 是系统滴答的频率。完成 timer_list 的初始化后，调用 add_timer()函数激活定时器。

发送子模块

1. 发送子模块的入口：如图 2-8 所示，发送子模块的入口为 NF_INET_POST_ROUTING 检测点。当内核完成输出包路由后，IP 数据包将会被送往 VNI 发送子模块。
2. 发送子模块对数据包的处理：如图 2-8 所示，发送子模块在 IP 数据包前封装 VNI 头部与 MAC 头部。在这个过程中，发送子模块将调用 skb_push()函数将 data 指针向上移动以便添加 VNI 与 MAC 帧头。其中在添加 vni 的序号字段以及 MAC 头部的长度类型域时，发送子模块需要将相关的数据转化为网络字节序。若 skb 头空间的大小不足以容纳新增的帧头，则调用 skb_copy_expand()函数在复制 skb 的同时扩展 skb 的头空间。
3. 发送子模块出口：如图 2-8 所示，完成数据包的封装后，发送子模块调用 dev_queue_xmit()函数发送数据包。

接收子模块

1. 接收子模块入口：如图 2-9 所示，当内核接收到 ETH_P_VNI 类型的数据帧时，数据帧被送往 VNI 接收子模块。
2. 接收子模块对数据包的处理：如图 2-9 所示，接收子模块通过 skb_pull()函数将 data 指针向下移动，去掉 vni 头部，然后将 skb 的 protocol 字段修改为 0x0800（网络字节序），pkt_type 字段修改为 PACKET_HOST。
3. 接收子模块出口：完成对数据帧的处理后，接收子模块调用 netif_rx()函数将 skb 重新送往接收队列。此时，由于 skb 的协议类型被更改为 IP 协议编号，内核会根据 ptype_base 表调用 ip_rcv()函数处理 skb，进而将数据包传递给 IP 模块。

统计子模块

1. 收发包数目统计：每当成功收发一帧，相应的统计变量执行一次自增；.
2. 收发包速率统计及定时打印：统计子模块在定时回调函数中调用 mod_timer()函数修改定时器的 expires 字段以实现定时器的周期运转[22]。同时，定时回调函数计算前后两次定时结束的收发包数量差，计算其与定时周期的比值即为收发包的速率。

注销

当从内核中卸载 VNI 模块时，其会完成以下工作：

1. 调用 nf_unregister_net_hook()函数，注销 nf_hook_ops 结构体
2. 调用 dev_remove_pack()函数，在 ptype_base 表中删除 VNI 相关的表项
3. 调用 del_timer()函数，删除定时器。

3 测试方法与结果分析

3.1 测试环境

VNI 模块的开发与测试环境为 Ubuntu16.04.1，Linux 内核版本为 4.15.0，如下图。

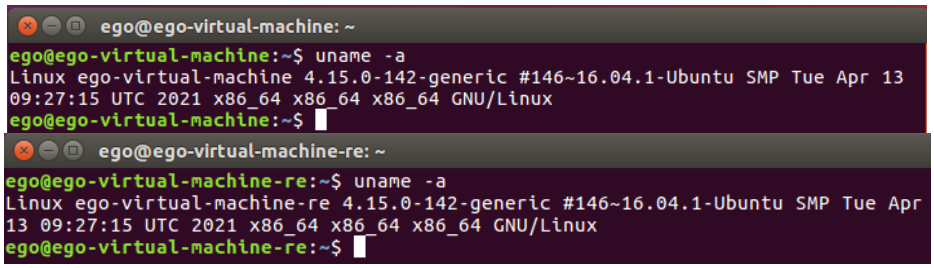


图 3-1 两台虚拟机的系统版本与内核版本

3.2 测试方法

本文使用两台虚拟机 vm1 与 vm2 测试 VNI 模块功能。两台虚拟机上均加载了 VNI 模块，如下图。由 vm1 向 vm2 发送 ping 报文。若 vm1 能够接收到来自 vm2 的回复，则说明 VNI 模块正确地发送和接收 VNI 数据帧。

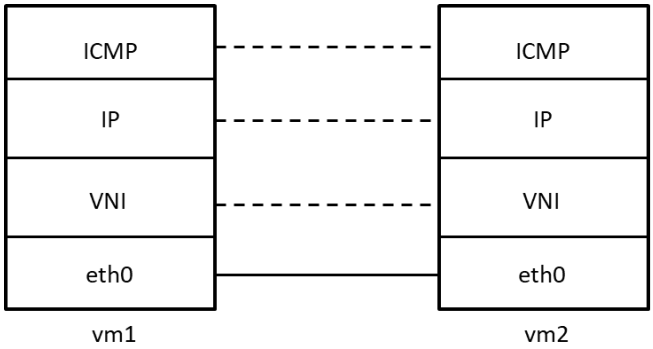


图 3-2 测试方法

其中 vm1 的 IP 地址为 10.168.1.207，转化为十六进制格式：0aa801cf；vm2 的 IP 地址为 10.168.1.186，转化为十六进制格式：0aa801cfba，如下图

图 3-3 vm1 的 IP 地址

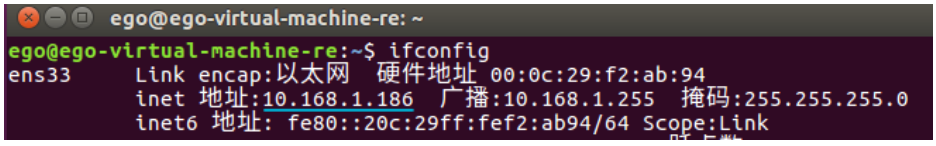
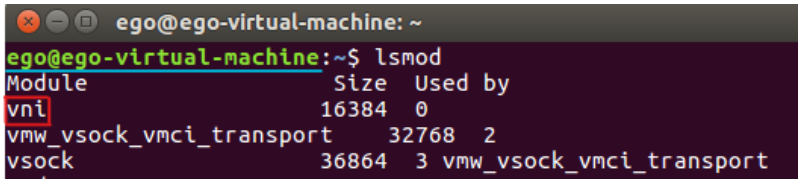


图 3-4 vm2 的 IP 地址

3.3 测试流程与结果分析

1. 编译源文件 vni.c，生成 vni.ko 文件，并使用 insmod 命令将 VNI 模块加载到内核中，使用 lsmod 命令查看当前内核已加载的模块列表，可以看到 VNI 模块已经被成功加载到内核中。



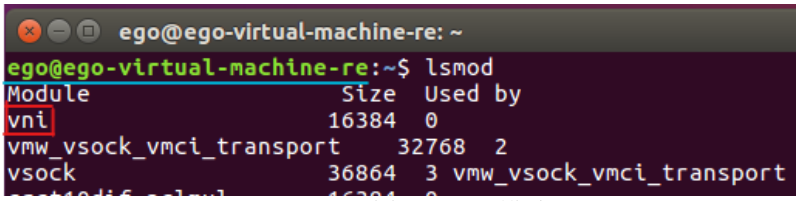


图 3-5 成功加载 VNI 模块

2. 在 vm1 中输入命令 ping 10.168.1.186 -c 100 向 vm2 发送 100 个回显请求报文，随后可以观察到如下现象：

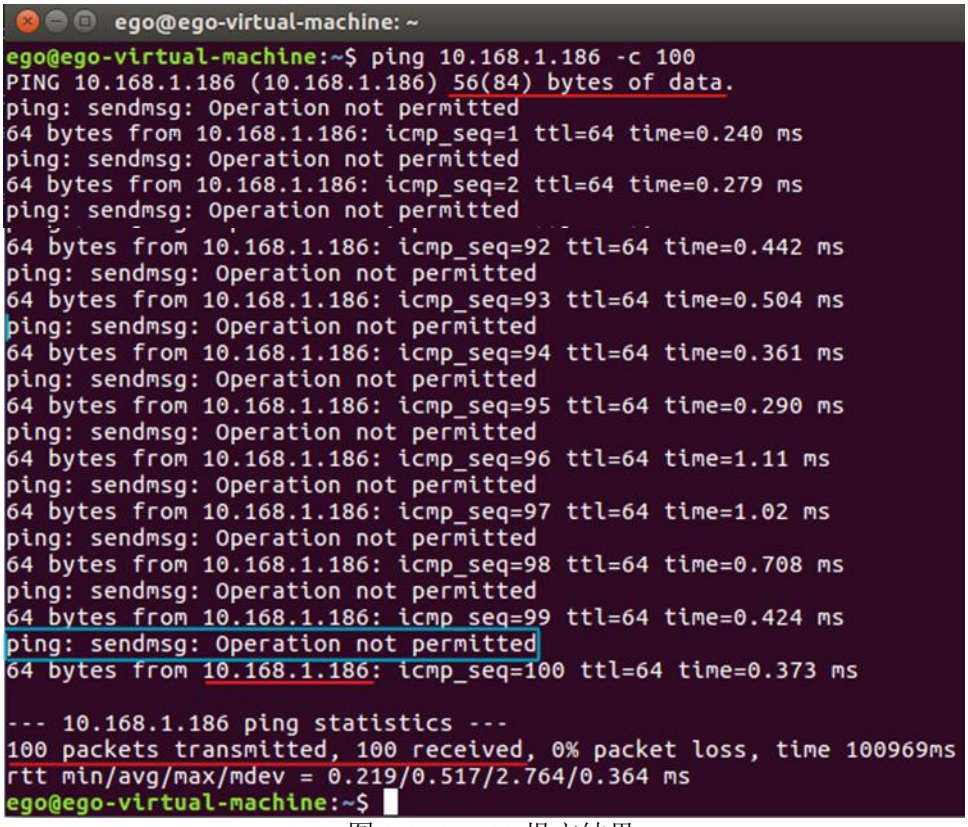


图 3-6 ping100 报文结果

a) 如红线的部分所示，ping 命令打印的统计信息显示，vm1 成功的向 vm2 发送了 100 个 ping 报文并收到了来自 vm2 的 100 个应答报文。这说明 VNI 的发送与接收子模块可以正常工作

b) 在发送回显请求报文的过程中，ping 命令提示 sendmsg: Operation not permitted，如上图中的蓝色方框所示。出现这一现象的原因为：VNI 发送子模块获得原数据包的拷贝后，其会通知内核将原先的数据包丢弃，并释放为该数据包分配所有资源。

c) 在一次 ping 交互中，vm1 与 vm2 上的 VNI 模块都分别完成了一次 VNI 数据帧的发送与接收。

   i. 首先，vm1 的回显请求报文被封装在 IP 数据包中；随后内核将 IP 数据包递交给 VNI 发送子模块，为其封装 VNI 头部并调用以太网设备发送

   ii. 接着，vm2 接收到 VNI 数据帧后，VNI 接收子模块去掉 VNI 头部，将 IP 数据包上交给内核 IP 模块；IP 模块去掉 IP 头后将数据上交给 ICMP 模块；

   iii. 然后，ICMP 模块根据回显请求报文构造应答报文；同样的，内核将 ICMP 报文封装在 IP 数据包中，由 vm2 上的 VNI 发送子模块处理发送。

   iv. 最后，vm1 接收到 VNI 数据帧，交由 VNI 接收子模块处理，层层解封装，上交给 ICMP 模块。

3. 在 vm1 向 vm2 发送 ping 报文的过程中，使用 wirshark 软件抓取经过网卡的数据包，过滤规则分别为：①eth.dst == ff:ff:ff:ff:ff:ff and eth.src == 00:0c:29:db:39:41；②eth.dst

== ff:ff:ff:ff:ff:ff and eth.src == 00:0c:29:f2:ab:94，抓包结果如下图：

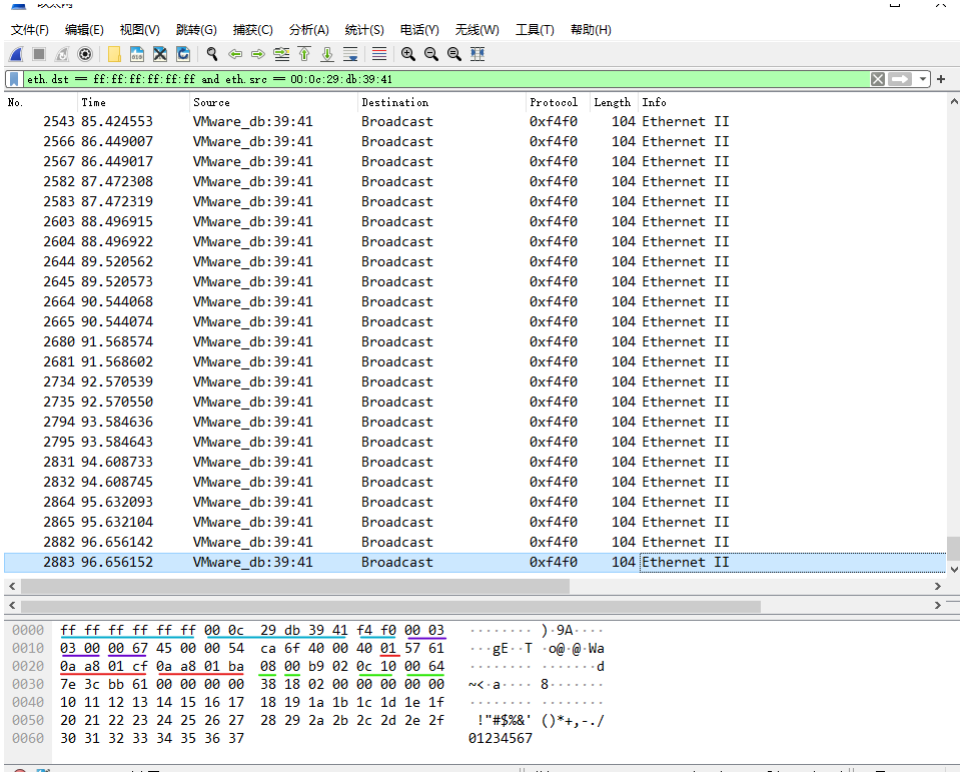


图 3-7 vm1 发出的包

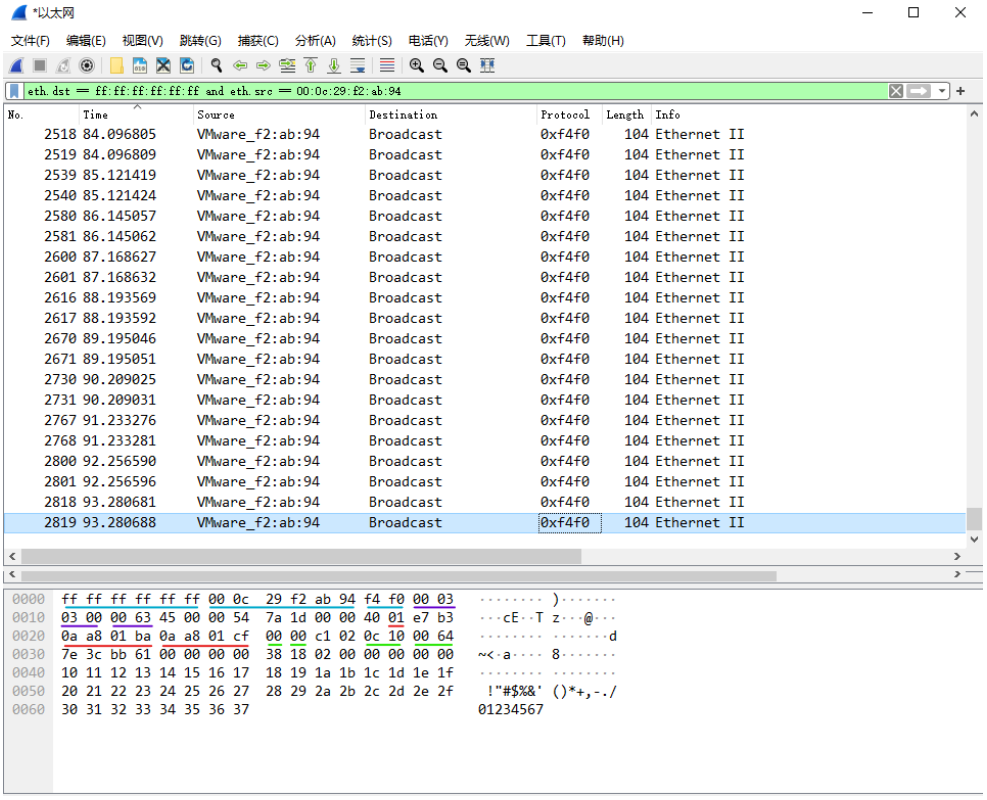


图 3-8 vm2 发出的包

以承载第 100 个回显请求与应答报文的 MAC 帧，即图 3-7、3-8 中被选中的数据包，为例，分析数据帧的内容如下：

a) 整个数据帧的长度为 104 字节，其中 MAC 头 14 字节，VNI 头 6 字节，IP 头 20 字节、ICMP 头 8 字节，ICMP 载荷 56 字节（如图 3-6 所示）

b) 被蓝色下划线标识的部分为 MAC 头部，根据 MAC 帧格式有，MAC 帧的目的

MAC 地址为广播地址；源 MAC 地址分别为 vm1 与 vm2 的 MAC 地址，如图 3-3、3-4 所示；MAC 帧长度类型域为 0xf4f0，表明 MAC 帧承载了 VNI 帧。

c) 被紫色下划线标识的部分为 VNI 头部，其中前四个字节为 0330，与图 1-1 一致；后两个字节为序号，其中 vm1 发送的第 100 请求报文的序号为 0x0067，转化成十进制数为 103，而非 99，原因是 vm1 在发送 ping 报文的同时可能还有其他报文产生，例如第 83 号 VNI 帧中，IP 数据包载荷的类型为 UDP 协议，如图 3-9 所示；而 vm2 发送的第 100 个应答报文的序号恰为 0x0063，即 99。

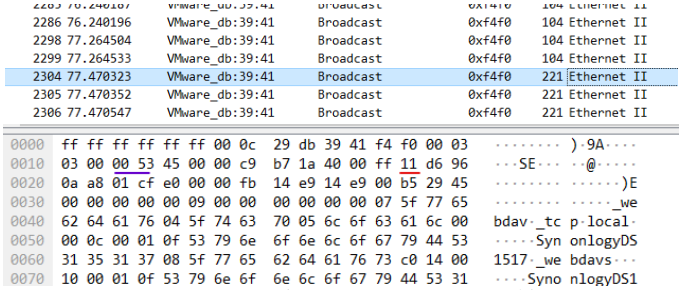


图 3-9 vm1 发出的第 83 号 VNI 帧

d) 被红色下划线标识的部分为 IP 头部，根据 IP 头部格式有，IP 承载的上层协议为 ICMP 协议，vm1 发出的包的源 IP 为 0aa801cf 即其自身的 IP 地址，目的 IP 为 0aa801cfba 即 vm2 的 IP 地址，而 vm2 发出的包则相反。

e) 被绿色下划线标识的部分为 ICMP 头部，其中 vm1 发出的报文的类型与代码域为 08_00，即回显请求（ping 请求）；vm2 发出的报文的类型与代码域为 00_00，即回显应答报文；两者的标识符皆为 0x0c10，序列号均为 0x0064，即 100，说明两者 vm2 发出的报文是对 vm1 发出的请求报文的应答。

4. 发送完 100 个 ping 报文后，在 vm1 上使用命令 dmesg -T 命令查看内核的输出信息，可以看到 VNI 统计子模块打印的信息，如下图:

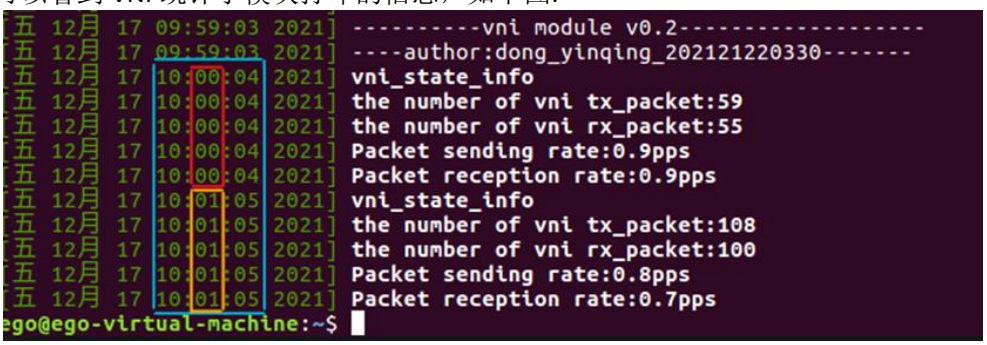


图 3-10 ping100 个报文后内核的输出信息

a) 其中，第一次打印时，VNI 模块发送了 59 个数据帧，收到 55 个数据帧，发送速率为 0.9pps，接收速率为 0.9pps。第二次打印时，VNI 模块发送了 108 个数据帧，收到 100 个数据帧，发送速率为 0.8pps，接收速率为 0.7pps。两次打印的时间如图 3-10 中蓝色方框中的信息所示。其中红色方框与黄色方框的内容显示：两次打印的时间间隔为一分钟。这说明 VNI 统计子模块每隔一分钟便会执行一次打印操作。

b) 一般情况下，Linux 系统中 ping 报文的发送与接收速率为 1pps，即每秒一个。而图 3-10 统计的结果略低于 1pps。这是由于两次统计速率的时间间隔过大，如果降低时间间隔则可以实现更为精确的统计。将间隔由 60s 改为 10s，获得 VNI 统计子模块打印的信息如下图：

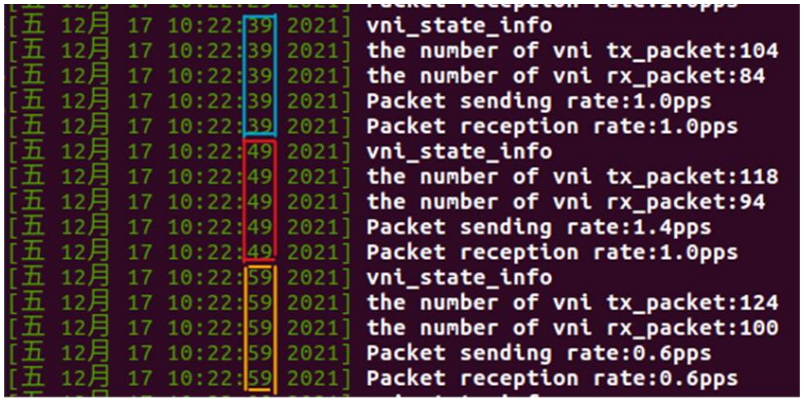


图 3-11 降低时间间隔后的统计结果

可见，蓝红黄三个方框中的内容说明 VNI 统计子模块打印信息的周期由 60s 变为了 10s，而 VNI 模块的接收速率为 1.0pps，与理论结果一致。

4 结束语

本文采用 LKM 的方式在 Linux 内核中添加了一个 VNI 模块。该模块介于内核 IP 模块与以太网络设备之间，兼具完整的发送与接收 VNI 类型数据帧的功能以及简单的统计功能。

参考文献:

[1] linux 网络栈结构 - AISEED - 博客园[EB/OL]. [2021-12-13]. https://www.cnblogs.com/yhp-smarthome/p/6926246.html.

[2] SHEH J. Linux 内核模块编程 HelloWorld[EB/OL]. J.e, 2018-03-07. (2018-03-07)[2021-12-13]. https://jerrysheh.com/post/75b0adbf.html.

[3] THYCHAN. Linux 内核模块 | Chan's Blog[EB/OL]. (2016-12-03)[2021-12-13]. http://yoursite.com.

[4] Linux 网络协议栈(二)——套接字缓存(socket buffer) - YY 哥 - 博客园[EB/OL]. [2021-12-13]. https://www.cnblogs.com/hustcat/archive/2009/09/19/1569859.html.

[5] How SKBs work[EB/OL]. [2021-12-13]. http://vger.kernel.org/~davem/skb_data.html.

[6] MXI1. SKB(struct sk_buff)数据结构的部分分析[J]. Minjun's Weblog, 2007.

[7] IP/TCP/UDP checsum - codestacklinuxer - 博客园[EB/OL]. [2021-12-15]. https://www.cnblogs.com/codestack/p/13633566.html.

[8] linux kernel --- checksum 相关 ip_summed 和 feature 字段解释_栀子花蛋糕的博客-CSDN 博客[EB/OL]. [2021-12-15]. https://blog.csdn.net/cherylchenyajun/article/details/109604983.

[9] netfilter/iptables project homepage - The netfilter.org project[EB/OL]. [2021-12-14]. https://www.netfilter.org/.

[10] networking - What is the difference between NF_DROP and NF_STOLEN in Netfilter hooks? - Stack Overflow[EB/OL]. [2021-12-14]. https://stackoverflow.com/questions/19342950/what-is-the-difference-between-nf-drop-and-nf-stolen-in-netfilter-hooks.

[11] Linux 内核分析 - 网络[三]：从 netif_receive_skb()说起 - 程序员宝宝[EB/OL]. [2021-12-14]. https://www.cxybb.com/article/qy532846454/6339789.

[12] dev_add_pack[EB/OL]. [2021-12-14]. https://www.kernel.org/doc/htmldocs/networking/API-dev-add-pack.html.

[13] dev_queue_xmit[EB/OL]. [2021-12-14]. http://kernelbook.sourceforge.net/kernel-api.html/r2822.html.

[14] Linux 内核构造数据包并发送(二)（dev_queue_xmit 方式）_stonesharp 的专栏-程序员宅基地 - 程序员宅基地[EB/OL]. [2021-12-15]. https://www.cxyzjd.com/article/stonesharp/8889333.

[15] 如何实现自定义 sk_buff 数据包并提交协议栈_遇见你是我最美丽的意外-程序员秘密 - 程序员秘密[EB/OL]. [2021-12-15]. https://www.cxymm.net/article/s2603898260/92019175.

[16] 使用 dev_add_pack 注册新的以太网类型_dean_gdp 的专栏-CSDN 博客[EB/OL]. [2021-12-16]. https://blog.csdn.net/dean_gdp/article/details/34091047.
[17] Linux 内核实践 - 如何添加网络协议[二]：实现_yoyo 的专栏-CSDN 博客_linux 网络协议实现[EB/OL]. [2021-12-16]. https://blog.csdn.net/qy532846454/article/details/6646122.
[18] Linux 网络协议栈(四)——链路层(1) - YY 哥 - 博客园[EB/OL]. [2021-12-15]. https://www.cnblogs.com/hustcat/archive/2009/09/26/1574371.html.
[19] Netfilter 之 钩子函数注册 - AlexAlex - 博客园[EB/OL]. [2021-12-15]. https://www.cnblogs.com/wanpengcoder/p/11755574.html.
[20] Netfilter 代码分析 - IBM@sdu[EB/OL]. [2021-12-15]. https://sites.google.com/site/ibmsdu/Home/linuxunix-programing/netfilter-%E4%BB%A3%E7%A0%81%E5%88%86%E6%9E%90.
[21] linux 内核定时器_hhhhhyyyyy8 的博客-CSDN 博客_timer_setup[EB/OL]. [2021-12-15]. https://blog.csdn.net/hhhhhyyyyy8/article/details/102885037.
[22] Linux 内核定时器 二 例子 demo_chyQino 的博客-CSDN 博客[EB/OL]. [2021-12-16]. https://blog.csdn.net/qq_38907791/article/details/90083389.