隧道模式下 Linux 路由器内存优化的研究

熊 丹 钱华林

（中国科学院计算机网络信息中心 北京 100080）

摘 要 在网络通信中，为了满足应用的需要，常常在数据包中添加特定的协议头部。例如，为了实现从 IPv4 数据包到 IPv6 数据包的转换，可以在原 IPv4 报头前封装上相应的 IPv6 报头；在以太网中，为了保证应用的 QoS 需要，可以在数据链路层报头之后封装上 MPLS 标记。这种在数据报头的固定位置封装固定长度协议头的方式，称为隧道模式。传统的隧道封装方式需要重新在内存中申请 skb 空间，需要重新拷贝整个数据包，这样的重复内存拷贝降低了系统的性能，并不是必要的。本文在内核代码的层次，分析了数据包的网络处理流程，并修改了 Linux 的系统调用，通过预留内存资源的方式，提出了一种避免隧道模式下额外内存拷贝的方法，并给出了代码实现。

关键词 隧道模式，Linux 路由器，预留内存，sk_buff

Study on Linux Router Memory Optimization in Tunnel Model

XIONG Dan QIAN Hua-Lin

(Computer Network Information Center of Chinese Academy of Sciences, Beijing 100080)

Abstract In network communication, we often encapsulate packets with special protocol header. For example, we can use IPv4-over-IPv6 tunnels to carry IPv4 traffic within an IPv6-only network. We can also add MPLS Label after the Data link Layer header for QoS guarantees. The traditional tunnel model always causes the additional memory copying and reduces the system performance. In this paper, we analyze the process of packet handling, and rewrite the Linux System Call. This paper presents a method which can avoid the additional memory copying by reserving memory in advance.

Keywords Tunnel model, Linux router, Reserving memory in advance, sk_buff

1 引言

Internet 获得广泛应用的一个重要原因是网络协议的层次化，例如 OSI 的七层模型、TCP/IP 的五层模型等。这种网络协议的层次化，表现在数据包的结构上，就是较底层的网络协议头部封装了较高层的协议头部；某一层次的网络处理程序，只处理相对应的数据包头部字段。为了满足应用的需要，我们常常在数据包中添加特定的协议头部[1]。例如，为了实现从 IPv4 数据包到 IPv6 数据包的转换，我们可以在原 IPv4 报头以前，封装上 40 字节的 IPv6 报头；在以太网中，为了保证应用的 QoS 需要，我们可以在数据链路层报头之后封装上 4 字节的 MPLS 标记（当然，在不同的网络中，MPLS 标签的位置并不固定，并非总是在数据链路层报头以后，例如光网络）。这种在数据报头的固定位置封装固定长度协议头的方式，本文称为隧道模式。我们可以使用专门的嵌入式路由器来实现上述的这些特定的隧道模式的应用。在嵌入式 Linux 路由器中，可以根据应用灵活定制系统组件，用比较少的计算资源和存储资源，比较低的成本，实现比较高的系统性能[2]。国际上已经有了很多项目，在内核模块的层次，优化了 Linux 系统，使之适用于嵌入式路由器，例如 Linux Router Project (LRP)，Linux Embedded Appliance Firewall (LEAF) 等。本文在内核代码的层次，分析了数据包的网络处理流程，并修改了 Linux 的系统调用，通过预留内存资源的方式，提出了一种避免隧道模式下额外内存拷贝的方法。

本文是这样组织的，第 2 节简单介绍隧道模式下网络内存处理的具体流程；第 3 节提出一种避免隧道模式下额外内存拷贝的方法，并给出其代码实现；最后是文章的小结。

2 网络内存处理的具体流程

在接收数据包时，从物理设备上获得低层的数字信号，转化成一个个的数据包，存放在内存中。当新的数据包到达网卡时，会触发一个中断函数 ei_interrupt()（Linux-2.6.0/drivers/net/3c503.c）。ei_interrupt() 函数只进行必要的中断操作，并没有对新的包进行太多的处理，就交给了接收处理函数 ei_receive()。ei_receive() 首先检查接收的包是否正确，如果是一个“好”包，就会为包分配一个缓冲结构（dev_alloc_skb()）。dev_alloc_skb() 函数调用 alloc_skb() 函数，首先在优先级较高的缓存中申请一个 sk_buff{} 结构（include/linux/skbuff.h），然后根据数据包的大小，在内存中申请一块连续的内存空间存放数据包。sk_buff{} 结构是一个指针的集合，不同的指针指向数据包的不同字段，方便不同层次的网络协议处理，其中的 head 指针指向内存中存放数据包的数据区。这样，驱动程序完成了对新数据包的接收过程。接下来，调用上层的函数 netif_rx()（net/core/dev.c），把包交给上层网络处理函数，例如隧道封装函数[3]。

以 IPv4 数据包到 IPv6 数据包的转换为例，我们需要在

熊 丹 硕士，主要研究方向是计算机网络体系结构、网络通信协议；钱华林 总工程师、国际互联网络名字与编号分配机构（ICANN）执行理事长，主要研究方向是网络体系结构、网络通信协议。

原 IPv4 报头以前，封装上 40 字节的 IPv6 报头[2]。进行隧道封装时，由于内存中数据包所占的内存空间已经固定，无法用 skb_push()函数等简单的指针操作得到额外的 40 字节空间，所以只好再申请一块足够容纳添加隧道封装后的数据包的内存空间作为 sk_buff{}的数据区，并将原数据区的内容拷贝到新数据区。不妨假设封装前的数据包(包括以太帧头)总长度为 1000 字节。驱动程序接收数据包时，使用系统调用 alloc_skb()分配了 1000 字节的连续地址空间。为了封装上 40 字节的隧道头，新申请了一块 1040 字节的空间，除了封装隧道头以外，还拷贝了 1000 字节的原数据区，最后还要把原数据区释放掉。在繁忙的网络上，这种大量拷贝内存的方法，会明显地影响路由器的性能。

3 避免隧道模式下额外内存拷贝的方法

其实，广义上说，对以太网上的路由器，对任何一个数据包都要进行隧道封装，因为我们总是要在 IP 报头前封装上 14 字节的以太帧头[3]。不过，我们都约定在 dev_alloc_skb()函数中，使用 alloc_skb(length + 16)和 skb_reserve(skb, 16)(include/linux/skbuff. h)，在申请内存空间时，预留出了 16 个字节的空间(以太帧头为 14 字节。为了使以太帧头后面的 IP 头和 long 型的地址对齐，约定预留 16 字节空间)。这样，封装以太帧头时，直接在原数据区的对应位置封装即可，并不需要对数据包进行再次拷贝。

由上面的方法得到启发，我们可以修改 Linux 的系统调用 alloc_skb()如下[4]：

```
struct sk_buff * alloc_skb(unsigned int size,int gfp_mask)
{
  struct sk_buff * skb;
  u8 * data;
  ...
  data = kmalloc( size + 40 + sizeof( struct skb_shared_info ),
         gfp_mask); //修改
         skb_reserve(skb,40); //增加
         if (! data)
            goto nodata;
         ...
}
//(include/linux/skbuff. h)
//以上程序在 Linux2. 4. 18,Linux2. 6. 0 上编译通过
```

分配内存空间时，使用 kmalloc(size+40)，多分配 40 字节的内存空间，然后使用 skb_reserve(skb,40)移动 skb→data 指针，在数据区的头部预留出 40 字节空间。alloc_skb()是 Linux 系统调用，任何网卡的驱动程序都使用它来分配内存空间。当重新编译 Linux 内核后，进行 IPv6 头的封装时，由于数据区 IPv4 地址前有 56 个空闲的地址空间，我们就可以使用 skb_push()，skb_pull()等函数直接操作 skb→data 指针，依次封装上 40 字节的 IPv6 头和 14 字节的以太帧头，而不需要像第 2 节中叙述的重新申请数据区并对数据包进行 2 次拷贝了。

sk_buff{}结构的数据区如图 1 所示。

我们使用 alloc_skb()函数申请的内存空间是 skb→head 到 skb→end 的部分，预留的空间是 skb→head 到 skb→data 的部分。接收数据包时，有效数据存储在 skb→data 到 skb→tail 的部分。封装时，使用 skb_push()将 skb→data 指针向前移动 40 字节，并将新的 IPv6 头填充到这 40 字节空间中。发送数据包时，发送的是 skb→data 到 skb→tail 之间的有效数据部分。

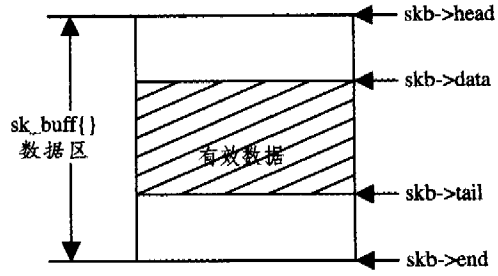

图 1 sk_buff{}数据区结构图

可以注意到我们修改 alloc_skb()函数时，规定了必须预留 40 字节空间。也就是说，这样的 Linux 路由器是专门适用于做 IPv4 数据包到 IPv6 数据包的转换的。如果它接收到一个不需要做 IPv4 数据包到 IPv6 数据包的转换的普通数据包(IPv4 包或 IPv6 包)，它同样可以正常地处理，唯一的不同是它额外占用了 skb→head 到 skb→data 之间的 40 字节的内存空间，因为不需要封装，这部分预留的内存一直没有使用。例如，接收到一个 1000 字节长度(包括以太帧头)的数据包，我们默认分配了 1040 字节的内存空间，并在数据区头部预留了 40 字节，然后将网卡上的数字信号，也就是有效数据包内容拷贝进剩下的 1000 字节内存里，完成接收过程。如果这个数据包需要封装，则封装后有效数据区的长度增加到 1040 字节，发送出的数据包长度增加到 1040 字节；如果这个数据包不需要封装，则有效数据区的长度仍为 1000 字节，发送出的数据包长度仍为 1000 字节。

同时，由于我们预留的内存空间是 40 字节，所以我们对 Linux 内核的改动同样适用于那些隧道头长度小于 40 字节的应用，例如封装 4 字节的 MPLS 标签。这种情况下，它额外占用了 36 字节的内存空间。

对于封装的隧道头部大于 40 字节的应用，只作 40 字节的预留就不够了。可以仿照上面的例子做专门的改动，增大在数据区头部预留的内存值。

小结 嵌入式 Linux 路由器，成本比较低，常使用它来做一些专门的网络应用。本文讨论了 Linux 路由器进行隧道封装时的内存处理流程，修改了 Linux 系统调用，提出了以预留内存的方式避免额外内存拷贝的方法。本文利用了 Linux 内核源代码开放易于定制的特点，根据应用需要灵活修改系统调用，并给出了代码实现，用比较少的计算资源和存储资源，实现比较高的系统性能。

参 考 文 献

1 Perkins C. [RFC2003] IP Encapsulation within IP, October 1996
2 李善平，刘文峰，王焕龙，等. Linux 与嵌入式系统. 2003. 110～132
3 李善平，刘文峰，李程远，等. Linux 内核 2. 4 版源代码分析大全. 2002. 433～580
4 Linus Torvalds ,and etc,Linux source code 2. 6. 0,2003