第 28 卷 第 1 期
2005 年 1 月

计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

Vol. 28 No. 1
Jan. 2005

零拷贝报文捕获平台的研究与实现

王佰玲 方滨兴 云晓春

(哈尔滨工业大学计算机网络与信息安全技术研究中心 哈尔滨 150001)

摘 要 根据零拷贝思想,实现了一种高性能的报文捕获平台(High-Performance Packet Capture Platform,HPPCP).通过实现网络接口设备直接将数据报文以 DMA 方式存储到应用程序可以访问的地址空间,避免数据报文在内核态里传输时的内存操作,缩短了数据报文的行走路径;通过环策略管理数据报文缓冲区,实现了网卡和应用程序能够无冲突访问共享资源.这两点有效地降低了网络通信的延迟,极大地节省了 CPU 的开销.通过性能的分析比较表明,接收 64Byte 与 1500Byte 的报文时吞吐量分别达到 90 万 pps (439Mbps) 和 8.2 万 pps (938Mbps),与传统的报文传输机制相比,报文捕获能力有了较为显著的提高.

关键词 零拷贝;高速网络接口;内存映射;地址翻译;无冲突访问

中图法分类号 TP393

The Study and Implementation of Zero-Copy Packet Capture Platform

WANG Bai-Ling FANG Bin-Xing YUN Xiao-Chun

(Research Center of Computer Network and Information Security Technology, Harbin Institute of Technology, Harbin 150001)

Abstract Based on the Zero-Copy network I/O, a high-performance packet capture platform (HPPCP) is proposed in this paper. By realizing DMA the packet from network card to the memory that the user program can access directly, HPPCP avoids the memory access in kernel state and shortens the path to transmit a packet; by managing the data buffer using the ring theory, HPPCP can work without collision to access the shared resource. The latency in communication is efficiently reduced for these two improvements. Experimental results indicate that the throughputs of HPPCP for 64Byte and 1500Byte messages are 900,000pps (about 439Mbps) and 82,000pps (about 939Mbps) respectively, and HPPCP surpasses the traditional ones' in performance.

Keywords zero copy; high-speed network interface; memory map; address translation; access without collision

1 引 言

高性能的报文捕获平台在面向大规模宽带网络的入侵检测系统[1]、大流量网络数据情况下的网络协议分析、宽带网络防火墙、高性能通信系统[2]、高性能路由器、主机路由器等领域中,都有广泛的应用前景.然而传统的报文捕获平台往往成为整个系统的性能瓶颈.其性能影响因素主要有两方面:(1)在传统报文处理过程中,在核心态进行了多次的内存操作,比如数据报头校验、控制顺序等.(2)数据报文从内核态向用户态传送的时候,是用单个报文驱动

收稿日期:2002-10-29;修改稿收到日期:2004-11-19.本课题得到国家"八六三"高技术研究发展计划项目"计算机病毒防范技术"(2001AA147010B)资助.王佰玲,男,1978 年生,博士研究生,研究方向为计算机网络安全、信息内容安全.E-mail:wbl@hit.edu.cn.方滨兴,男,1960 年生,博士,教授,博士生导师,主要研究方向为计算机网络安全、信息内容安全、并行计算.云晓春,男,1971 年生,博士,教授,博士生导师,主要研究方向为计算机网络安全、信息内容安全和网格计算.

机制,也就是说应用层每次系统调用只从内核读出一个数据报文,这样就造成了系统性能的不必要浪费.

为了提高报文捕获平台的性能,我们曾经提出一个方案,在这个方案中我们在内核注册了一个模块,避开了系统协议栈对数据报文的操作,增大了内核态数据报文的缓冲能力,批量地从内核态向用户态传输数据,提高了从内核态向用户态报文传送的速率.但是,我们发现改进后的报文捕获平台在宽带网特大流量下或对小数据报文处理能力要求特别高的情况下,报文捕获平台仍可能成为系统的性能瓶颈.所以我们认为要构建一个高性能的报文捕获平台,必须要有一个高性能的用户级的网络接口,以便能使用户的应用程序避开操作系统的干预,直接与网络接口进行交互,减少甚至消除内存拷贝的发生,由此我们设计了零拷贝报文捕获平台方案.

2　零拷贝平台结构设计

所谓零拷贝(zero copy)[3]是指在某节点的报文收发过程中不会出现任何内存间的拷贝,发送时数据包由应用程序的用户缓冲区直接经过网络接口到达外部网络;同理,接收时网络接口直接将数据包送入用户缓冲区.在结构上我们做了比较大的改动,为了方便,我们将传统报文捕获平台和零拷贝报文捕获平台结构上作了比较,如图 1 所示(在这里我们仅显示了接收报文过程).和正常的协议栈相比,我们注册了一个自己的模块 Dum,有关这个 Dum 的详细介绍,见第 3 节 Dum 的具体实现及关键技术.

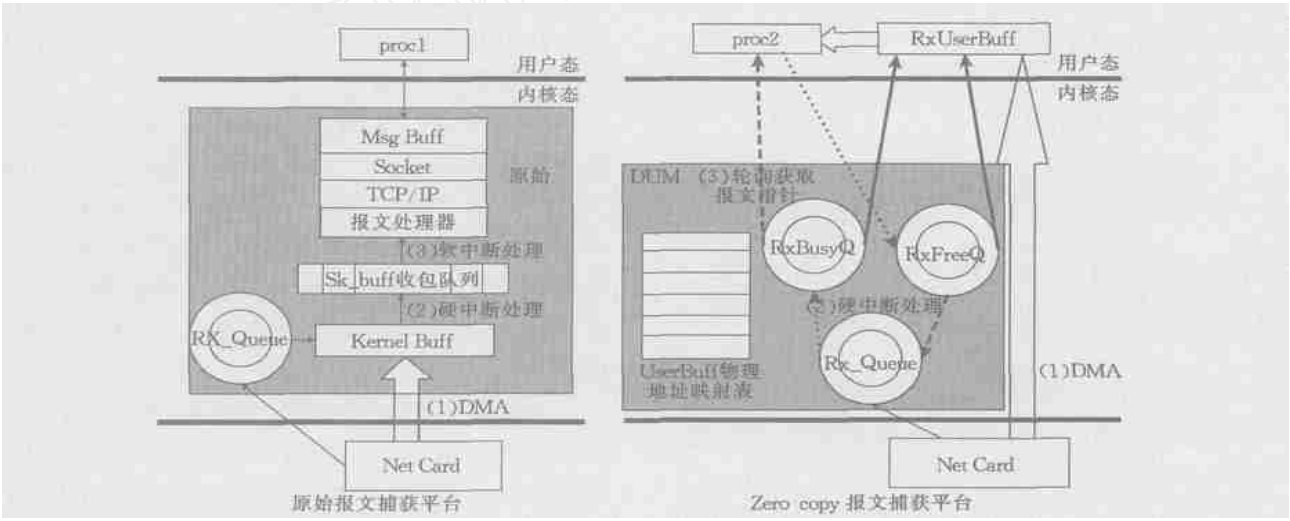

图 1　原始报文捕获平台与零拷贝报文捕获平台结构

很明显,在原始报文捕获平台结构上,当应用层想从网卡获得数据报文时需要经过两个缓冲区和正常 TCP/IP 协议栈,其中由软中断负责从第一个缓冲区(sk-buff)读取数据报文,驱动报文处理器,再拷贝到 MSG Buff 中,最后由应用层通过系统调用将数据报文读到用户态.而零拷贝平台则是放弃了操作系统的协议栈,在网卡进行 DMA 操作时直接将数据传送到用户态[4],由硬中断处理程序改变数据环中指针;而用户态捕包应用程序则可以采用轮寻策略或由硬中断处理函数唤醒的机制来获取数据报文指针.在数据报文从网卡到应用层消息传输的过程中,每个环保存的都是新数据报文在 UserBuff 中的索引,这样就避免了不必要的内存的拷贝.

3　实现及关键技术

Dum 设备是一个虚拟设备,由用户在内核创建,以模块的形式加载.Linux 有一个很好的特性就是它可以在运行的时候扩展内核代码,也就是说可以在系统运行的时候增加系统功能,每个可以增加到内核中的代码称为一个模块[5].在 Linux 中,设备大都是以模块形式加载的,我们创建的 Dum 设备也不例外.Dum 是用户和网卡之间的访问中介,是整个系统的核心所在.Dum 必须完成 3 件事情:

(1) 让网卡能够直接访问到用户态的 UserBuff.

(2) 让用户能够直接访问到内核态的BusyQ,

FreeQ.

(3) 提供接口,让网卡和应用程序能够无冲突地通过 BusyQ,FreeQ 访问 UserBuff.

3.1 内存分配及数据结构

(1) UserBuff 的内存结构. UserBuff 是在用户态申请的,是存放数据报文的真正缓冲区,我们将 UserBuff 分为接收数据报文缓冲区 RxUserBuff 和发送报文数据缓冲区 TxUserBuff,并被分成每 2K 一个的数据块. RxUserBuff 块数和 RxBusyQ 及 RxFreeQ 项数的和相同. TxUserBuff 块数和TxBusyQ 及TxFreeQ 项数的和相同. 我们之所以选择每块 2K 是因为在 Linux 系统中,页大小是 4K,存放数据报文的数据块又要大于一个 MTU(1514 字节);这样每页只放 2 个数据报文,就保证了一个数据报文不会跨越两个页,而因此网卡进行 DMA 操作时每次只和一个内存页交互即可. UserBuff 每个块结构体定义如下:

```
struct su-buff{
    char packet[1514];
    uint pktlen;
    uint koffset;
}
```

其中,*packet* 用来保存数据报文,*pktlen* 为数据报文的实际长度,*koffset* 为本数据块在 UserBuff 中的索引.

(2) Dum 的内存分配. 我们在内核中,将 BusyQ,FreeQ 以及 UserBuff 的物理地址映射表放在一起,用 vmalloc 同时申请出. 其中 BusyQ 分为 RxBusyQ,TxBusyQ;FreeQ 分为 RxFreeQ,TxFreeQ,结构安排如图 2 所示.

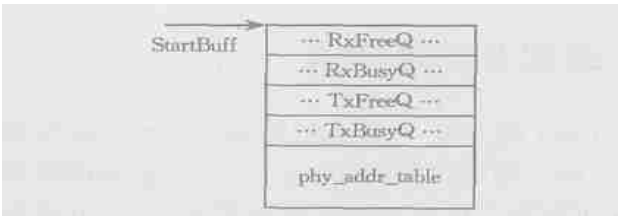

图 2 Dum 的内存分配

图 2 中 RxFreeQ,RxBusyQ,TxFreeQ,TxBusyQ 均为 SWAP-QUEUE 类型对象,SWAP-QUEUE 结构体定义如下:

```
typedef struct{
    volatile unsigned long readptr;
    volatile unsigned long writeptr;
    uint queue[PKT-NUM];
}SWAP-QUEUE
```

其中,*readptr* 为环读指针,*writeptr* 为环写指针,*queue[i]*为环项指针,若 *queue[i]=j*,则表示环项 *queue[i]*指向 UserBuff*[j]*. 切记:这里所谓的环项指针 *queue[i]*保存的值是 UserBuff 数组(RxUserBuff 或 TxUserBuff)下标.

3.2 网卡 DMA 直接访问到用户态的 UserBuff 的方法

为了解决内存空间不足的问题,操作系统引入了虚拟内存的概念[6]. 操作系统为每个用户进程分配一段虚拟内存,用户应用程序便使用内存的虚拟地址,操作系统然后再为每个用户进程维护一个多级页表(Linux 维护的是一个三级页表),以实现虚拟地址到物理地址的转换. 而网络接口直接使用内核的物理地址进行寻址. 要实现让网络接口将数据直接存储到用户空间的 RxUserBuff,或直接从 TxUserBuff 取数据报文,必须解决用户空间到内核空间的地址翻译问题[4,7].

在这里我们专门为 UserBuff 定义了一个用户物理地址映射表 phy-addr-table. phy-addr-table 其实就是一个 32 位无符号整形数组. 一个 phy-addr-table 元素保存着 UserBuff 中的一个物理内存页首地址,而这个物理内存页保存着两个数据报文块,所以 phy-addr-table 元素个数是 UserBuff 元素个数的一半.

在初始化的时候由用户将 UserBuff 进行页对齐得到对齐后的地址 UBVA,再通过系统调用将对齐后的首地址 UBVA 传到内核态. 对 phy-addr-table 进行初始化,使得 phy-addr-table*[i]*保存着从首地址 UBVA 开始的第 *i* 个页对应的物理地址;同时将这个物理页锁定,不允许操作系统在程序运行的过程中将这些物理页换出. 得到某个虚拟地址对应的物理地址并将其锁定方法如下(这里略去了一些边界讨论、判断),其中,*page-num* 为事先算好的页数,*vaddr* 初始值为 UBVA.

```
for (i=0; i<page-num; i++, vaddr += PAGE-SIZE){
    pgd-t *pgd = pgd-offset(current->mm,
                 (unsigned long) UBVA);
    pmd-t *pmd = pmd-offset(pgd,
                 (unsigned long) vaddr);
    pte-t *pte = pte-offset(pmd,
                 (unsigned long) vaddr);/
    phy-addr-table[i] =
```

```
        (pte-val(*pte) & PAGE-MASK);
    /*lock the user buffer page */
    page = pte-page(*pte);
    set-bit(PG-locked, &page->flags);
    atomic-inc(&page->count);
}
```

3.3 用户直接访问到内核态的 Busy Q, Free Q 的方法

在我们的设计方案里,系统收到数据报文后,要把数据从核心态直接存储到用户态.之后就是修改和用户态共享处于内核态的环 BusyQ,FreeQ 来达到同步的.但是在 Linux 操作系统中用户态程序不能够直接访问内核空间的数据.那么我们如何才能让用户态的应用程序访问到 BusyQ,FreeQ 呢?

幸好 Linux 操作系统提供了一种技术,这种技术可以将映象文件和数据文件直接映射到进程的地址空间.在这种映射中,文件的内容被直接连接到进程虚拟地址空间上,这就是内存映射[8,9]技术.我们就用这种方式实现了将虚拟设备 Dum 中的地址空间 BusyQ,FreeQ 映射到用户空间的.同时我们知道,在应用程序运行最开始的时候,操作系统并没有为其分配足够的页面,然而一个程序照样可以正常运行完.这是因为随着进程不断访问那些不在内存中的页面,系统将产生页面失效,内核就会为其分配空闲页框,将失效页面调入内存,这种处理方式叫做页面失效处理.

为了实现这个功能,我们在 Dum 模块的系统调用中定义了 dum-mmap 函数,当应用层对 Dum 虚拟设备进行 mmap 时,会调用这个函数.这个函数新建立一个 *vma*,同时将 *vma->vma-start* 返回给应用层作为用户可以使用的虚拟地址.这个函数中并没有做什么实质性的工作,代码如下(边界讨论、判断略去):

```
int dum-mmap(struct file * filp, struct vm-area-struct * vma)
{
  vma->vm-ops = &dum-vm-ops;
  vma->vm-flags | = VM-RESERVED;
  dum-vma-open(vma);
  return 0;
}
```

其中 *dum-vm-ops* 定义为:

```
struct vm-operations-struct dum-vm-ops = {
        open:    dum-vma-open,
        close:   dum-vma-close,
        nopage:  dum-vma-nopage,
};
```

正如以上代码所示,当用户 mmap 虚拟设备 Dum 的时候,只是得到了一个虚拟地址;而当用户程序访问这个虚拟地址空间中的数据,这个数据所在的页从未调入内存或调入后已被换出时,操作系统就会进行缺页处理. *dum-vma-nopage* 做了真正实质性的工作,它在用户层访问这个 *vma*,发生页面失效时将用户和内核共享的 StartBuff 相应内存地址返给用户层.这个函数首先将用户要访问的虚拟地址(由 *dum-vma-nopage* 的参数传入)和 *vma->vma-start* 做差得出偏移量 *Offset*,然后将 *StartBuff + Offset* 所对应的物理页调入内存,并返回这个物理页.比如:当应用层 mmap 虚拟设备 Dum 后,访问 mmap 返回的地址所指向的第一个字节时,*dum-vma-nopage* 会得到 *Offset* 为 0.这时就会将 StartBuff 首地址所在的页调入内存,并返回给应用层.获得物理页地址的方法和 3.2 节相似,这里不再赘述.

3.4 网卡和应用程序无冲突访问控制 Busy Q, Free Q 的方法

由于应用层读数据报文时要和网卡 DMA 传输时访问同一块缓存 RxUserBuff,发送报文时要和网卡 DMA 访问同一块缓存 TxUserBuff.所以我们有必要采取一定措施,使得在读写内存同步的时候不影响系统性能.这里我们采用对环操作.本文仅以接收报文相关的结构为例来说明无冲突访问的实现.环结构图如图 3 所示.

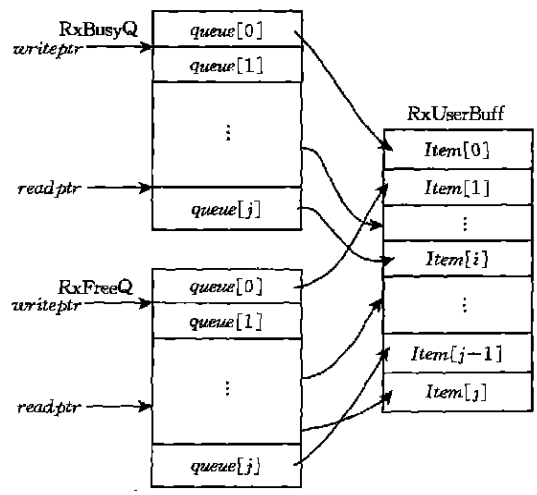

图 3　接收数据报文环结构

在这里,我们定义环 RxBusyQ 为接收报文数据

环,RxFreeQ 为接收报文空闲环,两个环中元素的个数均为 *PKT-NUM*;定义 RxUserBuff 为 *PKT-NUM* 个元素的数组,每个元素为 2K 字节空间. 设定访问规则如下.

规则 1. 当环的 *readptr* 和 *writeptr* 相等时表示为空;

规则 2. 当环的 *writeptr* + 1 等于 *readptr* 时表示环已经满了①.

为了描述方便,设定对环操作的方法.

方法 1. 对环读操作:

```
# define GET-READER-ITEM(Q) Q →
queue[Q →readptr];
```

方法 2. 对环写操作:

```
# define SET-WRITER-ITEM(Q,i) \
  Q →queue[Q →writeptr] = i;
```

方法 3. 向前移动环的写指针:

```
# define INC-WRITER(Q) Q →writeptr + + ; \
if(Q →writeptr > = PKT-NUM) Q →writeptr = 0;
```

方法 4. 向前移动环的读指针:

```
# define INC-READER(Q) Q →readptr + + ; \
if(Q →readptr > = PKT-NUM) Q →readptr = 0;
```

方法 5. 判断环是否空:

```
# define IS-EMPTY(Q) (Q →readptr = = Q →writeptr).
```

在以上方法和规则的基础之上,我们可以对数据环(RxBusyQ)和空闲环(RxFreeQ)进行读、写操作了,下面我们一一介绍:

1. 初始化(QueueInit).

将 RxBusyQ 初始化为 *writeptr* = 0, *readptr* = 0,即数据环为空;RxFreeQ 初始化为 *writeptr* = 0, *readptr* = 1,即空闲环为满(全是空闲的);同时将 RxFreeQ 各个 *queue* 项初始化:*queue*[*i*] = *i*,即 *queue*[*i*]指向 RxUserBuff[*i*].

2. 取一个空闲项地址(GetFreeItem).

```
If (IS-EMPTY(RxFreeQ))
  return NULL;    //RxFreeQ 没有空闲项
else {
      i = GET-READER-ITEM(RxFreeQ);
      INC-READER(RxFreeQ); //向前移动指针
      Return &RxUserBuff[i];  //返回地址
}
```

3. 添加一个数据项(AddDataItem).

由于 RxBusyQ 与 RxUserBuff 项的个数都是一样的,所以这个操作不存在满的情况(只要能从环中取出来,就可以放到环中去). 所以将 RxUserBuff[*i*]添加到 RxBusyQ 中表达为

```
SET-WRITER-ITEM(RxBusyQ,i); //赋值
INC-WRITER(RxBusyQ); //移动写指针
```

4. 取一个数据项(GetDataItem).

```
If IS-EMPTY(RxBusyQ)
  return NULL; //RxBusyQ 没有数据项
else
{
      i = GET-READER-ITEM(RxFreeQ);
      INC-READER(RxBusyQ); //向前移动指针
      Return &RxUserBuff[i];  //返回地址
}
```

5. 添加一个空闲项(释放一个数据项 AddFree-Item).

同步 3,由于 RxFreeQ 与 RxUserBuff 项的个数都是一样的,所以这个操作不存在满的情况所以将 *RxUserBuff*[*i*]添加到 RxFreeQ 中表达为

```
SET-WRITER-ITEM(RxFreeQ,i); //赋值
INC-WRITER(RxFreeQ); //移动写指针
```

在有新数据报文到达时,网卡直接以 DMA 方式直接存储到 RxUserBuff 中,然后发出硬中断;在硬中断处理程序里,首先要做的就是获取空闲数据块索引(GetFreeItem),再从 UserBuff 物理地址映射表中检索到相应的物理地址,初始化网卡 Rx-Queue,供下一次 DMA 操作使用;同时将这次到达的数据报文索引添加到 BusyQ 中(AddDataItem). 而应用层是不断地轮寻 RxBusyQ,从中获得数据报文索引(GetDataItem). 当处理完这个索引指向的数据报文后,将其释放,即添加到 RxFreeQ 中去(AddFreeItem). 通过这种访问机制,完全实现了无冲突访问控制.

4 零拷贝平台的性能测试

为了对改进报文捕获平台的性能做进一步的分析,我们对传统数据报文捕获平台和零拷贝报文捕获平台的实例进行了性能对比测试. 测试时由 SmartBits 控制发包速率和数据包长,捕包机使用高性能服务器,软件配置为 Linux 7.3 操作系统,内核为 2.4.18-5smp;硬件配置为 CPU:P Ⅳ2.0G Hz ×2,内存:4G,高速缓存:512KB/CPU,捕包网卡:Intel(R) PRO/1000 Network Driver.

4.1 报文捕获速率

在以上测试环境下,我们测试出了这两种报文捕获平台在以不丢包为前提的情况下,报文捕获最

① 说明:根据规则 1,2,对于一个有 *n* 项的环,当环为满的时候,只能够存储 *n* - 1 个数据. 这是由于为了避免访问冲突而设定的临界值引起的. 当环满的的时候 *writeptr* + 1 等于 *readptr*,共 *n* - 1 项;假如继续添加数据,必须向前移动指针,则 *writeptr* 就等于 *readptr* 了,而根据规则 1,此时为空环,这不是我们想要的. 所以此处为了避免访问冲突浪费了一个项的空间 2K.

大速率如表 1 所示.

表 1　两种报文捕获平台性能测试结果

IP 包长	libpcap 的最大速率		Zero copy 的最大速率	
	(×10⁴pps)	(Mbps)	(×10⁴pps)	(Mbps)
64B	17.0	83	90.0	439
512B	16.0	625	23.0	911
1500B	1.5	172	8.2	938

可以看出,零拷贝报文捕获平台无论对小报文还是大报文,其处理能力都得到了极大的提高.其实当数据报大小在 128 字节的时候,零拷贝平台就已经达到了线速.这是传统的报文捕获平台远远不及的.

4.2　小数据报文处理能力

一个面向宽带网络的 IDS 不仅要在其协议栈功能上发现并抵制 Dos 攻击,同时在性能上也要做到能够处理 Dos 攻击的数据报文;在并行计算中,常常用小消息报文控制信息的传递,它的延迟性能直接影响系统的计算速度.可见小报文处理能力也是报文捕获平台性能评价的另一标准.所以在这里我们同时给出了 IP 包长 64B 时的各种流量下的报文捕获平台捕包率对比折线图,如图 4 所示.

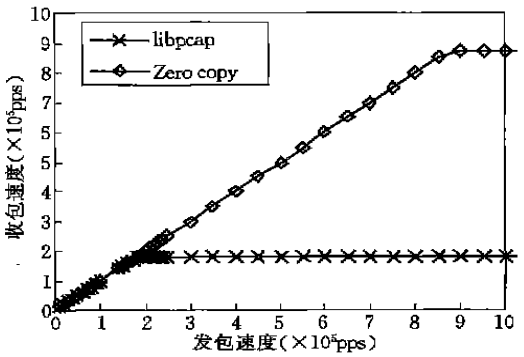

图 4　64 字节数据报文处理性能折线图

5　结束语

本文在对传统报文捕获平台性能影响因素分析的基础上,给出了一种高性能报文捕获平台——HPPCP.零拷贝思想的最大优点就是通过用户层和网络接口的直接交互,避免内存拷贝,可以缩短数据包的行走路径,极大地节省 CPU 的开销,从而为上层更复杂的处理赢得了宝贵的时间.实验数据表明收、发报文速率得到了极大的提高,甚至当数据报文长度在 128B 时,就已经达到了线速,这是其它报文捕获平台所达不到的.改进后的报文捕获平台在实际网络应用环境中吞吐率已经达到线速.

然而这种方案的应用影响了系统正常的协议栈,处于零拷贝状态的网卡,其收发数据必须通过调用平台提供的专用函数接口.以后的工作主要是设计一套有效的低延迟的协议,开发高性能的协议栈,充分结合已有的零拷贝平台,形成一套高速网络接口通信系统.

参　考　文　献

1　White G.B., Pooch U.. Cooperating security managers: Distributed intrusion detection systems. Computers & Security, 1996, 15(5): 441～450

2　Zhou Gui-Lin, Zhang Ying, Du Yi, Ma Qun-Sheng, Li San-Li. HPNI: A new network interface for cluster. Journal of Computer Research and Development, 2000, 37(2): 201～206(in Chinese)
(周桂林,张　瀛,杜　毅,马群生,李三立. HPNI:一种新型的机群系统网络接口. 计算机研究与发展, 2000, 37(2): 201～206)

3　Basu A., Buch V., Vogels W., von Eicken Thorsten. U-Net: A user-level network interface for parallel and distributed computing. In: Proceedings of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, 1995

4　Welsh M., Basu A., von Eicken T.. Incorporating memory management into user-level network interfaces. Cornell Univer-rsity Ithaca, NY, USA: Technical Report TR97-1620, 1997

5　Rubini A., Corbet J.. Linux Device Drivers. (2nd Edition): O'R-eilly, 2001(in Chinese)
(RUBINI A.等著.魏永明,骆　刚,姜　君译. LINUX 设备驱动程序(第 2 版). 北京:中国电力出版社,2002)

6　Peter J.. Denning: Virtual memory. The Computer Science and Engineering Handbook, 1997, 1747～1760. http://cne.edu/pjd/PUBS/vm.pdf

7　Jacob B., Mudge T.. Software-managed address translation. In: Proceedings of the 3rd International Symposium on High Performance Computer Architecture, San Antonio, Texas, 1997, 156～167

8　Liu Wei, Zheng Wei-Min, Shen Jun, Ju Da-Peng. Design and implementation of memory map mechanism in underlying communication. Journal of Software, 1999, 10(1): 24～28(in Chinese with English abstract)
(刘　炜,郑纬民,申　俊,鞠大鹏. 底层通信协议中内存映射机制的设计与实现. 软件学报, 1999, 10(1): 24～28)

9　Welsh M., Basu A., von Eicken T.. ATM and fast ethernet network interfaces for user-level communication. In: Proceedings of the 3rd International Symposium on High-Performance Computer Architecture (HPCA), San Antonio, Texas, 1997, 332～342

WANG Bai-Ling, born in 1978, Ph. D. candidate. His primary research interests include computer and network security, and information content security.

FANG Bin-Xing, born in 1960, professor and Ph. D. supervisor. His primary research interests include computer and network security, information content security, and parallel computing.

YUN Xiao-Chun, born in 1971, professor and Ph. D. supervisor. His primary research interests include computer and network security, information content security, and grid computing.

Background

Emerging high-speed networks provide several hundred megabits per second. However, the maximum achievable throughput available to the end-user or application is quite limited. So the study on high-performance network I/O has become a hot topic in this scope. Its goal is to fully utilize the network bandwidth and to improve the performance at the application level.

HPPCP and parallel TCP/IP stack running over it are two cores in our whole system, and most of the researchers are high on these fields. This paper proposes the study and implementation of HPPCP based on the Zero-Copy network I/O. The objectives of this study are: (1) to understand how the I/O subsystem relates to network operations, (2) to evaluate and analyze the performance of such a subsystem, and (3) to propose an approach for improving the maximum achievable bandwidth and reducing end-to-end communication latency.

The achievable bandwidth in communication is efficiently improved for these two improvements. Experimental results indicate that the throughputs of HPPCP for 64Byte and 1500Byte messages are 900,000pps (about 439Mbps) and 82,000pps (about 939Mbps) respectively. It surpasses the traditional ones by a long way.