第31章　BPF：BSD 分组过滤程序

31.1　引言

BSD分组过滤程序(BPF)是一种软件设备，用于过滤网络接口的数据流，即给网络接口加上“开关”。应用进程打开/dev/bpf0、/dev/bpf1等等后，可以读取BPF设备。每个应用进程一次只能打开一个BPF设备。

因为每个BPF设备需要8192字节的缓存，系统管理员一般限制 BPF设备的数目。如果open返回EBUSY，说明该设备已被使用，应用进程应该试着打开下一 BPF设备，直到open成功为止。

通过若干ioctl命令，可以配置 BPF设备，把它与某个网络接口相关连，并安装过滤程序，从而能够选择性地接收输入的分组。 BPF设备打开后，应用进程通过读写设备来接收分组，或将分组放入网络接口队列中。

我们将一直使用“分组”，尽管“帧”可能更准确一些，因为 BPF工作在数据链路层，在发送和接收的数据帧中包含了链路层的首部。

BPF设备工作的前提是网络接口必须能够支持 BPF。第3章中提到以太网、 SLIP和环回接口的驱动程序都调用了 bpfattach，用于配置读取 BPF设备的接口。本节中，我们将介绍BPF设备驱动程序是如何组织的，以及数据分组在驱动程序和网络接口之间是如何传递的。

BPF一般情况下用作诊断工具，查看某个本地网络上的流量，卷 1附录A 介绍的tcpdump程序是此类工具中最好的一个。通常情况下，用户感兴趣的是一组指定主机间交互的分组，或者某个特定协议，甚至某个特定 TCP连接上的数据流。BPF设备经过适当配置，能够根据过滤程序的定义丢弃或接受输入的分组。过滤程序的定义类似于伪机器指令， BPF的细节超出了本书的讨论范围，感兴趣的读者请参阅 bpf(4)和[McCanne and Jacobson 1993]。

31.2　代码介绍

下面将要介绍的有关 BPF设备驱动程序的代码，包括两个头文件和一个 C文件，在图31-1中给出。

文　件	描　述
net/bpf.h	BPF常量
net/bpfdesc.h	BPF结构
net/bpf.c	BPF设备支持

图31-1　本章讨论的文件

31.2.1　全局变量

本章用到的全局变量在图31-2中给出。

变 量	数 据 类 型	描 述
bpf_iflist	struct bpf_if *	支持BPF的接口组成的链表
bpf_dtab	struct bpf_d []	BPF描述符数组
bpf_bufsize	int	BPF缓存大小默认值

图31-2 本章用到的全局变量

31.2.2 统计量

图31-3列出了bpf_d结构中为每个活动的BPF设备维护的两个统计量。

bpf_d成员变量	描 述
bd_rcount	从网络接口接收的分组的数目
bd_dcount	由于缓存空间不足而丢弃的分组的数目

图31-3 本章讨论的统计值

本章的其余内容分为4个部分：

- BPF接口结构；
- BPF设备描述符；
- BPF输入处理；和
- BPF输出处理。

31.3 bpf_if结构

BPF维护一个链表，包括所有支持BPF的网络接口。每个接口都由一个bpf_if结构描述，全局指针bpf_iflist指向表中的第一个结构。图31-4给出了BPF接口结构。

bpfdesc.h

```
67 struct bpf_if {
68     struct bpf_if *bif_next;    /* list of all interfaces */
69     struct bpf_d *bif_dlist;    /* descriptor list */
70     struct bpf_if **bif_driverp;    /* pointer into softc */
71     u_int   bif_dlt;            /* link layer type */
72     u_int   bif_hdrlen;         /* length of header (with padding) */
73     struct ifnet *bif_ifp;      /* correspoding interface */
74 };
```

bpfdesc.h

图31-4 bpf_if 结构

67-79 bif_next指向链表中的下一个BPF接口结构。bif_dlist指向另一个链表，包括所有已打开并配置过的BPF设备。

70 如果某个网络接口已配置了BPF设备，即被加上了开关，则bif_driverp将指向ifnet结构中的bpf_if指针。如果网络接口还未加上开关，*bif_driverp为空。为某个网络接口配置BPF设备时，*bif_driverp将指向bif_if结构，从而告诉接口可以开始向BPF传递分组。

71 接口类型保存在bif_dlt中。图31-5中列出了前面提到的几个接口所分别对应的常量值。

bif_dlt	描 述
DLT_EN10MB	10 Mb以太网接口
DLT_SLIP	SLIP接口
DLT_NULL	环回接口

图31-5 bif_dlt 值

72-74 BPF接受的所有分组都有一个附加的BPF首部。bif_hdrlen等于首部大小。最后，bif_ifp指向对应接口的ifnet结构。

图31-6给出了每个输入分组中附加的bpf_hdr结构。

bpf.h

```
122 struct bpf_hdr {
123     struct timeval bh_tstamp;    /* time stamp */
124     u_long  bh_caplen;           /* length of captured portion */
125     u_long  bh_datalen;          /* original length of packet */
126     u_short bh_hdrlen;           /* length of bpf header (this struct plus
127                                     alignment padding) */
128 };
```

bpf.h

图31-6 bpf_hdr 结构

122-128 bh_tstamp记录了分组被捕捉的时间。bh_caplen等于BPF保存的字节数，bh_datalen等于原始分组中的字节数。bh_headlen等于bpf_hdr的大小加上所需填充字节的长度。它用于解释从BPF设备中读取的分组，应该等同于接收接口的bif_hdrlen。

图31-7给出了bpf_if结构是如何与前述3个接口(le_softc [0]、sl_softc[0]和loif)的ifnet结构建立连接的。

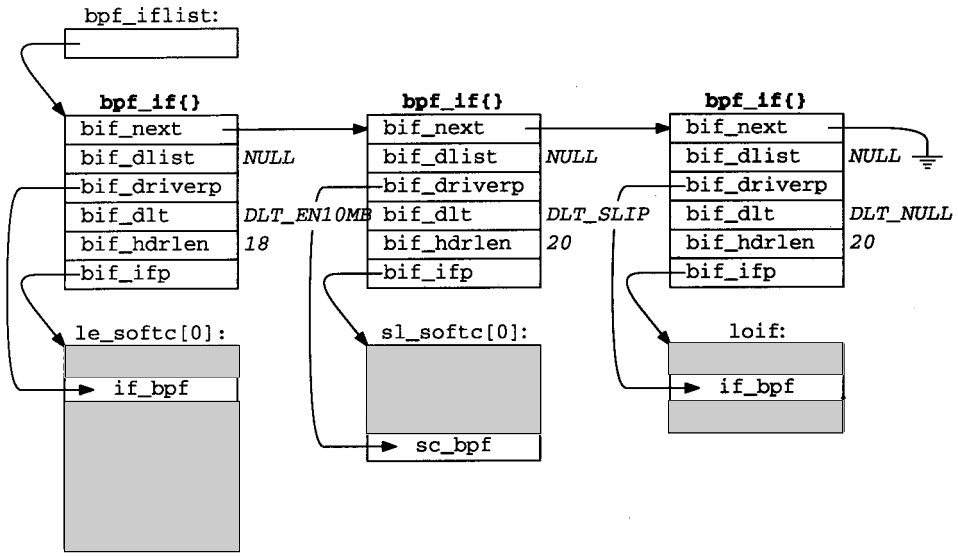

图31-7 bpf_if 和ifnet 结构

注意，bif_driverp指向网络接口的if_bpf和sc_bpf指针，而不是接口结构。

SLIP设备使用sc_bpf，而不是if_bpf。这可能是因为SLIP BPF代码完成时，

if_bpf成员变量还未加入到ifnet结构中。Net/2中的ifnet结构不包括if_bpf成员。

按照各接口驱动程序调用bpfattach时给出的信息，对3个接口初始化链路类型和首部长度成员变量。

第3章介绍了bpfattach被以太网、SLIP和环回接口的驱动程序调用。每个设备驱动程序初始化调用bpfattach时，将构建BPF接口结构链表。图31-8给出了该函数。

bpf.c

```
1053 void
1054 bpfattach(driverp, ifp, dlt, hdrlen)
1055 caddr_t *driverp;
1056 struct ifnet *ifp;
1057 u_int   dlt, hdrlen;
1058 {
1059     struct bpf_if *bp;
1060     int     i;
1061     bp = (struct bpf_if *) malloc(sizeof(*bp), M_DEVBUF, M_DONTWAIT);
1062     if (bp == 0)
1063         panic("bpfattach");

1064     bp->bif_dlist = 0;
1065     bp->bif_driverp = (struct bpf_if **) driverp;
1066     bp->bif_ifp = ifp;
1067     bp->bif_dlt = dlt;

1068     bp->bif_next = bpf_iflist;
1069     bpf_iflist = bp;

1070     *bp->bif_driverp = 0;

1071     /*
1072      * Compute the length of the bpf header.  This is not necessarily
1073      * equal to SIZEOF_BPF_HDR because we want to insert spacing such
1074      * that the network layer header begins on a longword boundary (for
1075      * performance reasons and to alleviate alignment restrictions).
1076      */
1077     bp->bif_hdrlen = BPF_WORDALIGN(hdrlen + SIZEOF_BPF_HDR) - hdrlen;

1078     /*
1079      * Mark all the descriptors free if this hasn't been done.
1080      */
1081     if (!D_ISFREE(&bpf_dtab[0]))
1082         for (i = 0; i < NBPFILTER; ++i)
1083             D_MARKFREE(&bpf_dtab[i]);

1084     printf("bpf: %s%d attached\n", ifp->if_name, ifp->if_unit);
1085 }
```

bpf.c

图31-8 bpfattach 函数

1053-1063 每个支持BPF的设备驱动程序都将调用 bpfattach。第一个参数是保存在bif_driverp的指针(图31-4给出)，第二个参数指向接口的ifnet结构，第三个参数确认数据链路层类型，第四个参数传递分组中的数据链路首部大小，为接口分配一个新的 bpf_if结构。

1. 初始化bpf_if结构

1064-1070 bpf_if结构根据函数的参数进行初始化，并插入到 BPF接口链表，bpf_iflist，的表头。

2. 计算BPF首部大小

1071-1077 设定bif_hdrlen大小，强迫网络层首部 (如IP首部)从一个长字的边界开始。这样可以提高性能，避免为 BPF加入不必要的对齐限制。图 31-9列出了在前述3个接口上，各自捕捉到的BPF分组的总体结构。

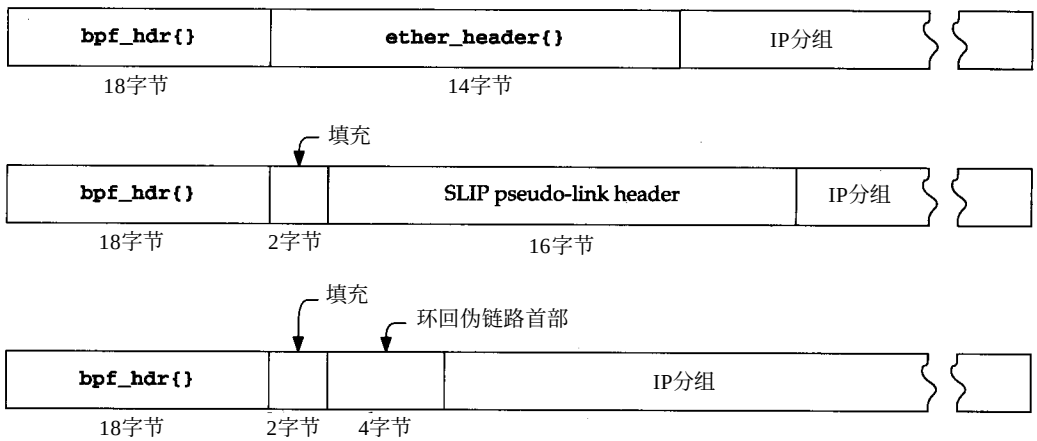

图31-9 BPF分组结构

ether_header结构在图4-10中给出，SLIP伪链路首部在图5-14中给出，而环回接口伪链路首部在图5-28中给出。

请注意，SLIP和环回接口分组需要填充2字节，以强迫IP首部按4字节对齐。

3. 初始化bpf_dtab表

1078-1083 代码初始化图 31-10中给出的 BPF描述符表。注意，仅在第一次调用bpfattach时进行初始化，后续调用将跳过初始化过程。

4. 打印控制台信息

1084-1085 系统向控制台输出一条短信息，宣告接口已配置完毕，可以支持 BPF。

31.4 bpf_d结构

为了能够选择性地接收输入报文，应用进程首先打开一个 BPF设备，调用若干ictl命令规定BPF过滤程序的条件，指明接口、读缓存大小和超时时限。每个 BPF设备都有一个相关的bpf_d结构，如图31-10所示。

45-46 如果同一网络接口上配置了多个BPF设备，与之相应的bpf_d结构将组成一个链表。bd_next指向链表中的下一个结构。

分组缓存

47-52 每个bpf_d结构都有两个分组缓存。输入分组通常保存在 bd_sbuf所对应的缓存(存储缓存)中。另一个缓存要么对应于 bd_fbuf(空闲缓存)，意味着缓存为空；或者对应于bd_hbuf(暂留缓存)，意味着缓存中有分组等待应用进程读取。 bd_slen和bd_hlen分别记

录了保存在存储缓存和暂留缓存中的字节数。

bpfdesc.h

```
45 struct bpf_d {
46     struct bpf_d *bd_next;      /* Linked list of descriptors */
47     caddr_t bd_sbuf;            /* store slot */
48     caddr_t bd_hbuf;            /* hold slot */
49     caddr_t bd_fbuf;            /* free slot */
50     int     bd_slen;            /* current length of store buffer */
51     int     bd_hlen;            /* current length of hold buffer */

52     int     bd_bufsize;         /* absolute length of buffers */

53     struct bpf_if *bd_bif;      /* interface descriptor */
54     u_long  bd_rtout;           /* Read timeout in 'ticks' */
55     struct bpf_insn *bd_filter; /* filter code */
56     u_long  bd_rcount;          /* number of packets received */
57     u_long  bd_dcount;          /* number of packets dropped */

58     u_char  bd_promisc;         /* true if listening promiscuously */
59     u_char  bd_state;           /* idle, waiting, or timed out */
60     u_char  bd_immediate;       /* true to return on packet arrival */
61     u_char  bd_pad;             /* explicit alignment */
62     struct selinfo bd_sel;      /* bsd select info */
63 };
```

bpfdesc.h

图31-10 bpf_d 结构

如果存储缓存已满，它将被连接到bd_hbuf，而空闲缓存将被连接到bd_sbuf。当暂留缓存清空时，它会被连接到 bd_fbuf。宏ROTATE_BUFFERS负责把存储缓存连接到bd_hbuf，空闲缓存连接到bd_sbuf，并清空bd_fbuf。存储缓存满或者应用进程不想再等待更多的分组时调用该宏。

bd_bufsize记录与设备相连的两个缓存的大小，其默认值等于4096(BPF_BUFSIZE)字节。修改内核代码可以改变默认值大小，或者通过 BIOCSBLEN ioctl 命令改变某个特定BPF设备的bd_bufsize。BIOCGBLEN命令返回bd_bufsize的当前值，其最大值不超过32768 (BPF_MAXBUFSIZE)字节，最小值为32 (BPF_MINBUFSIZE)字节。

53-57 bd_bif指向BPF设备所对应的 bpf_if结构。 BIOCSETIF命令可指明设备。bd_rtout是等待分组时，延迟的滴答数。 bd_filter指向BPF设备的过滤程序代码。两个统计值，应用进程可通过 BIOCGSTATS命令读取，分别保存在 bd_rcount和bd_dcount中。

58-63 bd_promisc通过BIOCPROMISC命令置位，从而使接口工作在混杂(promiscuous)状态。bd_state未使用。bd_immediate通过BIOCIMMEDIATE命令置位，促使驱动程序收到分组后即返回，不再等待暂留缓存填满。 bd_pad填弃bpf_d结构，从而与长字边界对齐。bd_sel保存的selinfo结构，可用于select系统调用。我们不准备介绍如何对BPF设备使用select系统调用，16.13节已介绍了select的一般用法。

31.4.1 bpfopen函数

应用进程调用open，试图打开一个BPF设备时，该调用将被转到bpfopen(图31-11)。

256-263 系统编译时，BPF设备的数目受到NBPFILTER的限制。如果设备的最小设备号大

于NBPFILTER，则返回ENXIO，这是因为系统管理员创建的/dev/bpfx项数大于NBPFILTER的值。

bpf.c

```
256 int
257 bpfopen(dev, flag)
258 dev_t   dev;
259 int     flag;
260 {
261     struct bpf_d *d;

262     if (minor(dev) >= NBPFILTER)
263         return (ENXIO);
264     /*
265      * Each minor can be opened by only one process.  If the requested
266      * minor is in use, return EBUSY.
267      */
268     d = &bpf_dtab[minor(dev)];
269     if (!D_ISFREE(d))
270         return (EBUSY);

271     /* Mark "free" and do most initialization. */
272     bzero((char *) d, sizeof(*d));
273     d->bd_bufsize = bpf_bufsize;

274     return (0);
275 }
```

bpf.c

图31-11 bpfopen 函数

分配bpf_d结构

264-275 同一时间内，一个应用进程只能访问一个BPF设备。如果bpf_d结构已被激活，则返回EBUSY。应用程序，如tcpdump，收到此返回值时，会自动寻找下一个设备。如果该设备已存在，最小设备号所指定的bpf_dtab表中的项被清除，分组缓存大小复位为默认值。

31.4.2 bpfioctl函数

设备打开后，可通过ioctl命令进行配置。图31-12总结了与BPF设备有关的ioctl命令。图31-13给出了bpfioctl函数，只列出BIOCSETF和BIOCSETIF的处理代码，其他未涉及到的ioctl命令则被忽略。

命　令	第三个参数	函　数	描　述
FIONREAD	u_int	bpfioctl	返回暂留缓存和存储缓存中的字节数
BIOCGBLEN	u_int	bpfioctl	返回分组缓存大小
BIOCSBLEN	u_int	bpfioctl	设定分组缓存大小
BIOCSETF	struct bpf_program	bpf_setf	安装BPF程序
BIOCFLUSH		reset_d	丢弃挂起分组
BIOCPROMISC		ifpromisc	设定混杂方式
BIOCGDLT	u_int	bpfioctl	返回bif_dlt
BIOCGETIF	struct ifreq	bpf_ifname	返回所属接口的名称
BIOCSETIF	struct ifreq	bpf_setif	为网络接口添加设备
BIOCSRTIMEOUT	struct timeval	bpfioctl	设定“读”操作的超时时限
BIOCGRTIMEOUT	struct timeval	bpfioctl	返回“读”操作的超时时限
BIOCGSTATS	struct bpf_stat	bpfioctl	返回BPF统计值
BIOCIMMEDIATE	u_int	bpfioctl	设定立即方式
BIOCVERSION	struct bpf_version	bpfioctl	返回BPF版本信息

图31-12 BPF ioctl 命令

bpf.c

```
501 int
502 bpfioctl(dev, cmd, addr, flag)
503 dev_t   dev;
504 int     cmd;
505 caddr_t addr;
506 int     flag;
507 {
508     struct bpf_d *d = &bpf_dtab[minor(dev)];
509     int      s, error = 0;
510     switch (cmd) {
511         /*
512          * Set link layer read filter.
513          */
514     case BIOCSETF:
515         error = bpf_setf(d, (struct bpf_program *) addr);
516         break;
517         /*
518          * Set interface.
519          */
520     case BIOCSETIF:
521         error = bpf_setif(d, (struct ifreq *) addr);
522         break;

            /* other ioctl commands from Figure 31.12 */

668     default:
669         error = EINVAL;
670         break;
671     }
672     return (error);
673 }
```

bpf.c

图31-13 bpfioctl 函数

501-509 与bpfopen类似，通过最小设备号从bpf_dtab表中选取相应的bpf_d结构。整个命令处理是一个大的switch/case语句。我们给出了两个命令，BIOCSETF和BIOCSETIF，以及default子句。

510-522 bpf_setf函数安装由addr指向的过滤程序，bpf_setif建立起指定名称接口与bpf_d结构间的对应关系。本书中没有给出bpf_setf的实现代码。

668-673 如果命令未知，则返回EINVAL。

图31-14的例子中，bpf_setif已把bpf_d结构连接到LANCE接口上。

图中，bif_dlist指向bpf_dtab[0]，以太网接口描述符链表中的第一个也是仅有的一个描述符。在bpf_dtab[0]中，bd_sbuf和bd_hbuf成员分别指向存储缓存和暂留缓存。两个缓存大小都等于4096(bd_bufsize)字节。bd_bif回指接口的bpf_if结构。

ifnet结构(le_softc[0])中的if_bpf也指回bpf_if结构。如图4-19和图4-11所示，如果if_bpf非空，则驱动程序开始调用bpf_tap，向BPF设备传递分组。

图31-15接着图31-10，给出了打开第二个BPF设备，并连接到同一个以太网网络接口后的各结构变量的状态。

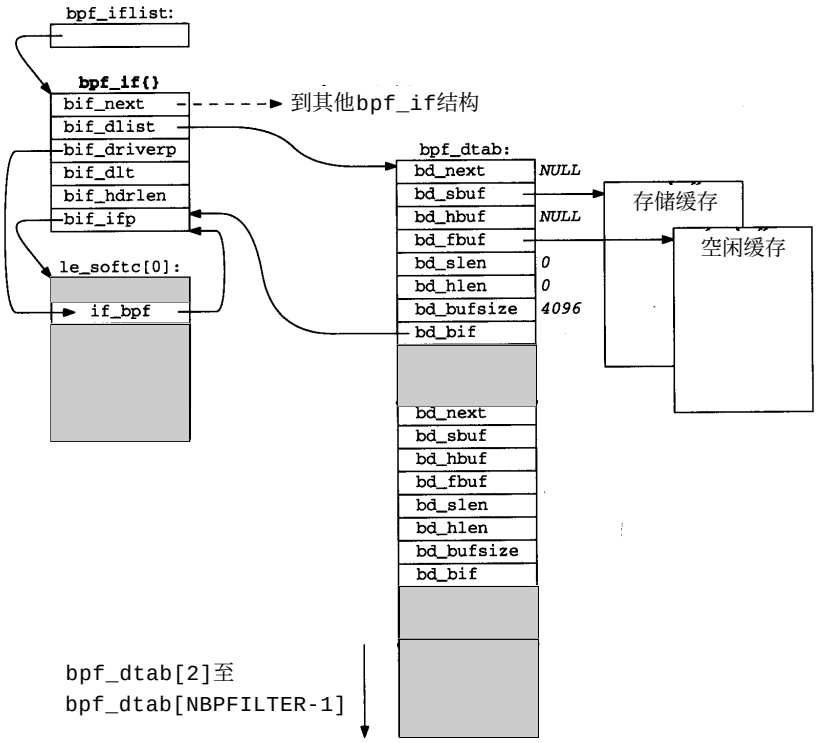

图31-14 连接到以太网接口的BPF设备

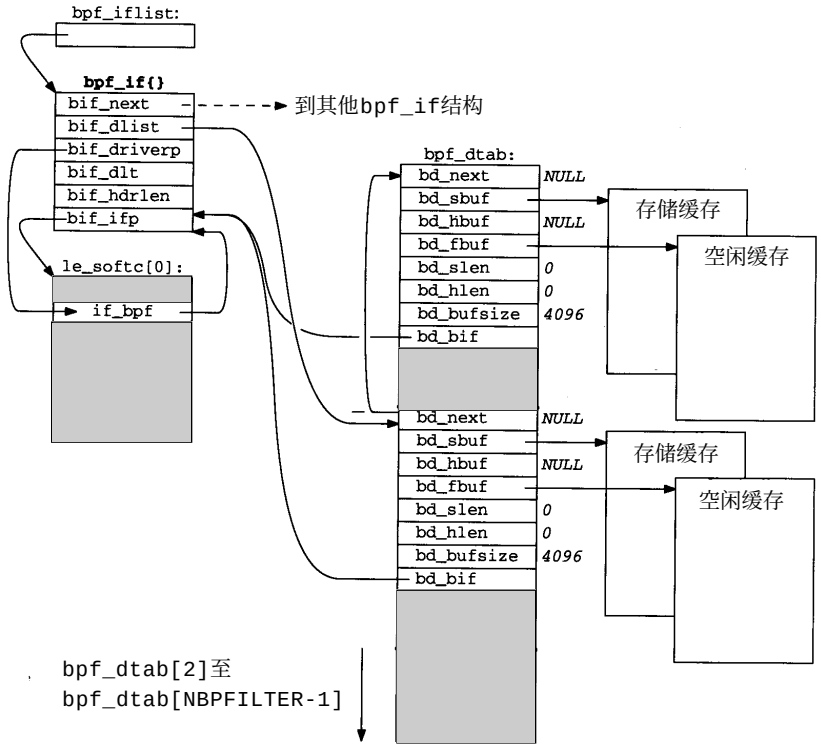

图31-15 连接到以太网接口的两个BPF设备

第二个BPF设备打开时，在bpf_dtab表中分配一个新的bpf_d结构，本例中为bpf_dtab[1]。因为第二个BPF设备也连接到同一个以太网接口，bif_dlist指向bpf_dtab[1]，并且bpf_dtab[1].bd_next指向bpf_dtab[0]，即以太网上对应的第一个BPF描述符。系统为新的描述符结构分别分配存储缓存和暂留缓存。

31.4.3 bpf_setif函数

bpf_setif函数，负责建立BPF描述符与网络接口间的连接，如图31-16所示。

bpf.c

```
721 static int
722 bpf_setif(d, ifr)
723 struct bpf_d *d;
724 struct ifreq *ifr;
725 {
726     struct bpf_if *bp;
727     char   *cp;
728     int     unit, s, error;

729     /*
730      * Separate string into name part and unit number.  Put a null
731      * byte at the end of the name part, and compute the number.
732      * If the a unit number is unspecified, the default is 0,
733      * as initialized above.  XXX This should be common code.
734      */
735     unit = 0;
736     cp = ifr->ifr_name;
737     cp[sizeof(ifr->ifr_name) - 1] = '\0';
738     while (*cp++) {
739         if (*cp >= '0' && *cp <= '9') {
740             unit = *cp - '0';
741             *cp++ = '\0';
742             while (*cp)
743                 unit = 10 * unit + *cp++ - '0';
744             break;
745         }
746     }
747     /*
748      * Look through attached interfaces for the named one.
749      */
750     for (bp = bpf_iflist; bp != 0; bp = bp->bif_next) {
751         struct ifnet *ifp = bp->bif_ifp;

752         if (ifp == 0 || unit != ifp->if_unit
753             || strcmp(ifp->if_name, ifr->ifr_name) != 0)
754             continue;
755         /*
756          * We found the requested interface.
757          * If it's not up, return an error.
758          * Allocate the packet buffers if we need to.
759          * If we're already attached to requested interface,
760          * just flush the buffer.
761          */
762         if ((ifp->if_flags & IFF_UP) == 0)
763             return (ENETDOWN);
```

图31-16 bpf_setif 函数

```
764             if (d->bd_sbuf == 0) {
765                 error = bpf_allocbufs(d);
766                 if (error != 0)
767                     return (error);
768             }
769             s = splimp();
770             if (bp != d->bd_bif) {
771                 if (d->bd_bif)
772                     /*
773                      * Detach if attached to something else.
774                      */
775                     bpf_detachd(d);

776                 bpf_attachd(d, bp);
777             }
778             reset_d(d);
779             splx(s);
780             return (0);
781         }
782         /* Not found. */
783         return (ENXIO);
784 }
```

bpf.c

图31-16 (续)

721-746 bpf_setif的第一部分完成ifreq结构(图4-23)中接口名的正文与数字部分的分离，数字部分保存在unit中。例如，如果ifr_name的头4字节为“sl1\0”，代码执行完毕后，将等于“sl\0\0”，且unit等于1。

1. 寻找匹配的ifnet结构

747-754 for循环用于在支持BPF的接口(bpf_iflist中)中查找符合ifreq定义的接口。

755-768 如果未找到匹配的接口，则返回ENETDOWN。如果接口存在，bpf_allocate为bpf_d分配空闲缓存和存储缓存，如果它们还未被分配的话。

2. 连接bpf_d结构

769-777 如果BPF设备还未与网络接口建立连接关系，或者连接的网络接口不是ifreq中指定的接口，则调用bpf_detachd丢弃原先的接口(如果存在)，并调用bpf_attachd将其连接到新的接口上。

778-784 reset_d复位分组缓存，丢弃所有在应用进程中等待的分组。函数返回0，说明处理成功；或者ENXIO，说明未找到指定接口。

31.4.4 bpf_attachd函数

图31-17给出的bpf_attachd函数，建立起BPF描述符与BPF设备和网络接口间的对应关系。

bpf.c

```
189 static void
190 bpf_attachd(d, bp)
191 struct bpf_d *d;
192 struct bpf_if *bp;
193 {
194     /*
195      * Point d at bp, and add d to the interface's list of listeners.
```

图31-17 bpf_attachd 函数

```
196      * Finally, point the driver's bpf cookie at the interface so
197      * it will divert packets to bpf.
198      */
199     d->bd_bif = bp;
200     d->bd_next = bp->bif_dlist;
201     bp->bif_dlist = d;

202     *bp->bif_driverp = bp;
203 }
```

bpf.c

图31-17 (续)

189-203 首先，令bd_bif指向网络接口的BPF接口结构。接着，bpf_d结构被插入到与设备对应的bpf_d结构链表的头部。最后，改变网络接口中的BPF指针，指向当前BPF结构，从而促使接口向BPF设备传递分组。

31.5 BPF的输入

一旦BPF设备打开并配置完毕，应用进程就通过read系统调用从接口中接收分组。BPF过滤程序复制输入分组，因此，不会干扰正常的网络处理。输入分组保存在与BPF设备相连的存储缓存和暂留缓存中。

31.5.1 bpf_tap函数

下面列出了图4-11中LANCE设备驱动程序调用bpf_tap的代码，并利用这一调用介绍bpf_tap函数。图4-11中的调用如下：

```
bpf_tap(le->sc_if.if_bpf, buf, len + sizeof(struct ether_header));
```

图31-18给出了bpf_tap函数。

bpf.c

```
869 void
870 bpf_tap(arg, pkt, pktlen)
871 caddr_t arg;
872 u_char *pkt;
873 u_int   pktlen;
874 {
875     struct bpf_if *bp;
876     struct bpf_d *d;
877     u_int   slen;
878     /*
879      * Note that the ipl does not have to be raised at this point.
880      * The only problem that could arise here is that if two different
881      * interfaces shared any data.  This is not the case.
882      */
883     bp = (struct bpf_if *) arg;
884     for (d = bp->bif_dlist; d != 0; d = d->bd_next) {
885         ++d->bd_rcount;
886         slen = bpf_filter(d->bd_filter, pkt, pktlen, pktlen);
887         if (slen != 0)
888             catchpacket(d, pkt, pktlen, slen, bcopy);
889     }
890 }
```

bpf.c

图31-18 bpf_tap 函数

869-882 第一个参数是指向bpf_if结构的指针，由bpfattach设定。第二个参数是指向进入分组的指针，包括以太网首部。第三个参数等于缓存中包含的字节数，本例中，等于以太网首部(14字节)大小加上以太网帧的数据部分。

向一个或多个BPF设备传递分组

883-890 for循环遍历连接到网络接口的BPF设备链表。对每个设备，分组被递交给bpf_filter。如果过滤程序接受了分组，它返回捕捉到的字节数，并调用catchpacket复制分组。如果过滤程序拒绝了分组，slen等于0，循环继续。循环终止时，bpf_tap返回。这一机制确保了同一网络接口上对应了多个BPF设备时，每个设备都能拥有一个独立的过滤程序。

环回驱动程序调用bpf_mtap，向BPF传递分组。这个函数与bpf_tap类似，然而是在mbuf链，而不是在一个内存的连续区域中复制分组。本书中不介绍这个函数。

31.5.2 catchpacket函数

图31-18中，过滤程序接受了分组后，将调用catchpacket，图31-19给出了这个函数。

bpf.c

```
946 static void
947 catchpacket(d, pkt, pktlen, snaplen, cpfn)
948 struct bpf_d *d;
949 u_char *pkt;
950 u_int   pktlen, snaplen;
951 void    (*cpfn) (const void *, void *, u_int);
952 {
953     struct bpf_hdr *hp;
954     int     totlen, curlen;
955     int     hdrlen = d->bd_bif->bif_hdrlen;
956     /*
957      * Figure out how many bytes to move.  If the packet is
958      * greater or equal to the snapshot length, transfer that
959      * much.  Otherwise, transfer the whole packet (unless
960      * we hit the buffer size limit).
961      */
962     totlen = hdrlen + min(snaplen, pktlen);
963     if (totlen > d->bd_bufsize)
964         totlen = d->bd_bufsize;
965     /*
966      * Round up the end of the previous packet to the next longword.
967      */
968     curlen = BPF_WORDALIGN(d->bd_slen);
969     if (curlen + totlen > d->bd_bufsize) {
970         /*
971          * This packet will overflow the storage buffer.
972          * Rotate the buffers if we can, then wakeup any
973          * pending reads.
974          */
975         if (d->bd_fbuf == 0) {
976             /*
977              * We haven't completed the previous read yet,
978              * so drop the packet.
979              */
980             ++d->bd_dcount;
```

图31-19 catchpacket 函数

```
981             return;
982         }
983         ROTATE_BUFFERS(d);
984         bpf_wakeup(d);
985         curlen = 0;
986     } else if (d->bd_immediate)
987         /*
988          * Immediate mode is set.  A packet arrived so any
989          * reads should be woken up.
990          */
991         bpf_wakeup(d);
992     /*
993      * Append the bpf header.
994      */
995     hp = (struct bpf_hdr *) (d->bd_sbuf + curlen);
996     microtime(&hp->bh_tstamp);
997     hp->bh_datalen = pktlen;
998     hp->bh_hdrlen = hdrlen;
999     /*
1000     * Copy the packet data into the store buffer and update its length.
1001     */
1002    (*cpfn) (pkt, (u_char *) hp + hdrlen, (hp->bh_caplen = totlen - hdrlen));
1003    d->bd_slen = curlen + totlen;
1004 }
```

bpf.c

图31-19 (续)

946-955 catchpacket的参数包括：d，指向BPF设备结构的指针；pkt，指向进入分组的通用指针；pktlen，分组被接收时的长度；snaplen，从分组中保存下来的字节数；cpfn，函数指针，把分组从pkt中复制到一块连续内存中。如果分组已经保存连续内存中，则cptn等于bcopy。如果分组被保存在mbuf中(pkt指向mbuf链表中的第一个mbuf，如环回驱动程序)，则cptn等于bpf_mcopy。

956-964 除了链路层首部和分组，catchpacket为每个分组添加bpf_hdr。从分组中保存的字节数等于snaplen和pktlen中较小的一个。处理过的分组和bpf_hdr必须能放入分组缓存中(bd_bufsize字节)。

1. 分组能否放入缓存

965-985 curlen等于存储缓存中已有的字节数加上所需的填充字节，以保证下一分组能从长字边界处开始存放。如果进入分组无法放入剩余的缓存空间，说明存储缓存已满。如果空闲缓存不可用(如应用进程正从暂留缓存中读取数据)，则进入分组被丢弃。如是空闲缓存可用，则调用ROTATE_BUFFERS宏轮转缓存，并通过bpf_wakeup唤醒所有等待输入数据的应用进程。

2. 立即方式处理

986-991 如果设备处于立即方式，则唤醒所有等待进程以处理进入分组——内核中没有分组的缓存。

3. 添加BPF 首部

992-1004 当前时间(microtime)、分组长度和首部长度均保存在bpf_hdr中。调用cptf所指的函数，把分组复制到存储缓存，并更新存储缓存的长度。因为在把分组从设备缓存传送到某个mbuf链表之前，bpf_tab已由leread直接调用，接收时间戳近似等于实际的接收

时间。

31.5.3 bpfread函数

内核把针对BPF设备的read转交给bpfread处理。通过BIOCSRTIMEOUT命令，BPF支持限时读取。这个“特性”也可通过select系统调用来实现，但至少tcpdump还是采用了BIOCSRTIMEOUT，而非select。应用进程提供一个读缓存，能够与设备的暂留缓存大小相匹配。BICOGBLEN命令返回缓存大小。一般情况下，读操作在存储缓存已满时返回。内核轮转缓存，把存储缓存转给暂留缓存，后者在read系统调用时被复制到应用进程提供的读缓存，同时BPF设备继续向存储缓存中存放进入分组。图31-20给出了bpfread。

bpf.c

```
344 int
345 bpfread(dev, uio)
346 dev_t   dev;
347 struct uio *uio;
348 {
349     struct bpf_d *d = &bpf_dtab[minor(dev)];
350     int      error;
351     int      s;

352     /*
353      * Restrict application to use a buffer the same size as
354      * as kernel buffers.
355      */
356     if (uio->uio_resid != d->bd_bufsize)
357         return (EINVAL);

358     s = splimp();
359     /*
360      * If the hold buffer is empty, then do a timed sleep, which
361      * ends when the timeout expires or when enough packets
362      * have arrived to fill the store buffer.
363      */
364     while (d->bd_hbuf == 0) {
365         if (d->bd_immediate && d->bd_slen != 0) {
366             /*
367              * A packet(s) either arrived since the previous
368              * read or arrived while we were asleep.
369              * Rotate the buffers and return what's here.
370              */
371             ROTATE_BUFFERS(d);
372             break;
373         }
374         error = tsleep((caddr_t) d, PRINET | PCATCH, "bpf", d->bd_rtout);
375         if (error == EINTR || error == ERESTART) {
376             splx(s);
377             return (error);
378         }
379         if (error == EWOULDBLOCK) {
380             /*
381              * On a timeout, return what's in the buffer,
382              * which may be nothing.  If there is something
383              * in the store buffer, we can rotate the buffers.
384              */
385             if (d->bd_hbuf)
```

图31-20 bpfread 函数

```
386                 /*
387                  * We filled up the buffer in between
388                  * getting the timeout and arriving
389                  * here, so we don't need to rotate.
390                  */
391                 break;
392             if (d->bd_slen == 0) {
393                 splx(s);
394                 return (0);
395             }
396             ROTATE_BUFFERS(d);
397             break;
398         }
399     }
400     /*
401      * At this point, we know we have something in the hold slot.
402      */
403     splx(s);

404     /*
405      * Move data from hold buffer into user space.
406      * We know the entire buffer is transferred since
407      * we checked above that the read buffer is bpf_bufsize bytes.
408      */
409     error = uiomove(d->bd_hbuf, d->bd_hlen, UIO_READ, uio);

410     s = splimp();
411     d->bd_fbuf = d->bd_hbuf;
412     d->bd_hbuf = 0;
413     d->bd_hlen = 0;
414     splx(s);

415     return (error);
416 }
```

bpf.c

图31-20 (续)

344-357 通过最小设备号在bpf_dtab中寻找相应的BPF设备。如果读缓存不能匹配BPF设备缓存的大小，则返回EINVAL。

1. 等待数据

358-364 因为多个应用进程能够从同一个BPF设备中读取数据，如果有某个进程已先读取了数据，while循环将强迫读操作继续。如果暂留缓存中存在数据，循环被跳过。这与两个应用进程通过两个不同的BPF设备过滤同一个网络接口的情况(见习题31.2)是不同的。

2. 立即方式

365-373 如果设备处于立即方式，且存储缓存中有数据，则轮回缓存，while循环被终止。

3. 无可用的分组

374-384 如果设备不处于立即方式，或者存储缓存中没有数据，则应用进程进入休眠状态，直到某个信号到达，读定时器超时，或者有数据到达暂留缓存。如果有信号到达，则返回EINTR或ERESTART。

记住，应用进程不会见到ERESTART，因为syscall函数将处理这一错误，且不会向应用进程返回这一错误。

4. 查看暂留缓存

385-391 如果定时器超时，且暂留缓存中存在数据，则循环终止。

5. 查看存储缓存

392-399 如果定时器超时，且存储缓存中没有数据，则read返回0。应用进程执行限时读取时，必须考虑到这种情况。如果定时器超时，且存储缓存中存在数据，则把存储缓存转给暂留缓存，循环终止。

如果tsleep返回正常且存在数据，同时while循环测试失败，则循环终止。

6. 分组可用

400-416 循环终止时，暂留缓存中已有数据。uiomove从暂留缓存中移出bd_hlen个字节，交给应用进程。把暂留缓存转给空闲缓存，清除缓存计数器，函数返回。 uiomove调用前的注释指出，uiomove通常能向应用进程复制bd_hlen字节的数据，因为前面已检查过读缓存大小，确保它大于BPF设备缓存的最大值，即bd_bufsize。

31.6 BPF的输出

最后，我们讨论如何向带有BPF设备的网络接口输出队列中添加分组。首先，应用进程必须构造完整的数据链路帧。对以太网而言，包括源和目的主机的硬件地址和数据帧类型(图4-8)。内核在把它放入接口的输出队列前不会修改链路帧。

bpfwrite函数

内核把应用进程的write系统调用转给图31-21给出的bpfwrite处理，数据帧被传给BPF设备。

bpf.c

```
437 int
438 bpfwrite(dev, uio)
439 dev_t   dev;
440 struct uio *uio;
441 {
442     struct bpf_d *d = &bpf_dtab[minor(dev)];
443     struct ifnet *ifp;
444     struct mbuf *m;
445     int     error, s;
446     static struct sockaddr dst;
447     int     datlen;

448     if (d->bd_bif == 0)
449         return (ENXIO);

450     ifp = d->bd_bif->bif_ifp;

451     if (uio->uio_resid == 0)
452         return (0);

453     error = bpf_movein(uio, (int) d->bd_bif->bif_dlt, &m, &dst, &datlen);
454     if (error)
455         return (error);

456     if (datlen > ifp->if_mtu)
457         return (EMSGSIZE);

458     s = splnet();
459     error = (*ifp->if_output) (ifp, m, &dst, (struct rtentry *) 0);
460     splx(s);
```

图31-21 bpfwrite 函数

```
461     /*
462      * The driver frees the mbuf.
463      */
464     return (error);
465 }
```

bpf.c

图31-21 (续)

1. 检查设备号

437-449 通过最小的设备号选择BPF设备，它必须已连接到某个网络接口。如果还没有，则返回ENXIO。

2. 向mbuf链中复制数据

450-457 如果write给出的写入数据长度等于0，则立即返回0。bpf_movein从应用进程复制数据到一个mbuf链表，并基于由bif_dlt传递的接口类型计算去除了链路层首部后的分组长度，并在datlen中返回该值。它还在dst中返回一个已初始化过的sockaddr结构。对以太网而言，这个地址结构的类型应该等于AF_UNSPEC，说明mbuf链中保存了外出数据帧的数据链路层首部。如果分组大于接口的MTU，则返回EMSGSIZE。

3. 分组排队

458-465 调用ifnet结构中指定的if_output函数，得到的mbuf链被提交给网络接口。对于以太网，if_output等于ether_output。

31.7 小结

本章中，我们讨论了如何配置BPF设备，如何向BPF设备递交进入数据帧，及如何在一个BPF设备上传送外出数据帧。

一个网络接口可以有多个BPF设备，每个BPF设备都有自己的过滤程序。存储缓存和暂留缓存最大限度地减少了应用进程为了处理进入数据帧而调用read的次数。

本章中只介绍了BPF的一些主要特性。有关BPF设备过滤程序代码的详细情况和其他一些特性，感兴趣的读者请参阅源代码和Net/3手册。

习题

31.1 为什么在分组存入BPF缓存之前，就能在catchpacket中调用bpf_wakeup？

31.2 图31-20中，我们提到可能会有两个进程在同一 BPF设备上等待数据。图31-11中，我们指出同一时间只能有一个应用进程可以打开一个特定的 BPF设备。为什么这两种说法都正确呢？

31.3 如果BIOCSETIF命令中指定的设备不支持BPF，会发生什么现象？