

第21章 ARP：地址解析协议

21.1 介绍

地址解析协议(ARP)用于实现 IP 地址到网络接口硬件地址的映射。常见的以太网网络接口硬件地址长度为 48 bit。ARP 同时也可以工作在其他类型的数据链路下，但在本章中，我们只考虑将 IP 地址映射到 48 bit 的以太网地址。ARP 在 RFC 826 [Plummer 1982] 中定义。

当某主机要向以太网中另一台主机发送 IP 数据时，它首先根据目的主机的 IP 地址在 ARP 高速缓存中查询相应的以太网地址，ARP 高速缓存是主机维护的一个 IP 地址到相应以太网地址的映射表。如果查到匹配的结点，则相应的以太网地址被写入以太网帧首部，数据报被加入到输出队列等候发送。如果查询失败，ARP 会先保留待发送的 IP 数据报，然后广播一个询问目的主机硬件地址的 ARP 报文，等收到回答后再将 IP 数据报发送出去。

以上只是简要描述了 ARP 协议的基本工作过程，下面我们将结合 Net/3 中的 ARP 实现来详细描述其具体细节。卷 1 的第 4 章包含了 ARP 的例子。

21.2 ARP 和路由表

Net/3 中 ARP 的实现是和路由表紧密关联的，这也是为什么我们要在描述路由表结构之后再来讲解 ARP 的原因。图 21-1 显示了本章中我们描述 ARP 要用到的一个例子。整个图是与本书中用到的网络实例相对应的，它显示了 bsd1 主机上当前 ARP 缓存的相关结构。其中 Ifnet、ifaddr 和 in_ifaddr 结构是由图 3-32 和图 6-5 简化而来的，所以在这里忽略了在第 3 章和第 6 章中描述过的这三个结构中的某些细节。例如，图中没有画出在两个 ifaddr 结构之后的 sockaddr_dl 结构——而仅仅是概述了这两个结构中的相应信息。同样，我们也仅仅是概述了三个 in_ifaddr 结构中的信息。

下面，我们简要概述图中的有关要点。细节部分将随着本章的进行而详细展开。

1) llinfo_arp 结构的双向链表包含了每一个 ARP 已知的硬件地址的少量信息。同名全局变量 llinfo_arp 是该链表的头结点，图中没有画出第一位的 la_prev 指针指向最后一项，最后一项的 la_next 指针指向第一项。该链表由 ARP 时钟函数每隔 5 分钟处理一次。

2) 每一个已知硬件地址的 IP 地址都对应一个路由表结点 (rtentry 结构)。llinfo_arp 结构的 la_rt 指针成员用来指向相应的 rtentry 结构，同样地，rtentry 结构的 rt_llinfo 指针成员指向 llinfo_arp 结构。图中对应主机 sun(140.252.13.33)、svr4(140.252.13.34) 和 bsd1(140.252.13.35) 的三个路由表结点各自具有相应的 llinfo_arp 结构。如图 18-2 所示。

3) 而在图的最左边第四个路由表结点则没有对应的 llinfo_arp 结构，该结点对应于本地以太网(140.252.13.32)的路由项。该结点的 rt_flags 中设置了 C 比特，表明该结点是被用来复制形成其他结点的。设置接口 IP 地址功能的 in_ifinit 函数(图 6-19)通过调用 rtinit 函数来创建该结点。其他三个结点是主机路由结点 (H 标志)，并由 bsd1 向其他机器发送数据

时通过ARP间接调用路由相关函数产生的(L标志)。

4) `rtentry`结构中的`rt_gateway`指针成员指向一个`sockaddr_dl`结构变量。如果保存物理地址长度的结构`sdl_alen`成员为6,那么`sockaddr_dl`结构就包含相应的硬件地址信息。

5) 路由结点变量的`rt_ifp`成员的相应指针成员指向对应网络设备接口的`ifnet`结构。中间的两个路由结点对应的是以太网上的其他主机,这两个结点都指向`le_softc[0]`。而右边的路由结点对应的是`bsdi`,指向环回结构`loif`。因为`rt_ifp.if_output`指向输出函数,所以目的为本机的数据报被路由至环回接口。

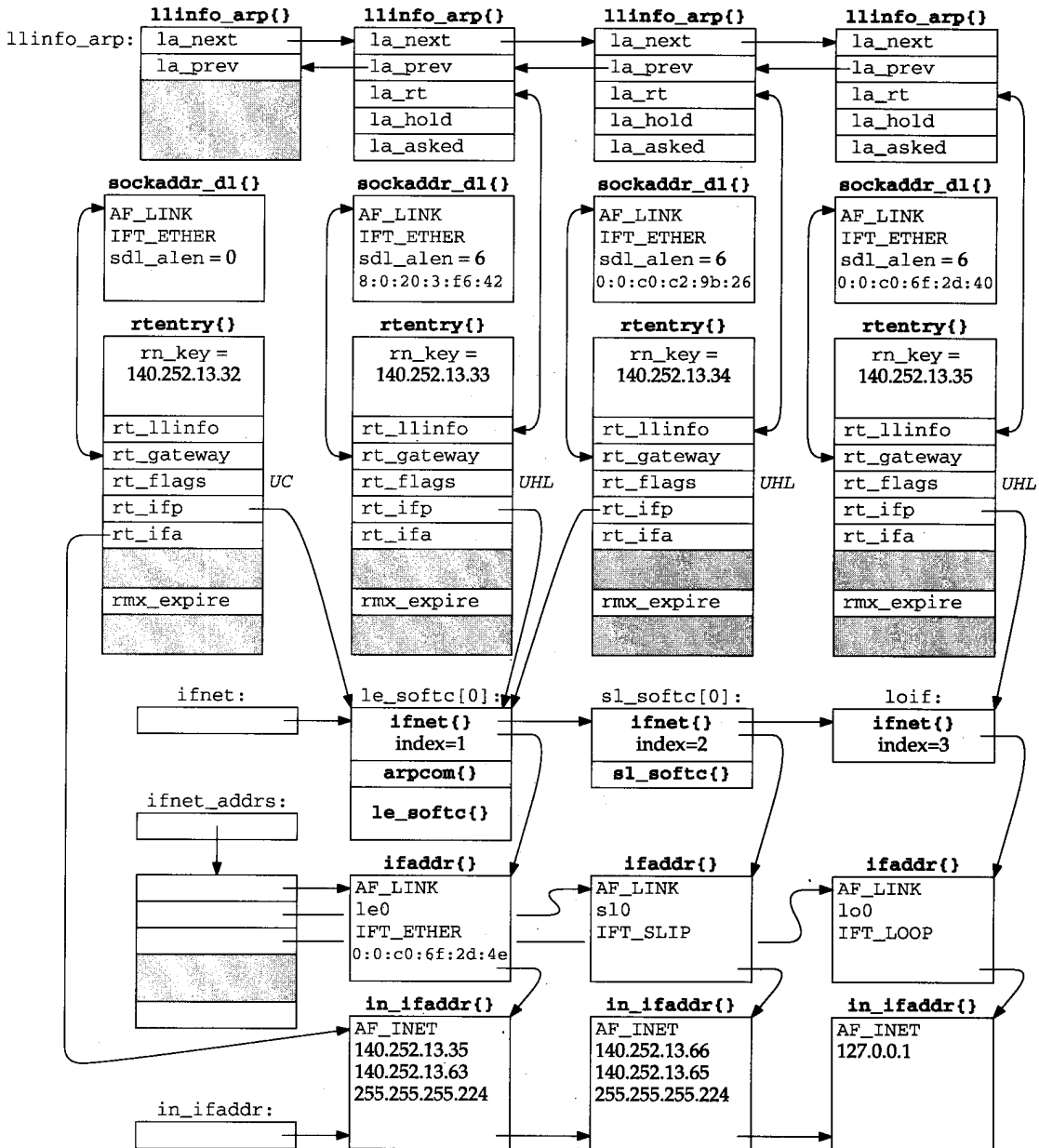


图21-1 ARP与路由表和接口结构的关系

6) 每一个路由结点还有指向相应的 `in_ifaddr` 结构的指针变量 (图6-8中指出了 `in_ifaddr` 结构内的第一个成员是一个 `ifaddr` 结构, 因此, `rt_ifa` 同样是指向了 `ifaddr` 结构变量)。在本图中, 我们只显示一个路由结点的相应指向, 其余的路由结点具有同样的性质。而一个接口如 `le0`, 可以同时设置多个IP地址, 每个IP地址都有对应的 `in_ifaddr` 结构, 这就是为什么除了 `rt_ifp` 之外还需要 `rt_ifa` 的原因。

7) `la_hold` 成员是指向 `mbuf` 链表的指针。当要向某个IP传送数据报时, 就需要广播一个ARP请求。当内核等待ARP回答时, 存放该待发数据报的 `mbuf` 链的头结点的地址信息就存放在 `la_hold` 里。当收到ARP回答后, `la_hold` 指向的 `mbuf` 链表中的IP数据被发送出去。

8) 路由表结点中 `rt_metric` 结构的变量 `rmx_expire` 存放的是与对应的ARP结点相关的定时信息, 用来实现删除超时 (通常20分钟) 的ARP结点。

在4.3BSD Reno中, 路由表结构定义有了很大的变化, 但4.3BSD Reno和Net/2.4.4BSD中依然定义有ARP缓存, 只是去除了作为单独结构的ARP缓存链表, 而把ARP信息放在了路由表结点里。

在Net/2中, ARP表是一个结构数组, 其中每个元素包含有以下成员: IP地址、以太网地址、定时器、标志和一个指向 `mbuf` 的指针 (类似于图21-1中的 `la_hold` 成员)。在Net/3中, 我们可以看到, 这些信息被分散到多个相互链接的结构里。

21.3 代码介绍

如图21-2所示, 共有包含9个ARP函数的一个C文件和两个头文件。

文 件	描 述
<code>net/if_arp.h</code>	<code>arphdr</code> 结构的定义
<code>netinet/if_ether.h</code>	多个结构和常量的定义
<code>netinet/if_ether.c</code>	ARP函数

图21-2 本章中讨论的文件

图21-3显示了ARP函数与其他内核函数的关系。该图中还说明了ARP函数与第19章中某些子函数的关系, 下面将逐步解释这些关系。

21.3.1 全局变量

本章中将介绍10个全局变量, 如图21-4所示。

21.3.2 统计量

保存ARP的统计量有两个全局变量: `arp_inuse` 和 `arp_allocated`, 如图21-4所示。前者用来记录当前正在使用的ARP结点数, 后者用来记录在系统初始化时分配的ARP结点数。两个统计数都不能由 `netstat` 程序输出, 但可以通过调试器来查看。

可以使用命令 `arp -a` 来显示当前ARP缓存的信息, 该命令使用 `sysctl` 系统调用, 参数如图19-36所示。图21-5显示该命令的一个输出结果。

由于图18-2中对应多播组224.0.0.1的相应路由表项设置了L标志, 而同时由于 `arp` 程序查询带有 `RTF_LLINFO` 标志位的ARP结点, 所以该程序也输出多播地址。后面我们将解释为什么该表项标识为 “incomplete”, 而在它上面的表项是 “permanent”。

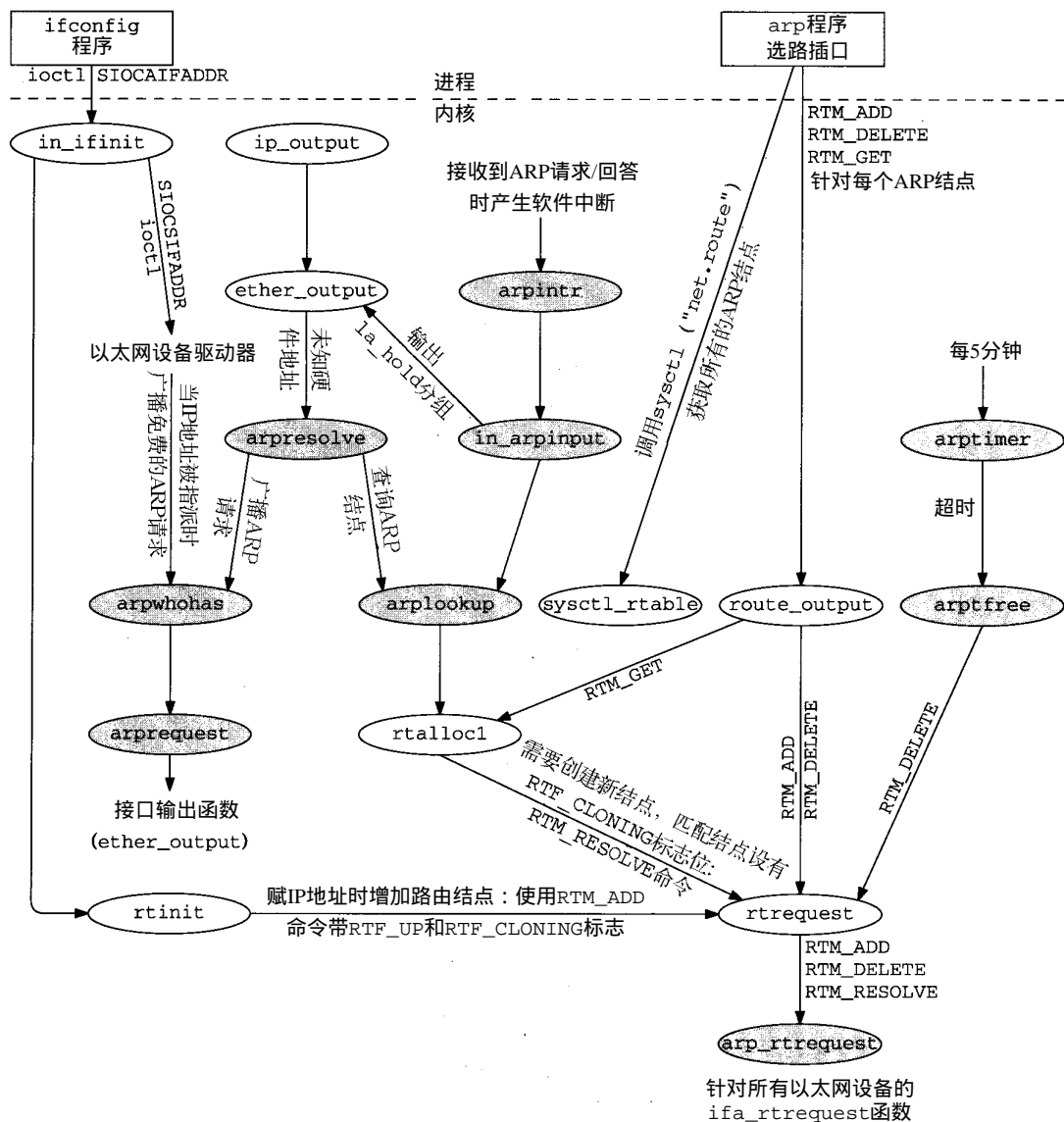


图21-3 ARP函数和内核中其他函数的关系

变 量	数 据 类 型	描 述
llinfo_arp	struct llinfo_arp	双向链接表的表头
arpintrq	struct ifqueue	来自以太网设备驱动程序的ARP输入队列
arpt_prune	Int	检查ARP链表的时间间隔的分钟数(5)
arpt_keep	Int	ARP结点的有效时间的分钟数(20)
arpt_down	Int	ARP洪泛算法的时间间隔的秒数(20)
arp_inuse	Int	正在使用的ARP结点数
arp_allocated	Int	已经分配的ARP结点数
arp_maxtries	Int	对一个IP地址发送ARP请求的重试次数(5)
arpinit_done	Int	初始化标志
uselookback	int	对本机使用环回(默认)

图21-4 本章介绍的全局变量

```
bsdi $ arp -a
sun.tuc.noao.edu (140.252.13.33) at 8:0:20:3:f6:42
svr4.tuc.noao.edu (140.252.13.34) at 0:0:c0:c2:9b:26
bsdi.tuc.noao.edu (140.252.13.35) at 0:0:c0:6f:2d:40 permanent
ALL-SYSTEMS.MCAST.NET (224.0.0.1) at (incomplete)
```

图21-5 与图18-2相应的arp -a命令的输出

21.3.3 SNMP变量

在卷1的25.8节中我们讲过，最初的SNMP MIB定义了一个地址映射组，该组对应的是系统的当前ARP缓存信息。在MIB-II中不再使用该组，而用各个网络协议组（如IP组）分别包含地址映射表来替代。注意，从Net/2到Net/3，将单独结构的ARP缓存演化为在路由表中集成的ARP信息是与SNMP的变化并行的。

IP地址映射表，index = <ipNetToMediaIfIndex>.<ipNetToMediaNetAddress>		
名 称	成 员	描 述
ipNetToMediaIfIndex	if_index	相应接口：ifIndex
ipNetToMediaPhysAddress	rt_gateway	硬件地址
ipNetToMediaNetAddress	rt_key	IP地址
ipNetToMediaType	rt_flags	映射类型：1=其他，2=失效，3=动态，4=静态(见正文)

图21-6 IP地址映射表：ipNetToMediaTable

图21-6所示的是MIB-II中的一个IP地址映射表，ipNetToMediaTable，该表保存的值来自于路由表结点和相应的ifnet结构。

如果路由表结点的生存期为0，则被认为是永久的，也即静态的。否则就是动态的。

21.4 ARP 结构

在以太网中传送的ARP分组的格式图21-7所示。

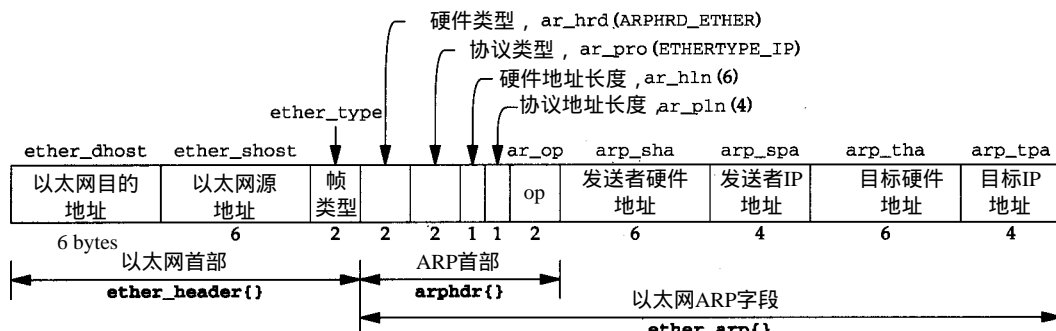


图21-7 在以太网上使用时ARP请求或回答的格式

结构ether_header定义了以太网帧首部；结构arphdr定义了其后的5个字段，其信息用于在任何类型的介质上传送ARP请求和回答；ether_arp结构除了包含arphdr结构外，还包含源主机和目的主机的地址。

结构arphdr的定义如图21-8所示。图21-7显示了该结构中的前4个字段。

```

45 struct arphdr {
46     u_short ar_hrd;           /* format of hardware address */
47     u_short ar_pro;           /* format of protocol address */
48     u_char  ar_hln;           /* length of hardware address */
49     u_char  ar_pln;           /* length of protocol address */
50     u_short ar_op;            /* ARP/RARP operation, Figure 21.15 */
51 };

```

if_arp.h

图21-8 arphdr 结构：通用的ARP请求/回答数据首部

图21-9显示了ether_arp结构的定义，其中包含了 arphdr结构、源主机和目的主机的IP地址和硬件地址。注意，ARP用硬件地址来表示48 bit以太网地址，用协议地址来表示32 bit IP地址。

```

79 struct ether_arp {
80     struct arphdr ea_hdr;      /* fixed-size header */
81     u_char  arp_sha[6];        /* sender hardware address */
82     u_char  arp_spa[4];        /* sender protocol address */
83     u_char  arp_tha[6];        /* target hardware address */
84     u_char  arp_tpa[4];        /* target protocol address */
85 };

86 #define arp_hrd ea_hdr.ar_hrd
87 #define arp_pro ea_hdr.ar_pro
88 #define arp_hln ea_hdr.ar_hln
89 #define arp_pln ea_hdr.ar_pln
90 #define arp_op  ea_hdr.ar_op

```

if_ether.h

图21-9 ether_arp 结构

每个ARP结点中，都有一个llinfo_arp结构，如图21-10所示。所有这些结构组成的链表的头结点是作为全局变量分配的。我们经常把该链接表称为 ARP高速缓存，因为在图21-1中，只有该数据结构是与ARP结点一一对应的。

```

103 struct llinfo_arp {
104     struct llinfo_arp *la_next;
105     struct llinfo_arp *la_prev;
106     struct rtentry *la_rt;
107     struct mbuf *la_hold;      /* last packet until resolved/timeout */
108     long la_asked;             /* #times we've queried for this addr */
109 };

110 #define la_timer la_rt->rt_rmx.rmx_expire /* deletion time in seconds */

```

if_ether.h

图21-10 llinfo_arp结构

在Net/2及以前的系统中，很容易识别作为 ARP高速缓存的数据结构，因为每一个ARP结点的信息都存放在单一的结构中。而Net/3则把ARP信息存放在多个结构中，没有哪个数据结构被称为 ARP高速缓存。但是为了讨论方便，我们依然用 ARP高速缓存的概念来表示一个ARP结点的信息。

104-106 该双向链接表的前两项由insque和remque两个函数更新。la_rt指向相关的路

由表结点，该路由表结点的 `rt_llinfo` 成员指向 `la_rt`。

107 当ARP接收到一个要发往其他主机的IP数据报，且不知道相应硬件地址时，必须发送一个ARP请求，并等待回答。在等待ARP回答时，指向待发数据报的指针存放在 `la_hold` 中。收到回答后，`la_hold` 所指的数据报被发送出去。

108-109 `la_asked` 记录了连续为某个IP地址发送请求而没有收到回答的次数。在图 21-24 中，我们可以看到，当这个数值达到某个限定值时，我们就认为该主机是关闭的，并在其后一段时间内不再发送该主机的ARP请求。

110 这个定义使用路由结点中 `rt_metrics` 结构的 `rmx_expire` 成员作为ARP定时器。当值为0时，ARP项被认为是永久的；当为非零时，值为当结点到期时算起的秒数。

21.5 arpwhoas函数

`arpwhoas` 函数通常由 `arpresolve` 调用，用于广播一个ARP请求。如图 21-11 所示。它还可由每个以太网设备驱动程序调用，在将IP地址赋予该设备接口时主动发送一个地址联编信息(图6-28中的 `SIOCSIFADDR` ioctl)。主动发送地址联编信息不但可以检测在以太网中是否存在IP地址冲突，并且可以使其他机器更新其相应信息。`arpwhoas` 只是简单调用下一部分将要介绍的 `arprequest` 函数。

```
196 void  
197 arpwhoas(ac, addr)  
198 struct arpcom *ac;  
199 struct in_addr *addr;  
200 {  
201     arprequest(ac, &ac->ac_ipaddr.s_addr, &addr->s_addr, ac->ac_enaddr);  
202 }
```

if_ether.c

图21-11 `arpwhoas` 函数：广播一个ARP请求

196-202 `arpcom` 结构(图3-26)对所有以太网设备是通用的，是 `le_softc` 结构(图3-20)的一部分。`ac_ipaddr` 成员是接口的IP地址的复制，当 `SIOCSIFADDR` ioctl 执行时由驱动程序填写(图6-28)。`ac_enaddr` 是该设备的以太网地址。

该函数的第二个参数 `addr`，是ARP请求的目的IP地址。在主动发送动态联编信息时，`addr` 等于 `ac_ipaddr`，所以 `arprequest` 的第二和第三个参数是一样的，即发送IP地址和目的IP地址在主动发送动态联编信息时是一样的。

21.6 arprequest函数

`arprequest` 函数由 `arpwhoas` 函数调用，用于广播一个ARP请求。该函数建立一个ARP请求分组，并将它传送到接口的输出函数。

在分析代码之前，我们先来看一下该函数建立的数据结构。传送ARP请求需要调用以太网设备的接口输出函数 `ether_output`。`ether_output` 的一个参数是 `mbuf`，它包含待发数据，即图 21-7 中以太网类型字段后的所有内容。另外一个参数包含目的地址的端口地址结构。通常情况下，该目的地址是IP地址(例如，在图 21-3 中，`ip_output` 调用 `ether_output`)。特殊情况下，端口地址的 `sa_family` 被设为 `AF_UNSPEC`，即告知 `ether_output` 它所带的是一个已填充的以太网帧首部，包含了目的主机的硬件地址，这就

防止了ether_output去调用arpreslove而导致死循环。图21-3中没有显示这种循环，在arprequest下面的接口输出函数是 ether_output。如果ether_output再去调用arpreslove，将导致死循环。

图21-12显示了该函数建立的两个数据结构 mbuf和sockaddr。另外还有两个函数中用到的指针eh和ea。

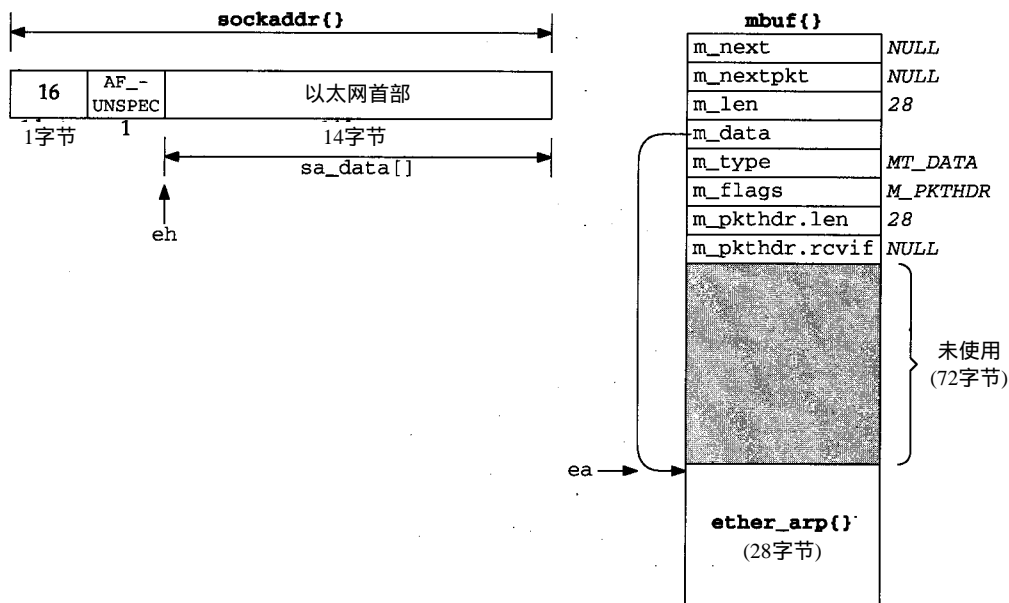


图21-12 arprequest 建立的sockaddr 和mbuf

图21-13给出了arprequest函数的源代码。

```

209 static void
210 arprequest(ac, sip, tip, enaddr)
211 struct arpcom *ac;
212 u_long *sip, *tip;
213 u_char *enaddr;
214 {
215     struct mbuf *m;
216     struct ether_header *eh;
217     struct ether_arp *ea;
218     struct sockaddr sa;
219
220     if ((m = m_gethdr(M_DONTWAIT, MT_DATA)) == NULL)
221         return;
222     m->m_len = sizeof(*ea);
223     m->m_pkthdr.len = sizeof(*ea);
224     MH_ALIGN(m, sizeof(*ea));
225
226     ea = mtod(m, struct ether_arp *);
227     eh = (struct ether_header *) sa.sa_data;
228     bzero((caddr_t) ea, sizeof(*ea));
229
230     bcopy((caddr_t) etherbroadcastaddr, (caddr_t) eh->ether_dhost,
231         sizeof(eh->ether_dhost));

```

if_ether.c

图21-13 arprequest 函数：创建一个ARP请求并发送

```

229     eh->ether_type = ETHERTYPE_ARP;      /* if_output() will swap */

230     ea->arp_hrd = htons(ARPHRD_ETHER);
231     ea->arp_pro = htons(ETHERTYPE_IP);
232     ea->arp_hln = sizeof(ea->arp_sha); /* hardware address length */
233     ea->arp_pln = sizeof(ea->arp_spa); /* protocol address length */
234     ea->arp_op = htons(ARPOP_REQUEST);
235     bcopy((caddr_t) enaddr, (caddr_t) ea->arp_sha, sizeof(ea->arp_sha));
236     bcopy((caddr_t) sip, (caddr_t) ea->arp_spa, sizeof(ea->arp_spa));
237     bcopy((caddr_t) tip, (caddr_t) ea->arp_tpa, sizeof(ea->arp_tpa));

238     sa.sa_family = AF_UNSPEC;
239     sa.sa_len = sizeof(sa);

240     (*ac->ac_if.if_output) (&ac->ac_if, m, &sa, (struct rtentry *) 0);
241 }

```

if_ether.c

图21-13 (续)

1. 分配和初始化mbuf

209-223 分配一个分组数据首部的mbuf，并对两个长度字段赋值。MH_ALIGN将28字节的ether_arp结构置于mbuf的尾部，并相应地设置m_data指针的值。将该数据结构置于mbuf尾部，是为了允许ether_output预先考虑将14字节的以太网帧首部置于同一mbuf中。

2. 初始化指针

224-226 给ea和eh两个指针赋值，并将ether_arp结构的值赋为0。bzero的惟一目的是将目的硬件地址置0，该结构中其余8个字段已被设成相应的值。

3. 填充以太网帧首部

227-229 目的以太网地址设为以太网广播地址，并将以太网帧类型设为ETHERTYPE_ARP。注意代码中的注释，接口输出函数将该字段从主机字节序转化为网络字节序，该函数还将填充本机的以太网地址。图21-14显示了不同以太网帧类型字段的常量值。

常 量	值	描 述
ETHERTYPE_IP	0x0800	IP帧
ETHERTYPE_ARP	0x0806	ARP帧
ETHERTYPE_REVARP	0x8035	逆ARP帧
ETHERTYPE_IPTRAILERS	0x1000	尾部封装(已废弃)

图21-14 以太网帧类型字段

RARP 将硬件地址映射成IP地址，通常在无盘工作站系统引导时使用。一般来说，RARP部分不属于内核TCP/IP实现，所以本书将不作描述，卷1的第5章讲述了RARP的概念。

4. 填充ARP字段

230-237 填充了ether_arp的所有字段，除了ARP请求所要询问的目的硬件地址。常量ARPHRD_ETHER的值为1时，表示硬件地址的格式是6字节的以太网地址。为了表示协议地址是4字节的IP地址，arp_pro的值设为图21-14中所指的IP协议地址类型(0x800)。图21-15显示了不同的ARP操作码。本章中，我们将看到前两种。后两种在RARP中使用。

5. 填充sockaddr，并调用接口输出函数

238-241 接口地址结构的sa_family成员的值设为AF_UNSPEC，sa_member成员的值设为16。调用接口输出函数ether_output。

常 量	值	描 述
ARPOP_REQUEST	1	解析协议地址的ARP请求
ARPOP_REPLY	2	回答ARP请求
ARPOP_REVREQUEST	3	解析硬件地址的RARP请求
ARPOP_REVREPLY	4	回答RARP请求

图21-15 ARP 操作码

21.7 arpintr函数

在图4-13中,当ether_input函数接收到帧类型字段为ETHERTYPE_ARP的以太网帧时,产生优先级为NETISR_ARP的软件中断,并将该帧挂在ARP输入队列arpintrq的后面。当内核处理该软件中断时,调用arpintr函数,如图21-16所示。

```

319 void
320 arpintr()
321 {
322     struct mbuf *m;
323     struct arphdr *ar;
324     int s;
325
326     while (arpintrq.ifq_head) {
327         s = splimp();
328         IF_DEQUEUE(&arpintrq, m);
329         splx(s);
330         if (m == 0 || (m->m_flags & M_PKTHDR) == 0)
331             panic("arpintr");
332
333         if (m->m_len >= sizeof(struct arphdr) &&
334             (ar = mtod(m, struct arphdr *)) &&
335             ntohs(ar->ar_hrd) == ARPHRD_ETHER &&
336             m->m_len >= sizeof(struct arphdr) + 2*ar->ar_hln + 2*ar->ar_pln)
337
338             switch (ntohs(ar->ar_pro)) {
339                 case ETHERTYPE_IP:
340                 case ETHERTYPE_IPTRAILERS:
341                     in_arpinput(m);
342                     continue;
343             }
344
345         m_freem(m);
346     }
347 }

```

if_ether.c

if_ether.c

图21-16 arpintr 函数：处理包含ARP请求/回答的以太网帧

319-343 while循环一次处理一个以太网帧,直到处理完队列中的所有帧为止。只有当帧的硬件类型指明为以太网地址,并且帧的长度大于或等于 arphdr结构的长度加上两个硬件地址和两个协议地址的长度时,该帧才能被处理。如果协议地址的类型是 ETHERTYPE_IP或 ETHERTYPE_IPTRAILERS时,调用in_arpinput函数,否则该帧将被丢弃。

注意if语句中对条件的检测顺序。共两次检查帧的长度。首先,当帧长大于或等于 arphdr结构的长度时,才去检查帧结构中的其他字段;然后,利用 arphdr中的两个长度字段再次检查帧长。

21.8 in_arpinput函数

该函数由arpintr调用，用于处理接收到的ARP请求/回答。ARP本身的概念比较简单，但是加上许多规则后，实现就比较复杂，下面先来看一下两种典型情况：

- 1) 如果收到一个针对本机IP地址的请求，则发送一个回答。这是一种普通情况。很显然，我们将继续从那个主机收到数据报，随后也会向它回送报文。所以，如果我们还没有对应它的ARP结点，就应该添加一个ARP结点，因为这时我们已经知道了对方的IP地址和硬件地址。这会优化其后与该主机的通信。
- 2) 如果收到一个ARP回答，那么此时ARP结点是完整的，因此就知道了对方的硬件地址。该地址存放在sockaddr_dl结构中，所有发往该地址的数据报将被发送。ARP请求是被广播发送的，所以以太网上的所有主机都将看到该请求，当然包括那些非目的主机。回想一下arprequest函数，在发送ARP请求时，帧中包含着请求方的IP地址和硬件地址，这就产生了下面的情况：
- 3) 如果其他主机发送了一个ARP请求或回答，其中发送方的IP地址与本机相同，那么肯定有一个主机配置有误。Net/3将检测到该差错，并向管理员登记一个报文（这里我们不分请求或回答，因为in_arpinput不检查操作类型，但是ARP回答将被单播，只有目的主机才能收到信息）。
- 4) 如果主机收到来自其他主机的请求或回答，对应的ARP结点早已存在，但硬件地址发生了变化，那么ARP结点将被更新。这种情况是这样发生的：其他主机以不同的硬件地址重新启动，而本机的对应ARP结点还未失效。这样，根据机器重新启动时主动发送动态联编信息，可以使主机不至于因其他主机重新启动后导致的ARP结点失效而不能通信。
- 5) 主机可以被配置为代理ARP服务器。这种情况下，主机可以代其他主机响应ARP请求，在回答中提供其他主机的硬件地址。代理ARP回答中对应目的硬件地址的主机必须能够把IP数据报转发至ARP请求中指定的目的主机。卷1的4.6节讨论了代理ARP。

一个Net/3系统可以配置成代理ARP服务器。这些ARP结点可以通过arp命令添加，该命令中指定IP地址、硬件地址并使用关键词pub。我们将在图21-20中看到该实现，并在21-12节讨论其实现细节。

将in_arpinput的分析分为四部分，图21-17显示了第一部分。

358-375 ether_arp结构的长度由调用者(arp_intr函数)验证，所以ea指针指向接收到的分组。ARP操作码(请求或回答)被拷贝至op字段，但具体值要到后面来验证。发送方和目的方的IP地址拷贝到isaddr和itaddr。

1. 查找匹配的接口和IP地址

376-382 搜索本机的Internet地址链表(in_ifaddr结构的链表，图6-5)。要记住一个接口可以有多个IP地址。收到的数据报中有指向接收接口ifnet结构的指针(在mbuf数据报的首部)，for循环只考虑与接收接口相关的IP地址。如果查询到有IP地址等于目的方IP地址或发送方IP地址，则退出循环。

383-384 如果循环退出时，变量maybe_ia的值为0，说明已经搜索了配置的IP地址整个链表而没有找到相关项。函数跳至out(图21-19)，丢弃mbuf，并返回。这种情况只发生在收到ARP请求的接口虽然已初始化但还没有分配IP地址时。

385 如果退出循环时，maybe_ia值不为0，即找到了一个接收端接口，但没有一个IP地址与目的方IP地址或发送方IP地址匹配，则myaddr的值设为该接口的最后一个IP地址；否则（正常情况），myaddr包含与目的方或发送方的IP地址匹配的本地IP地址。

```

358 static void                                     if_ether.c
359 in_arpinput(m)
360 struct mbuf *m;
361 {
362     struct ether_arp *ea;
363     struct arpcom *ac = (struct arpcom *) m->m_pkthdr.rcvif;
364     struct ether_header *eh;
365     struct llinf_arp *la = 0;
366     struct rtentry *rt;
367     struct in_ifaddr *ia, *maybe_ia = 0;
368     struct sockaddr_dl *sdl;
369     struct sockaddr sa;
370     struct in_addr isaddr, itaddr, myaddr;
371     int op;

372     ea = mtod(m, struct ether_arp *);
373     op = ntohs(ea->arp_op);
374     bcopy((caddr_t) ea->arp_spa, (caddr_t) &isaddr, sizeof(isaddr));
375     bcopy((caddr_t) ea->arp_tpa, (caddr_t) &itaddr, sizeof(itaddr));

376     for (ia = in_ifaddr; ia; ia = ia->ia_next)
377         if (ia->ia_ifp == &ac->ac_if) {
378             maybe_ia = ia;
379             if ((itaddr.s_addr == ia->ia_addr.sin_addr.s_addr) ||
380                 (isaddr.s_addr == ia->ia_addr.sin_addr.s_addr))
381                 break;
382         }
383     if (maybe_ia == 0)
384         goto out;
385     myaddr = ia ? ia->ia_addr.sin_addr : maybe_ia->ia_addr.sin_addr;

```

if_ether.c

图21-17 in_arpinput 函数：查找匹配接口

图21-18显示了in_arpinput函数的第二部分，执行分组的验证。

```

386     if (!bcmp((caddr_t) ea->arp_sha, (caddr_t) ac->ac_enaddr,
387               sizeof(ea->arp_sha)))
388         goto out; /* it's from me, ignore it. */
389     if (!bcmp((caddr_t) ea->arp_sha, (caddr_t) etherbroadcastaddr,
390               sizeof(ea->arp_sha))) {
391         log(LOG_ERR,
392             "arp: ether address is broadcast for IP address %x!\n",
393             ntohl(isaddr.s_addr));
394         goto out;
395     }
396     if (isaddr.s_addr == myaddr.s_addr) {
397         log(LOG_ERR,
398             "duplicate IP address %x!! sent from ethernet address: %s!\n",
399             ntohl(isaddr.s_addr), ether_sprintf(ea->arp_sha));
400         itaddr = myaddr;
401         goto reply;
402     }

```

if_ether.c

图21-18 in_arpinput 函数：验证接收到的分组

2. 验证发送方的硬件地址

386-388 如果发送方的硬件地址等于本机接口的硬件地址，那是因为收到了本机发出的请求，忽略该分组。

389-395 如果发送方的硬件地址等于以太网的广播地址，说明出了差错。记录该差错，并丢弃该分组。

3. 检查发送方 IP 地址

396-402 如果发送方的 IP 地址等于 myaddr，说明发送方和本机正在使用同一个 IP 地址。这也是一个差错——要么是发送方，要么是本机系统配置出了差错。记录该差错，在将目的 IP 地址设为 myaddr 后，程序转至 reply(图21-19)。注意该 ARP 分组本来要送往以太网中其他主机的——该分组本来不是要送给本机的。但是，如果这种形式的 IP 地址欺骗被检测到，应记录差错，并产生回答。

图21-19显示了 in_arpinput 函数的第三部分。

```

403     la = arplookup(isaddr.s_addr, itaddr.s_addr == myaddr.s_addr, 0);
404     if (la && (rt = la->la_rt) && (sdl = SDL(rt->rt_gateway))) {
405         if (sdl->sdl_alen &&
406             bcmp((caddr_t) ea->arp_sha, LLADDR(sdl), sdl->sdl_alen))
407             log(LOG_INFO, "arp info overwritten for %x by %s\n",
408                 isaddr.s_addr, ether_sprintf(ea->arp_sha));
409         bcopy((caddr_t) ea->arp_sha, LLADDR(sdl),
410              sdl->sdl_alen = sizeof(ea->arp_sha));
411         if (rt->rt_expire)
412             rt->rt_expire = time.tv_sec + arpt_keep;
413         rt->rt_flags &= ~RTF_REJECT;
414         la->la_asked = 0;
415         if (la->la_hold) {
416             (*ac->ac_if.if_output) (&ac->ac_if, la->la_hold,
417                                     rt_key(rt), rt);
418             la->la_hold = 0;
419         }
420     }

421     reply:
422     if (op != ARPOP_REQUEST) {
423         out:
424         m_freem(m);
425         return;
426     }

```

if_ether.c

图21-19 in_arpinput 函数：创建新的 ARP 结点或更新已有的 ARP 结点

4. 在路由表中搜索与发送方 IP 地址匹配的结点

403 arplookup 在 ARP 高速缓存中查找符合发送方的 IP 地址(isaddr)。当 ARP 分组中的目的 IP 地址等于本机 IP 时，如果要创建新的 ARP 结点，那么第二个参数是 1，如果不需要创建新的 ARP 结点，那么第二个参数是 0。如果本机就是目的主机，总是要创建 ARP 结点的，除非一个查找其他主机的广播分组，这种情况下只是在已有的 ARP 结点中查询。正如前面提到的，如果主机收到一个对应它自己的 ARP 请求，则说明以太网中有其他主机将要与它通信，所以应该创建一个对应该主机的 ARP 结点。

第三个参数是 0，意味着不去查找代理 ARP 结点(后面要证明)。返回值是指向 llinfo_

arp结构的指针；如果查不到或没有创建，返回值就是空。

5. 更新已有结点或填充新的结点

404 只有当以下三个条件为真时 if 语句才执行：

- 1) 找到一个已有的ARP结点或成功创建一个新的ARP结点(即la非空)；
- 2) ARP结点指向一个路由表结点(rt)；
- 3) 路由表结点的re_gateway字段指向一个sockaddr_dl结构。

对于每一个目的并非本机的广播 ARP 请求，如果发送方的 IP 地址不在路由表，则第一个条件为假。

6. 检查发送方的硬件地址是否已改变

405-408 如果链路层地址长度(sdl_alen)非0，说明引用的路由表结点是现存的而非新创建的，则比较链路层地址和发送方的硬件地址。如果不同，则说明发送方的硬件地址已经改变，这是因为发送方以不同的以太网地址重新启动了系统，而本机的 ARP 结点还未超时。这种情况虽然很少出现，但也必须考虑到。记录差错信息后，程序继续往下执行，更新 ARP 结点的硬件地址。

在这个记录报文中，发送方的 IP 地址必须转换为主机字节序，这是一个错误。

7. 记录发送方硬件地址

409-410 将发送方的硬件地址写入路由表结点中 rt_gateway 成员指向的 sockaddr_dl 结构。sockaddr_dl 结构的链路层地址长度(sdl_alen)也被设为6。该赋值对于最近创建的ARP结点是需要的(习题21-3)。

8. 更新最近解析的ARP结点

411-412 在解析了发送方的硬件地址后，执行以下步骤。如果时限是非零的，则将被复位成20分钟(arpt_keep)。arp命令可以创建永久的ARP结点，即该结点永远不会超时。这些ARP结点的时限值置为0。在图21-24中我们将看到，在发送ARP请求(非永久性ARP结点)时，时限被设为本地时间，它是非0的。

413-414 清除RTF_REJECT标志，la_asked计数器设为0。我们将看到，在arpresolve中使用最后两个步骤是为了防止ARP洪泛。

415-420 如果ARP中保持有正在等待ARP解析该目的方硬件地址的mbuf，那么将mbuf送至接口输出函数(如图21-1所示)。由于该mbuf是由ARP保持的，即目的地址肯定是在以太网上，所以接口输出函数应该是ether_outout。该函数也调用arpresolve，但这时硬件地址已被填充，所以允许mbuf加入实际的设备输出队列。

9. 如果是ARP回答分组，则返回

421-426 如果该ARP操作不是请求，那么丢弃接收到的分组，并返回。

in_arpinput的剩下部分如图21-20所示，产生一个对应于ARP请求的回答。只有当以下两种情况时才会产生ARP回答：

- 1) 本机就是该请求所要查找的目的主机；
- 2) 本机是该请求所要查找的目的主机的ARP代理服务器。

函数执行到这个时刻，已经接收了ARP请求，但ARP请求是广播发送的，所以目的主机可能是以太网上的任何主机。

10. 本机就是所要查找的目的主机

427-432 如果目的IP地址等于myaddr，那么本机就是所要查找的目的主机。将发送方硬件地址拷贝到目的硬件地址字段（发送方现在变成了目的主机），arpcom结构中的接口以太网地址拷贝到源硬件地址字段。ARP回答中的其余部分在else语句后处理。

```

427     if (itaddr.s_addr == myaddr.s_addr) {                                     if_ether.c
428         /* I am the target */
429         bcopy((caddr_t) ea->arp_sha, (caddr_t) ea->arp_tha,
430             sizeof(ea->arp_sha));
431         bcopy((caddr_t) ac->ac_enaddr, (caddr_t) ea->arp_sha,
432             sizeof(ea->arp_sha));
433     } else {
434         la = arpllookup(itaddr.s_addr, 0, SIN_PROXY);
435         if (la == NULL)
436             goto out;
437         rt = la->la_rt;
438         bcopy((caddr_t) ea->arp_sha, (caddr_t) ea->arp_tha,
439             sizeof(ea->arp_sha));
440         sdl = SDL(rt->rt_gateway);
441         bcopy(LLADDR(sdl), (caddr_t) ea->arp_sha, sizeof(ea->arp_sha));
442     }

443     bcopy((caddr_t) ea->arp_spa, (caddr_t) ea->arp_tpa, sizeof(ea->arp_spa));
444     bcopy((caddr_t) & itaddr, (caddr_t) ea->arp_spa, sizeof(ea->arp_spa));
445     ea->arp_op = htons(ARPOP_REPLY);
446     ea->arp_pro = htons(ETHERTYPE_IP); /* let's be sure! */
447     eh = (struct ether_header *) sa.sa_data;
448     bcopy((caddr_t) ea->arp_tha, (caddr_t) eh->ether_dhost,
449         sizeof(eh->ether_dhost));
450     eh->ether_type = ETHERTYPE_ARP;
451     sa.sa_family = AF_UNSPEC;
452     sa.sa_len = sizeof(sa);
453     (*ac->ac_if.if_output) (&ac->ac_if, m, &sa, (struct rentry *) 0);
454     return;
455 }

```

图21-20 in_arpinput函数：形成ARP回答，并发送出去

11. 检测本机是否为目的主机的ARP代理服务器

433-437 即使本机不是所要查找的目的主机，也可能被配置为目的主机的ARP代理服务器。再次调用arpllookup函数，将第二个参数设为0，第三个参数设为SIN_PROXY，这将在路由表中查找SIN_PROXY标志为1的结点。如果查找不到（这是通常情况，本机收到了以太网上其他ARP请求的拷贝），out处的代码将丢弃mbuf，并返回。

12. 产生代理回答

437-442 处理代理ARP请求时，发送方的硬件地址变成目的硬件地址，ARP结点中的以太网地址拷贝到发送方硬件地址。该ARP结点中的硬件地址可以是以太网中任一台主机的硬件地址，只要它可以向目的主机转发IP数据报。通常，提供代理ARP服务的主机会填入自己的硬件地址，当然这不是要求的。代理ARP结点是由系统管理员用arp命令带关键字pub创建的，标明目的IP地址（这是路由表项的关键值）和在ARP回答中返回的以太网地址。

13. 完成构造ARP回答分组

443-444 继续完成ARP回答分组的构建。发送方和目标的硬件地址已经填充好了，现在交换发送方和目标的IP地址。目的IP地址在itaddr中，如果发现以太网中有其他主机使用同一

IP地址，则该值已经被填充了(见图21-18)。

445-446 ARP操作码字段设为ARPOP_REPLY，协议地址类型设为ETHERTYPE_IP。旁边加了注释“你需要确定”，是因为当协议地址类型为ETHERTYPE_IPTRAILERS时arpintr也会调用该函数，但现在跟踪封装(trailer encapsulation)已不再使用了。

14. 用以太网帧首部填充sockaddr

447-452 sockaddr结构用14字节的以太网帧首部填充，如图21-12所示。目的硬件地址变成了以太网目的地址。

453-455 将ARP回答传送到接口输出函数，并返回。

21.9 ARP定时器函数

ARP结点一般是动态的——需要时创建，超时时自动删除。也允许管理员创建永久性结点，前面我们讨论的代理结点就是永久性的。回忆一下图21-1和图21-10中最后的#define语句，路由度量结构中的rmx_expire成员就是用作ARP定时器的。

21.9.1 arptimer函数

如图21-21所示，该函数每5分钟被调用一次。它查看所有ARP结点是否超时。

```

74 static void
75 arptimer(ignored_arg)
76 void *ignored_arg;
77 {
78     int s = splnet();
79     struct llinfo_arp *la = llinfo_arp.la_next;
80     timeout(arptimer, (caddr_t) 0, arpt_prune * hz);
81     while (la != &llinfo_arp) {
82         struct rtentry *rt = la->la_rt;
83         la = la->la_next;
84         if (rt->rt_expire && rt->rt_expire <= time.tv_sec)
85             arptfree(la->la_prev); /* timer has expired, clear */
86     }
87     splx(s);
88 }

```

if_ether.c

if_ether.c

图21-21 arptimer 函数：每5分钟查看所有ARP定时器

1. 设置下一个时限

80 arp_rtrequest函数使arptimer函数第一次被调用，随后 arptimer每隔5分钟(arpt_prune)使自己被调用一次。

2. 查看所有ARP结点

81-86 查看ARP结点链表中的每一个结点。如果定时器值是非零的(不是一个永久结点)，而且时间已经超时，那么arptfree就删除该结点。如果rt_expire是非零的，它的值是从结点超时起到现在的秒数。

21.9.2 arptfree函数

如图21-22所示，arptfree函数由arptimer函数调用，用于从链接llinfo_dl表项的

列表中删除一个超时的 ARP 结点。

1. 使正在使用的结点无效(不删除)

467-473 如果路由表引用计数器值大于0，而且 `rt_gateway` 成员指向一个 `sockaddr_dl` 结构，则 `arptfree` 执行以下步骤：

- 1) 将链路层地址长度设为0；
- 2) 将 `la_asked` 计数器值设为0；
- 3) 清除 `RTF_REJECT` 标志。

随后函数返回。因为路由表引用计数器值非零，所以该路由结点不能删除。但是将 `sdl_alen` 值设为0，该结点也就无效了。下次要使用该结点时，还将产生一个 ARP 请求。

```

459 static void                                     if_ether.c
460 arptfree(la)
461 struct llinfo_arp *la;
462 {
463     struct rtentry *rt = la->la_rt;
464     struct sockaddr_dl *sdl;
465     if (rt == 0)
466         panic("arptfree");
467     if (rt->rt_refcnt > 0 && (sdl = SDL(rt->rt_gateway)) &&
468         sdl->sdl_family == AF_LINK) {
469         sdl->sdl_alen = 0;
470         la->la_asked = 0;
471         rt->rt_flags &= ~RTF_REJECT;
472         return;
473     }
474     rtrequest(RTM_DELETE, rt_key(rt), (struct sockaddr *) 0, rt_mask(rt),
475             0, (struct rtentry **) 0);
476 }

```

图21-22 `arptfree` 函数：删除或使一个 ARP 结点无效

2. 删除没有被引用的结点

474-475 `rtrequest` 删除路由结点，在 21.13 节中，我们将看到它调用了 `arp_rtrequest`。`arp_rtrequest` 函数释放所有该 ARP 结点保持的 mbuf (由 `la_hold` 指针所指向)，并删除相应的 `llinfo_arp` 结点。

21.10 `arpresolve` 函数

在图4-16中，`ether_output` 函数调用 `arpresolve` 函数以获得对应某个 IP 地址的以太网地址。如果已知该以太网地址，则 `arpresolve` 返回值为1，允许将待发 IP 数据报挂在接口输出队列上。如果不知道该以太网地址，则 `arpresolve` 返回值为0，`arpresolve` 函数利用 `llinfo_arp` 结构的 `la_hold` 成员指针“保持(held)”待发 IP 数据报，并发送一个 ARP 请求。收到 ARP 应答后，再将保持的 IP 数据报发送出去。

`arpresolve` 应避免 ARP 洪泛，也就是说，它不应在尚未收到 ARP 回答时高速重复发送 ARP 请求。出现这种情况主要有两个原因，第一，有多个 IP 数据报要发往同一个尚未解析硬件地址的主机；第二，一个 IP 数据报的每个分片都会作为独立分组调用 `ether_output`。11.9 节讨论了一个由分片引起的 ARP 洪泛的例子及相关的问题。图 21-23 显示了 `arpresolve` 的前半部分。

252-261 `dst`是一个指向`sockaddr_in`的指针，它包含目的IP地址和对应的以太网地址（一个6字节的数组）。

```

252 int
253 arpresolve(ac, rt, m, dst, desten)
254 struct arpcom *ac;
255 struct rtentry *rt;
256 struct mbuf *m;
257 struct sockaddr *dst;
258 u_char *desten;
259 {
260     struct llinfo_arp *la;
261     struct sockaddr_dl *sdl;
262     if (m->m_flags & M_BCAST) { /* broadcast */
263         bcopy((caddr_t) etherbroadcastaddr, (caddr_t) desten,
264             sizeof(etherbroadcastaddr));
265         return (1);
266     }
267     if (m->m_flags & M_MCAST) { /* multicast */
268         ETHER_MAP_IP_MULTICAST(&SIN(dst)->sin_addr, desten);
269         return (1);
270     }
271     if (rt)
272         la = (struct llinfo_arp *) rt->rt_llinfo;
273     else {
274         if (la = arplookup(SIN(dst)->sin_addr.s_addr, 1, 0))
275             rt = la->la_rt;
276     }
277     if (la == 0 || rt == 0) {
278         log(LOG_DEBUG, "arpresolve: can't allocate llinfo");
279         m_freem(m);
280         return (0);
281     }

```

图21-23 `arpresolve` 函数：查找所需的ARP结点

1. 处理广播和多播地址

262-270 如果`mbuf`的`M_BCAST`标志置位，则用以太网广播地址填充目的硬件地址字段，函数返回1。如果`M_MCAST`标志置位，则宏`ETHER_MAP_IP_MULTICAST`(图12-6)将D类地址映射为相应的以太网地址。

2. 得到指向`llinfo_arp`结构的指针

271-276 目的地址是单播地址。如果调用者传输了一个指向路由表结点的指针，则将`la`设置为相应的`llinfo_arp`结构。否则，`arplookup`根据给定IP的地址搜索路由表。第二个参数是1，告诉`arplookup`如果搜索不到相应的ARP结点就创建一个新的；第三个参数是0，即意味着不去查找代理ARP结点。

277-281 如果`rt`或`la`中有一个是空指针，说明刚才请求分配内存时失败，因为即使不存在已有结点，`arplookup`也已经创建了一个，`rt`和`la`都不应是空值。记录一个差错报文，释放分组，函数返回0。

图21-24显示了`arpresolve`的后半部分。它检查ARP结点是否有效，如无效，则发送一个ARP请求。

```

282     sdl = SDL(rt->rt_gateway);                                if_ether.c
283     /*
284     * Check the address family and length is valid, the address
285     * is resolved; otherwise, try to resolve.
286     */
287     if ((rt->rt_expire == 0 || rt->rt_expire > time.tv_sec) &&
288         sdl->sdl_family == AF_LINK && sdl->sdl_alen != 0) {
289         bcopy(LLADDR(sdl), desten, sdl->sdl_alen);
290         return 1;
291     }
292     /*
293     * There is an arptab entry, but no ethernet address
294     * response yet. Replace the held mbuf with this
295     * latest one.
296     */
297     if (la->la_hold)
298         m_freem(la->la_hold);
299     la->la_hold = m;

300     if (rt->rt_expire) {
301         rt->rt_flags &= ~RTF_REJECT;
302         if (la->la_asked == 0 || rt->rt_expire != time.tv_sec) {
303             rt->rt_expire = time.tv_sec;
304             if (la->la_asked++ < arp_maxtries)
305                 arpwhoas(ac, &(SIN(dst)->sin_addr));
306             else {
307                 rt->rt_flags |= RTF_REJECT;
308                 rt->rt_expire += arpt_down;
309                 la->la_asked = 0;
310             }
311         }
312     }
313     return (0);
314 }

```

图21-24 arpresolve 函数：检查ARP结点是否有效，如无效，则发送一个ARP请求

3. 检查ARP结点的有效性

282-291 即使找到了一个ARP结点，还需检查其有效性。如以下条件成立，则ARP结点是有效的：

- 1) 结点是永久有效的(时限值为0)，或尚未超时；
- 2) 由rt_gateway指向的插口地址结构的sdl_family字段为AF_LINK；
- 3) 链路层地址长度值(sdl_alen)不等于0。

arpfree使一个仍被引用的ARP结点失效的方法是将sdl_alen值置0。如果结点是有效的，则将sockaddr_dl中的以太网地址拷贝到desten，函数返回1。

4. 只保持最近的IP数据报

292-299 此时，已经有了ARP结点，但它没有一个有效的以太网地址，因此，必须发送一个ARP请求。将la_hold指针指向mbuf，同时也就释放了刚才la_hold所指的内容。这意味着，在发送ARP请求到收到ARP回答之间，如果有多个发往同一目的地的IP数据报要发送，只有最近的一个IP数据报才被la_hold保留，之前的全部丢弃。NFS就是这样的一个例子，如果NFS要传送一个8500字节的IP数据报，需要将其分割成6个分片。如果每个分片都在发送ARP请求到收到ARP回答之间由ip_output送往ether_output，那么前5个分片将被丢弃，

当收到ARP回答时，只有最后一个分片被保留了下来。这会使NFS超时，并重发这6个分片。

5. 发送ARP请求，但避免ARP洪泛

300-314 RFC 1122要求ARP避免在收到ARP回答之前以过高的速度对一个以太网地址重发ARP请求。Net/3采用以下方法来避免ARP洪泛：

- Net/3不在同一秒钟内发送多个对应同一目的地的ARP请求；
- 如果在连续5个ARP请求(也就是5秒钟)后还没有收到回答，路由结点的RTF_REJECT标志置1，时限设为往后的20秒。这会使ether_output在20秒内拒绝发往该目的地址的IP数据报，并返回EHOSTDOWN或EHOSTUNREACH(如图4-15所示)。
- 20秒钟后，arpresolve会继续发送该目的主机的ARP请求。

如果时限值不等于0(非永久性结点)，则清除RTF_REJECT标志，该标志是在早些时候为避免ARP洪泛而设置的。计数器la_asked记录的是连续发往该目的地址的ARP请求数。如果计数器值为0或时限值不等于当前时钟(只需看一下当前时钟的秒钟部分)，那么需要再发送一个ARP请求。这就避免了在同一秒钟内发送多个ARP请求。然后将时限值设为当前时钟的秒钟部分(也就是微秒部分，time_tv_usec被忽略)。

将la_asked所含计数器值与限定值5(arp_maxtries)比较，然后加1。如果小于5，则arpwhoas发送ARP请求；如果等于5，则ARP已经达到了限定值：将RTF_REJECT标志置1，时限值置为往后的20秒钟，la_asked计数器值复位为0。

图21-25显示了一个例子，进一步解释了arpresolve和ether_output为了避免ARP洪泛所采用的算法。

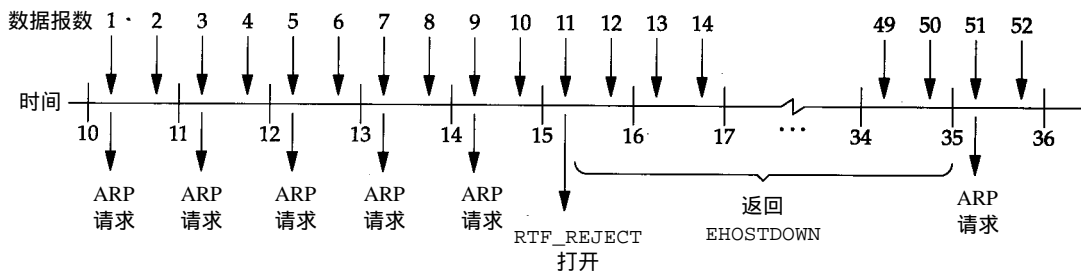


图21-25 避免ARP洪泛所采用的算法

图中总共显示了26秒的时间，从10到36。我们假定有一个进程每隔0.5秒发送一个IP数据报，也就是说，一秒钟内有两个数据报等待发送。数据报依次被标号为1~52。我们还假定目的主机已经关闭，所以收不到ARP回答。ARP将采取以下行动：

- 假定当进程写数据报1时la_asked的值为0。la_hold设为指向数据报1，rt_expire值设为当前时钟(10)，la_asked值变为1，发送ARP请求。函数返回0。
- 进程写数据报2时，丢弃数据报1，la_hold指向数据报2。由于rt_expire值等于当前时钟(10)，所以不发送ARP请求，函数返回，返回值为0。
- 进程写数据报3时，丢弃数据报2，la_hold指向数据报3。由于当前时钟(11)不等于rt_expire(10)，所以将rt_expire设为11。la_asked值为1，小于5，所以发送ARP请求，并将la_asked值置为2。
- 进程写数据报4时，丢弃数据报3，la_hold指向数据报4。由于rt_expire值等于当

前时钟(11)，所以无须其他动作，函数返回0。

- 对于数据报5~10，情况都是一样的。在数据报9到达后，发送ARP请求后，la_asked值被设为5；
- 进程写数据报11时，丢弃数据报10，la_hold指向数据报11。当前时钟(15)不等于rt_expire(14)，所以将rt_expire的值设为15。此时la_asked的值不再小于5，ARP避免洪泛的算法开始作用：RTF_REJECT标志位置1，rt_expire的值被设为35(即往后20秒)，la_asked的值设为0，函数返回0。
- 进程写数据报12时，ether_output注意到RTF_REJECT标志位为1，而且当前时钟小于rt_expire(35)，因此，返回EHOSTDOWN给发送者(通常是ip_output)。
- 从数据报13到50，都返回EHOSTDOWN给发送者。
- 当进程写数据报51时，尽管此时的RTF_REJECT标志位仍然为1，但当前时钟的值(35)不再小于rt_expire(35)，因此不会返回出错信息。调用arpresolve，整个过程重新开始，5秒钟内发送5个ARP请求，然后是20秒钟的等待，直到发送者放弃或目的主机响应ARP请求。

21.11 arplookup函数

arplookup函数调用选路函数rtallocl在Internet路由表中查找ARP结点。我们已经看到过3次调用arplookup的情况：

- 1) 在in_arpinput中，在接收到ARP分组后，对应源IP地址查找或创建一个ARP结点。
- 2) 在in_arpinput中，接收到ARP请求后，查看是否存在目的硬件地址的代理ARP结点。
- 3) 在arpresolve中，查找或创建一个对应待发送数据报IP地址的ARP结点。

如果arplookup执行成功，则返回一个指向相应llinfo_arp结构的指针，否则返回一个空指针。

arplookup带有三个参数，第一个参数是目的IP地址；第二个参数是个标志，为真时表示若找不到相应结点就创建一个新的结点；第三个参数也是一个标志，为真时表示查找或创建代理ARP结点。

代理ARP结点通过定义一个不同形式的Internet插口地址结构来处理，即sockaddr_inarp结构，如图21-26所示。该结构只在ARP中使用。

```

111 struct sockaddr_inarp {
112     u_char  sin_len;           /* sizeof(struct sockaddr_inarp) = 16 */
113     u_char  sin_family;       /* AF_INET */
114     u_short sin_port;
115     struct in_addr sin_addr;   /* IP address */
116     struct in_addr sin_srcaddr; /* not used */
117     u_short sin_tos;          /* not used */
118     u_short sin_other;        /* 0 or SIN_PROXY */
119 };

```

if_ether.h

图21-26 sockaddr_inarp 结构

111-119 前面8个字节与sockaddr_in结构相同，sin_family被设为AF_INET。最后8

个字节有所不同：sin_srcaddr、sin_tos和sin_other成员。当结点作为代理结点时，只用到sin_other成员，并将其设为SIN_PROXY(1)。

图21-27显示了arplookup函数。

```

480 static struct llinfo_arp *                               if_ether.c
481 arplookup(addr, create, proxy)
482 u_long  addr;
483 int      create, proxy;
484 {
485     struct rtentry *rt;
486     static struct sockaddr_inarp sin =
487     {sizeof(sin), AF_INET};

488     sin.sin_addr.s_addr = addr;
489     sin.sin_other = proxy ? SIN_PROXY : 0;
490     rt = rtalloc1((struct sockaddr *) &sin, create);
491     if (rt == 0)
492         return (0);
493     rt->rt_refcnt--;
494     if ((rt->rt_flags & RTF_GATEWAY) || (rt->rt_flags & RTF_LLINFO) == 0 ||
495         rt->rt_gateway->sa_family != AF_LINK) {
496         if (create)
497             log(LOG_DEBUG, "arptnew failed on %x\n", ntohl(addr));
498         return (0);
499     }
500     return ((struct llinfo_arp *) rt->rt_llinfo);
501 }

```

图21-27 arplookup 函数：在路由表中查找ARP结点

1. 初始化sockaddr_inarp结构，准备查找

480-489 sin_addr成员设为将要查找的 IP地址。如果 proxy参数值不为 0，则 sin_other成员设为SIN_PROXY；否则设为0。

2. 路由表中查找结点

490-492 rtalloc1在Internet路由表中查找IP地址，如果create参数值不为0，就创建一个新的结点。如果找不到结点，则函数返回值为 0(空指针)。

3. 减少路由表结点的引用计数值

493 如果找到了结点，则减少路由表结点的引用计数。因为，此时ARP不再被认为像运输层一样“持有”路由表结点，因此，路由表查找时对rt_refcnt计数的递增，应在这里由ARP取消。

494-499 如果将标志RTF_GATEWAY置位，或者标志RTF_LLINFO没有置位，或者由rt_gateway指向的插口地址结构的地址族字段值不是AF_LINK，说明出了某些差错，返回一个空指针。如果结点是这样创建的，应创建一个记录报文。

记录报文中对arptnew的注释是针对老版本Net/2中创建ARP结点的。

如果rtalloc1由于匹配结点的RTF_CLONING标志置位而创建一个新的结点，那么函数arp_rtrequest (21.13节)也要被rtrequest调用。

21.12 代理ARP

Net/3支持代理ARP，有两种不同类型的代理ARP结点，可以通过arp命令及pub选项将

它们加入到路由表中。添加代理 ARP选项会使 `arp_rtrequest` 主动发送动态联编信息 (如图 21-28 所示), 因为在创建结点时 `RTF_ANNOUNCE` 标志位被置 1。

代理 ARP 结点的第一种类型: 它允许将网络内的某一主机的 IP 地址填入到 ARP 高速缓存内。硬件地址可以设为任意值。这种结点加入到路由表中时使用了直接的掩码 `0xffffffff`。加掩码的目的是即使插口地址的 `SIN_PROXY` 标志位为 1, 在调用图 21-27 中的 `rtalloc1` 时能与该结点匹配。于是在调用图 21-20 中的 `arplookup` 时也能与该结点匹配, 目的地址的 `SIN_PROXY` 置位。

如果本网中的主机 H1 不能实现 ARP, 那么可以使用这种类型的代理 ARP 结点。作为代理的主机代替 H1 回答所有的 ARP 请求, 同时提供创建代理 ARP 结点时设定的硬件地址 (比如可以是 H1 的以太网地址)。这种类型的结点可以通过 `arp -a` 命令查看, 它带有 “published” 符号。

第二种类型的代理 ARP 结点用于已经存有路由表结点的主机。内核为该目的地址创建另外一个路由表结点, 在这个新的结点中含有链路层的信息 (如以太网地址)。该新结点中 `sockaddr_inar` 结构 (图 21-26) 的 `sin_other` 成员的 `SIN_PROXY` 标志置位。回想一下, 搜索路由表时是比较 12 字节的 Internet 插口地址 (图 18-39)。只有当该结构的最后 8 字节非零时, 才会用到 `SIX_PROXY` 标志位。当 `arplookup` 指定送往 `rtalloc1` 的结构中的 `sin_other` 成员中 `SIN_PROXY` 的值时, 只有路由表中那些匹配的结点的 `SIN_PROXY` 标志置位。

这种类型的代理 ARP 结点通常指明了作为代理 ARP 服务器的以太网地址。如果某代理 ARP 结点是为主机 HD 创建的, 一般有以下步骤:

- 1) 代理服务器收到来自主机 HS 的查找 HD 硬件地址的广播 ARP 请求, 主机 HS 认为 HD 在本地网上;
- 2) 代理服务器回答请求, 并提供本机的以太网地址;
- 3) HS 将发往 HD 的数据报发送给代理服务器;
- 4) 收到发往 HD 的数据报后, 代理服务器利用路由表中关于 HD 的信息将数据报转发给 HD。

路由器 `netb` 使用这种类型的代理 ARP 结点, 见卷 1 4.6 节中的例子。可以通过命令 `arp -a` 来查看这些带有 “published (proxy only)” 的结点。

21.13 `arp_rtrequest` 函数

图 21-3 简要显示了 ARP 函数和选路函数之间的关系。在 ARP 中, 我们将调用两个路由表函数:

- 1) `arplookup` 调用 `rtalloc1` 查找 ARP 结点, 如果找不到匹配结点, 则创建一个新的 ARP 结点。

如果在路由表中找到了匹配结点, 且该结点的 `RTF_CLONING` 标志位没有置位 (即该结点就是目的主机的结点), 则返回该结点。如果 `RTF_CLONING` 标志位被置位, `rtalloc1` 以 `RTM_RESOLVE` 命令为参数调用 `rtrequest`。图 18-2 中的 140.252.13.33 和 140.252.13.34 结点就是这么创建的, 它们是从 140.252.13.32 的结点复制而来的。

- 2) `arptfree` 以 `RTM_DELETE` 命令为参数调用 `rtrequest`, 删除对应 ARP 结点的路由表结点。

此外, `arp` 命令通过发送和接收路由插口上的路由报文来操纵 ARP 高速缓存。 `arp` 以命令

RTM_RESOLVE、RTM_DELETE和RT_GET为参数发布路由信息。前两个参数用于调用rtrequest，第三个参数用于调用rtallocl。

最后，当以太网设备驱动程序获得了赋予该接口的IP地址后，rtinit增加一个网络路由。于是rtrequest函数被调用，参数是RTM_ADD，标志位是RTF_UP和RTF_CLONING。图18-2中140 252 13 32结点就是这么创建的。

在第19章中我们讲过，每一个ifa_addr结构都有一个指向函数(ifa_rtrequest成员)的指针，该函数在创建或删除一个路由表结点时被自动调用。在图6-17中，对于所有以太网设备，in_ifinit将该指针指向arp_rtrequest函数。因此，当调用路由函数为ARP创建或删除路由表结点时，总会调用arp_rtrequest。当任意路由表函数被调用时，arp_rtrequest函数的作用是做各种初始化或退出处理所需的工作。例如：当创建新的ARP结点时，arp_rtrequest内要为llinfo_arp结构分配内存。同样，当路由函数处理完一个RTM_DELETE命令后，arp_rtrequest的工作是删除llinfo_arp结构。

图21-28显示了arp_rtrequest函数的第一部分。

if_ether.c

```

92 void
93 arp_rtrequest(req, rt, sa)
94 int req;
95 struct rtentry *rt;
96 struct sockaddr *sa;
97 {
98     struct sockaddr *gate = rt->rt_gateway;
99     struct llinfo_arp *la = (struct llinfo_arp *) rt->rt_llinfo;
100     static struct sockaddr_dl null_sdl =
101         {sizeof(null_sdl), AF_LINK};

102     if (!arpinit_done) {
103         arpinit_done = 1;
104         timeout(arptimer, (caddr_t) 0, hz);
105     }
106     if (rt->rt_flags & RTF_GATEWAY)
107         return;
108     switch (req) {

109     case RTM_ADD:
110         /*
111          * XXX: If this is a manually added route to interface
112          * such as older version of routed or gated might provide,
113          * restore cloning bit.
114          */
115         if ((rt->rt_flags & RTF_HOST) == 0 &&
116             SIN(rt_mask(rt))->sin_addr.s_addr != 0xffffffff)
117             rt->rt_flags |= RTF_CLONING;
118         if (rt->rt_flags & RTF_CLONING) {
119             /*
120              * Case 1: This route should come from a route to iface.
121              */
122             rt_setgate(rt, rt_key(rt),
123                 (struct sockaddr *) &null_sdl);
124             gate = rt->rt_gateway;
125             SDL(gate)->sdl_type = rt->rt_ifp->if_type;
126             SDL(gate)->sdl_index = rt->rt_ifp->if_index;
127             rt->rt_expire = time.tv_sec;

```

图21-28 arp_rtrequest 函数：RTM_ADD 命令

```

128         break;
129     }
130     /* Announce a new entry if requested. */
131     if (rt->rt_flags & RTF_ANNOUNCE)
132         arprequest((struct arpcom *) rt->rt_ifp,
133                   &SIN(rt_key(rt))->sin_addr.s_addr,
134                   &SIN(rt_key(rt))->sin_addr.s_addr,
135                   (u_char *) LLADDR(SDL(gate)));
136     /* FALLTHROUGH */

```

if_ether.c

图21-28 (续)

1. 初始化ARP timeout函数

92-105 第一次调用arp_rtrequest函数时(系统初始化阶段,在对第一个以太网接口赋IP地址时),timeout函数在一个时钟滴答内调用arptimer函数。此后,ARP定时器代码每5分钟运行一次,因为arptimer总是要调用timeout的。

2. 忽略间接路由

106-107 如果将标志RTF_GATEWAY置位,则函数返回。RTF_GATEWAY标志表明该路由表结点是间接的,而所有ARP结点都是直接的。

108 一个带有三种可能的switch语句:RTM_ADD、RTM_RESOLVE和RTM_DELETE(后两种在后面的图中显示)。

3. RTM_ADD命令

109 RTM_ADD命令出现在以下两种情况中:执行arp命令手工创建ARP结点或者rtinit函数对以太网接口赋IP地址(图21-3)。

4. 向后兼容

110-117 若标志RTF_HOST没有置位,说明该路由表结点与一个掩码相关(也就是说网络路由,而非主机路由)。如果掩码不是全1,那么该结点确实是某一接口的路由,因此,将标志RTF_CLONING置位。如注释中所述,这是为了与某些旧版本的路由守护程序兼容。此外,/etc/netstart中的命令:

```
route add -net 224.0.0.0 -interface bsdi
```

为图18-2所示网络创建带有RTF_CLONING标志的路由表结点。

5. 初始化到接口的网络路由结点

118-126 若标志RTF_CLONING(in_ifinit为所有以太网接口设置该标志)置位,那么该路由表结点是由rtinit添加的。rt_setgate为sockaddr_dl结构分配空间,该结构由rt_gateway指针所指。与图21-1中140.252.13.32的路由表结点相关的就是该数据链路接口地址结构。sdl_family和sdl_len成员的值是根据静态定义的null_sd而初始化的,sdl_type(可能是IFT_ETHER)和sdl_index成员的值来自接口的ifnet结构。该结构不包含以太网地址,sdl_alen成员的值为0。

127-128 最后将时限值设为当前时间,也就是结点的创建时间,执行break后返回。对于在系统初始化时创建的结点,它们的rmx_expire值为系统启动的时间。注意,图21-1中该路由表结点没有相应的llinfo_arp结构,所以它不会被arptimer处理。但是要用它的sockaddr_dl结构,对于以太网中特定主机的路由结点来说,要复制的是rt_gateway结构,用RTM_RESOLVE命令参数创建路由表结点时,rtrequest复制该结构。此外,

netstat程序将sdl_index的值输出为link#n，见图18-2。

6. 发送免费ARP请求

130-135 若将标志RTF_ANNOUNCE置位，则该结点是由arp命令带pub选项创建的。该选项有两个分支：(1) sockaddr_inarp结构中sin_other成员的SIN_PROXY标志被置位；(2)标志RTF_ANNOUNCE被置位。因为标志RTF_ANNOUNCE被置位，所以arprequest广播免费ARP请求。注意，第二个和第三个参数是相同的，即该ARP请求中，发送方IP地址和目的方IP地址是一样的。

136 继续执行针对RTM_RESOLVE命令的case语句。

图21-29显示了arp_rtrequest函数的第二部分，处理RTM_RESOLVE命令。当rtallocl找到一个RTF_CLONING标志位置位的路由表结点且rtallocl的第二个参数值(arplookup的create参数)不为0时，调用该命令。需要分配一个新的llinfo_arp结构，并将其初始化。

```

137     case RTM_RESOLVE:
138         if (gate->sa_family != AF_LINK ||
139             gate->sa_len < sizeof(null_sdl)) {
140             log(LOG_DEBUG, "arp_rtrequest: bad gateway value");
141             break;
142         }
143         SDL(gate)->sdl_type = rt->rt_ifp->if_type;
144         SDL(gate)->sdl_index = rt->rt_ifp->if_index;
145         if (la != 0)
146             break; /* This happens on a route change */
147         /*
148          * Case 2: This route may come from cloning, or a manual route
149          * add with a LL address.
150          */
151         R_Malloc(la, struct llinfo_arp *, sizeof(*la));
152         rt->rt_llinfo = (caddr_t) la;
153         if (la == 0) {
154             log(LOG_DEBUG, "arp_rtrequest: malloc failed\n");
155             break;
156         }
157         arp_inuse++, arp_allocated++;
158         Bzero(la, sizeof(*la));

159         la->la_rt = rt;
160         rt->rt_flags |= RTF_LLINFO;
161         insque(la, &llinfo_arp);

162         if (SIN(rt_key(rt))->sin_addr.s_addr ==
163             (IA_SIN(rt->rt_ifa))->sin_addr.s_addr) {
164             /*
165              * This test used to be
166              * if (loif.if_flags & IFF_UP)
167              * It allowed local traffic to be forced
168              * through the hardware by configuring the loopback down.
169              * However, it causes problems during network configuration
170              * for boards that can't receive packets they send.
171              * It is now necessary to clear "useloopback" and remove
172              * the route to force traffic out to the hardware.
173              */
174             rt->rt_expire = 0;

```

图21-29 arp_rtrequest 函数：RTM_RESOLVE 命令

```

175         Bcopy(((struct arpcom *) rt->rt_ifp)->ac_enaddr,
176                LLADDR(SDL(gate)), SDL(gate)->sdl_alen = 6);
177         if (useloopback)
178             rt->rt_ifp = &loif;

179     }
180     break;

```

—*if_ether.c*

图21-29 (续)

7. 验证sockaddr_dl结构

137-144 验证`rt_gateway`指针所指的`sockaddr_dl`结构的`sa_family`和`sa_len`成员的值。接口类型(可能是`IFT_ETHER`)和索引值填入新的`sockaddr_dl`结构。

8. 处理路由变化

145-146 正常情况下, 该路由表结点是新创建的, 并没有指向一个`llinfo_arp`结构。如果`la`指针非空, 则在路由已发生了变化时调用`arp_rtrequest`。此时`llinfo_arp`已经分配, 执行`break`, 函数返回。

9. 初始化llinfo_arp结构

147-158 分配一个`llinfo_arp`结构, `rt_llinfo`中存有指向该结构的指针。统计值变量`arp_inuse`和`arp_allocated`各加1, `llinfo_arp`结构置0。将`la_hold`指针置空, `la_asked`值置0。

159-161 将`rt`指针存储于`llinfo_arp`结构中, 置`RTF_LLLINFO`标志位。如图18-2所示, ARP创建的三个结点140.252.13.33、140.252.13.34和140.252.13.35都有`L`标志, 和240.0.0.1一样。`arp`程序只检查该标志(图19-36)。最后`insque`将`llinfo_arp`加入到链接表的首部。

就这样创建了一个ARP结点：`rtrequest`创建路由表结点(经常为以太网克隆一个特定网络的结点), `arp_rtrequest`分配和初始化`llinfo_arp`结构。剩下只需广播一个ARP请求, 在收到回答后填充主机的以太网地址。事件发生的一般次序是：`arpresolve`调用`arplookup`, 于是`arp_rtrequest`被调用(中间可能跟有函数调用, 见图21-3)。当控制返回到`arpresolve`时, 发送ARP广播请求。

10. 处理发给本机的特例情况

162-173 这是4.4BSD新增的测试特例部分(注释是老版本留下的)。它创建了图21-1中最右边的路由表结点, 该结点包含了本机的IP地址(140.252.13.35)。`if`语句检测它是否等于本机IP地址, 如等于, 那么这个刚创建的结点代表的是本机。

11. 将结点置为永久性, 并设置以太网地址

174-176 时限值设为0, 意味着该结点是永久有效的——永远不会超时。从接口的`arpcom`结构中将硬件地址拷贝至`rt_gateway`所指的`sockaddr_dl`结构中。

12. 将接口指针指向环回接口

177-178 若全局变量`usrloopback`值不为0(默认为1), 则将路由表结点内的接口指针指向环回接口。这意味着, 如果有数据报发给自己, 就送往环回接口。在4.4BSD以前的版本中, 可以通过`/etc/netstart`文件中的命令:

```
route add 140.252.13.35 127.0.0.1
```

来建立从本机IP地址到环回接口的路由。4.4BSD仍然支持这种方式, 但已不是必需的了。当

第一次有数据报发给本机 IP 地址时，我们刚才看到的代码会自动创建一个这样的路由。此外，这些代码对于一个接口只会执行一次。一旦路由表结点和永久性 ARP 结点创建好后，它们就不会超时，所以不会再次出现对本机 IP 地址的 RTM_RESOLVE 命令。

arp_rtrequest 函数的最后部分如图 21-30 所示，处理 RTM_DELETE 请求。从图 21-3 中，我们可以看到，该命令是由 arp 命令产生的，用于手工删除一个结点；或者在一个 ARP 结点超时由 arptfree 产生。

```

181     case RTM_DELETE:
182         if (la == 0)
183             break;
184         arp_inuse--;
185         remque(la);
186         rt->rt_llinfo = 0;
187         rt->rt_flags &= ~RTF_LLINFO;
188         if (la->la_hold)
189             m_freem(la->la_hold);
190         Free((caddr_t) la);
191     }
192 }

```

if_ether.c

if_ether.c

图21-30 arp_rtrequest函数：RTM_DELETE 命令

13. 验证la指针

182-183 la 指针应该非空的，也就是说路由表结点必须指向一个 llinfo_arp 结构；否则，执行 break，函数返回。

14. 删除llinfo_arp结构

184-190 统计值变量 arp_inuse 减 1，remque 从链表中删除 llinfo_arp 结构。rt_llinfo 指针置 0，清除 RTF_LLINFO 标志。如果该 ARP 结点保持有 mbuf（即该 ARP 请求未收到回答），则将 mbuf 释放。最后释放 llinfo_arp 结构。

注意，switch 语句中没有包含 default 情况，也没有考虑 RTM_GET 命令。这是因为 arp 程序产生的 RTM_GET 命令全部由 route_output 函数处理，并不调用 rtrequest。此外，见图 21-3，在 RTM_GET 命令产生的对 rtalloc1 调用中，指定第二个参数是 0，所以 rtalloc1 并不调用 rtrequest。

21.14 ARP和多播

如果一个 IP 数据报要采用多播方式发送，ip_output 检测进程是否已将某个特定的接口赋予插口（见图 12-40）。如果已经赋值，则将数据报发往该接口，否则，ip_output 利用路由表选择输出接口（见图 8-24）。因此，对于具有多个多播发送接口的系统来说，IP 路由表应指定每个多播组的默认接口。

在图 18-2 中我们看到，路由表中有一个结点是为网络 224.0.0.0 创建的，该结点具有“flag”标志。所有以 224 开头的多播组都以该结点指定的接口（le0）为默认接口。对于其他的多播组（以 225~239 开头），可以分别创建新的路由表结点，也可以对某个指定多播组创建一个路由表结点。例如，可以为 224.0.11（网络定时协议）创建一个与 224.0.0.0 不同的路由表结点。如果路由表中没有对应某个多播组的结点，同时进程没有用 IP_MULTICAST_IF 插口选项指明接口，那么该组的默认接口成为路由表中默认路由的接口。其实图 18-2 中对应 224.0.0.0 的路由表结

点并不是必要的，因为默认接口就是 `le0`。

如果选定的接口是以太网接口，则调用 `arpresolve` 将多播组地址映射为相应的以太网地址。在图 21-23 中，映射通过调用宏 `ETHER_MAP_IP_MULTICAST` 来完成。该宏所做的就是将该多播组地址的低 23 位与一个常量逻辑或（图 12-6），映射不需要 ARP 请求和回答，也不需要进入 ARP 高速缓存。每次需要映射时，调用该宏。

如果多播组是从另外一个结点复制得来的，那么多播组地址会出现在 ARP 缓存里，如图 21-5 所示。因为这些结点将 `RTF_LLINFO` 标志置位。它们不会有 ARP 请求和回答，所以说不是真正的 ARP 结点。它们也没有相应的链路层地址，宏 `ETHER_MAP_IP_MULTICAST` 就可以完成映射。

这些多播组的 ARP 结点的时效与正常的 ARP 结点不同。在为某个多播组创建一个路由表结点时，如图 18-2 中的 224.0.0.1，`rtrequest` 从被克隆的结点中复制 `rt_metrics` 结构（图 19-9）。图 21-28 中，网络路由结点的 `rmx_expire` 值被设为 `RTM_ADD` 命令执行的时间，也即系统初始化的时间。为 224.0.0.1 设置的结点也设置为同样的时间。

这就意味着在下次 `arptimer` 执行时，对应多播组 224.0.0.1 的 ARP 结点总是超时的。所以，当下一次在路由表中查找时就需重新创建该结点。

21.15 小结

ARP 提供了 IP 地址到硬件地址的映射，本章讲述了如何实现这种映射。

Net/3 实现与以往的 BSD 版本有很大不同。ARP 信息被存放在多个结构里面：路由表、数据链路插口地址结构和 `llinfo_arp` 结构。图 21-1 显示了这些结构之间的关系。

发送一个 ARP 请求是很简单的：正确填充相关字段后，将请求广播发送出去就行了。处理请求就要复杂一些，因为每个主机都收到了广播的 ARP 请求。除了响应请求外，`in_arpinput` 还要检测是否有其他主机正与它使用同一个 IP 地址。因为每一个 ARP 请求中包含发送方的 IP 和硬件地址，所以网络上的所有主机都可以通过它来更新自己的 ARP 结点。

在局域网中，ARP 洪泛将是一个问题，Net/3 是第一个考虑这种问题的 BSD 版本。对于同一个目的地，一秒钟内只可发送一个 ARP 请求，如果连续 5 个请求都没有收到回答，必须暂停 20 秒钟才可再发送去往该目的地的 ARP 请求。

习题

- 21.1 图 21-17 中给局部变量 `ac` 赋值时，做过什么假设？
- 21.2 如果我们先 ping 本地以太网的广播地址，之后执行 `arp -a`，就可以发现几乎所有本地以太网上的其他主机的表项都填入到了 ARP 高速缓存中。这是为什么？
- 21.3 查看代码并解释为什么图 21-19 中需要把 `sdl_alen` 的值赋为 6。
- 21.4 在 Net/2 中有一个独立于路由表而存在的 ARP 表，每次调用 `arpresolve` 时，都要在该 ARP 表中查找。试与 Net/3 的方法比较，哪个更有效？
- 21.5 Net/2 中的 ARP 代码显式地设置 ARP 高速缓存中非完整表项的超时为 3 分钟，非完整表项是指正在等待 ARP 回答的表项。但我们从没有提过 Net/3 如何处理该超时，那么 Net/3 何时才认为非完整表项超时？
- 21.6 当 Net/3 系统作为一个路由器并且导致洪泛的分组来自其他主机时，为避免 ARP 洪

泛要做哪些变动？

- 21.7 图21-1中给出的四个rmx_expire变量的值是什么？代码在何处设置该值？
- 21.8 对广播ARP请求的每个主机，本章中引起要创建一个ARP结点的代码需要做哪些变动？
- 21.9 为了验证图21-25中的例子，作者运行了卷1附录C的sock程序，每隔500 ms向本地以太网上一个不存在的主机发送一个UDP数据报(程序的-p选项改为等待的毫秒数)。但是在返回第一个EHOSTDOWN差错之前，仅无差错地发送了10个UDP数据报，而不是图21-25所示的11个，这是为什么？
- 21.10 修改ARP，使得它在等待ARP回答时持有到目的主机的所有分组，而不是持有最近的一个。如何实现这种改变？是否像每个接口的输出队列一样，需要一个限制？是否需要改变数据结构？