

第13章 IGMP : Internet组管理协议

13.1 引言

IGMP在本地网络上的主机和路由器之间传达组成员信息。路由器定时向“所有主机组”多播IGMP查询。主机多播IGMP报告报文以响应查询。IGMP规范在RFC 1112中。卷1的第13章讨论了IGMP的规范，并给出了一些例子。

从体系结构的观点来看，IGMP是位于IP上面的运输层协议。它有一个协议号(2)，它的报文是由IP数据报运载的(与ICMP一样)。与ICMP一样，进程通常不直接访问IGMP，但进程可以通过IGMP插口发送或接收IGMP报文。这个特性使得能够把多播选路守护程序作为用户级进程实现。

图13-1显示了Net/3中IGMP协议的整体结构。

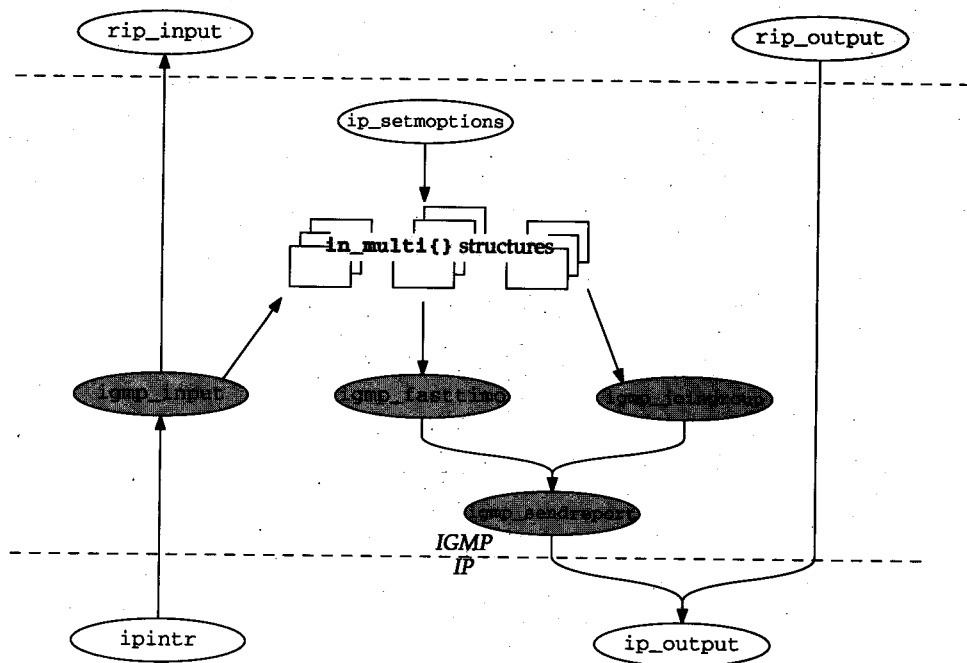


图13-1 IGMP处理概要

IGMP处理的关键是一组在图 13-1中心显示的 `in_multi` 结构。到达的 IGMP 查询使 `igmp_input` 为每个 `in_multi` 结构初始化一个递减定时器。该定时器由 `igmp_fasttimo` 更新，当每个定时器超时，`igmp_fasttimo` 调用 `igmp_sendreport`。

我们在第12章中看到，当创建一个新的 `in_multi` 结构时，`ip_setoptions` 调用 `igmp_joingroup`。`igmp_joingroup` 调用 `igmp_sendreport` 来发布新的组成员信息，使组的定时器能够在短时间内安排第二次通告。`igmp_sendreport` 完成对IGMP报文的格式

化，并把它传给ip_output。
 在图13-1的左边和右边，我们看到一个原始插口可以直接发送和接收 IGMP报文。

13.2 代码介绍

图13-2中列出了实现IGMP协议的4个文件。

文 件	描 述
netinet/igmp.h	IGMP协议定义
netinet/igmp_var.h	IGMP实现定义
netinet/in_var.h	IP多播数据结构
netinet/igmp.c	IGMP协议实现

图13-2 本章讨论的文件

13.2.1 全局变量

本章中介绍的新的全局变量显示在图 13-3中。

变 量	数 据 类 型	描 述
igmp_all_hosts_group	u_long	网络字节序的“所有主机组”地址
igmp_timer_are_running	int	如果所有IGMP定时器都有效，则为真；否则为假
igmp_stat	struct igmpstat	IGMP统计(图13-4)

图13-3 本章介绍的全局变量

13.2.2 统计量

IGMP统计信息是在图13-4的igmpstat变量中维护的。

Igmpstat成员	描 述
igps_rcv_badqueries	作为无效查询接收的报文数
igps_rcv_badreports	作为无效报告接收的报文数
igps_rcv_badsum	接收的报文检验和错误数
igps_rcv_ourreports	作为逻辑组的报告接收的报文数
igps_rcv_queries	作为成员关系查询接收的报文数
igps_rcv_reports	作为成员关系报告接收的报文数
igps_rcv_tooshort	字节数太少的报文数
igps_rcv_total	接收的全部报文数
igps_snd_reports	作为成员关系报告发送的报文数

图13-4 IGMP统计

图13-5是在vangogh.cs.berkeley.edu上执行netstat -p igmp命令后，输出的统计信息。

在图13-5中，我们看到 vangogh是连到一个使用 IGMP的网络上的，但是 vangogh没有加入任何多播组，因为igps_snd_reports是0。

netstat -p igmp 输出	igmpstat 成员
18774 messages received	igps_rcv_total
0 messages received with too few bytes	igps_rcv_tooshort
0 messages received with bad checksum	igps_rcv_badsum
18774 membership queries received	igps_rcv_queries
0 membership queries received with invalid field(s)	igps_rcv_badqueries
0 membership reports received	igps_rcv_reports
0 membership reports received with invalid field(s)	igps_rcv_badreports
0 membership reports received for groups to which we belong	igps_rcv_ourreports
0 membership reports sent	igps_snd_reports

图13-5 IGMP统计示例

13.2.3 SNMP变量

IGMP没有标准的SNMP MIB，但 [McCloghrie Farinacci 1994a]描述了一个IGMP的实验MIB。

13.3 igmp结构

IGMP报文只有8字节长。图13-6显示了Net/3使用的igmp结构。

```

43 struct igmp {
44     u_char  igmp_type;           /* version & type of IGMP message */
45     u_char  igmp_code;           /* unused, should be zero */
46     u_short igmp_cksum;          /* IP-style checksum */
47     struct in_addr igmp_group;   /* group address being reported */
48 };                               /* (zero for queries) */

```

igmp.h

图13-6 igmp结构

igmp_type包括一个4 bit的版本码和一个4 bit的类型码。图13-7显示了标准值。

版本	类型	igmp_type	描 述
1	1	0x11 (IGMP_HOST_MEMBERSHIP_QUERY)	成员关系查询
1	2	0x11 (IGMP_HOST_MEMBERSHIP_REPORT)	成员关系报告
1	3	0x13	DVMRP报文(第14章)

图13-7 IGMP报文类型

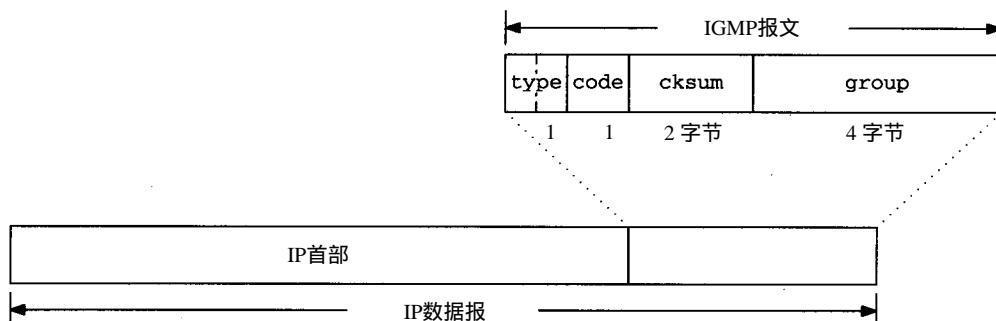


图13-8 IGMP 报文(省略 igmp_)

43-44 Net/3只使用版本1的报文。多播路由器发送1类报文(IGMP_HOST_MEMBERSHIP_QUERY)向本地网络上所有主机请求成员关系报告。对1类IGMP报文的响应是主机的一个2类报文(IGMP_HOST_MEMBERSHIP_REPORT)，报告它们的多播成员信息。3类报文在路由器之间传输多播选路信息(第14章)。主机不处理3类报文。本章后面部分只讨论1类和2类报文。

45-46 在IGMP版本1中没有使用igmp_code。igmp_cksum与IP类似，计算IGMP报文的所有8个字节。

47-48 对查询，igmp_group是0。对回答，它包括报告的多播组。

图13-8是相对于IP数据报的IGMP报文结构。

13.4 IGMP的protosw的结构

图13-9是IGMP的protosw结构。

成 员	Inetsw[5]	描 述
pr_type	SOCK_RAW	IGMP提供原始分组服务
pr_domain	&inetdomain	IGMP是Internet域的一部分
pr_protocol	IPPROTO_IGMP (2)	显示在IP首部的ip_p字段
pr_flags	PR_ATOMIC / PR_ADDR	插口层标志，协议处理不使用
pr_input	igmp_input	从IP层接收报文
pr_output	rip_output	向IP层发送IGMP报文
pr_ctlinput	0	IGMP没有使用
pr_ctloutput	rip_ctloutput	响应来自进程的管理请求
pr_usrreq	rip_usrreq	响应来自进程的通信请求
pr_init	igmp_init	为IGMP初始化
pr_fasttimo	igmp_fasttimo	进程挂起成员关系报告
pr_slowtimo	0	IGMP没有使用
pr_drain	0	IGMP没有使用
pr_sysctl	0	IGMP没有使用

图13-9 IGMP protosw 的结构

尽管进程有可能通过IGMP protosw入口发送原始IP分组，但在本章，我们只考虑内核如何处理IGMP报文。第32章讨论进程如何用原始插口访问IGMP。

三种事件触发IGMP处理：

- 一个本地接口加入一个新的多播组(13.5节)；
- 某个IGMP定时器超时(13.6节)；和
- 收到一个IGMP查询(13.7节)。

还有两种事件也触发本地IGMP处理，但结果不发送任何报文：

- 收到一个IGMP报告(13.7节)；和
- 某个本地接口离开一个多播组(13.8节)。

下一节将讨论这五种事件。

13.5 加入一个组：igmp_joingroup函数

在第12章中我们看到，当一个新的in_multi结构被创建时，in_addmulti调用igmp_joingroup。后面加入同一多播组的请求只增加in_multi结构里的引用计数；不调

用igmp_joingroup。igmp_joingroup如图13-10所示。

```

164 void
165 igmp_joingroup(inm)
166 struct in_multi *inm;
167 {
168     int      s = splnet();
169     if (inm->inm_addr.s_addr == igmp_all_hosts_group ||
170         inm->inm_ifp == &loif)
171         inm->inm_timer = 0;
172     else {
173         igmp_sendreport(inm);
174         inm->inm_timer = IGMP_RANDOM_DELAY(inm->inm_addr);
175         igmp_timers_are_running = 1;
176     }
177     splx(s);
178 }

```

igmp.c

图13-10 igmp_joingroup 函数

164-178 inm指向组的新in_multi结构。如果新的组是“所有主机组”，或成员关系请求是环回接口的，则inm_timer被禁止，igmp_joingroup返回。不报告“所有主机组”的成员关系，因为假定每个多播主机都是该组的成员。没必要向环回接口发送组成员报告，因为本地主机是在回路网络上的唯一系统，它已经知道它的成员状态了。

在其他情况下，新组的报告被立即发送，并根据组的情况为组定时器选择一个随机的值。全局标志位 igmp_timers_are_running被设置，表明至少使能一个定时器。igmp_fasttimo (13.6节)检查这个变量，避免不必要的处理。

59-73 当新组的定时器超时，就发布第2次成员关系报告。复制报告是无害的，当第一次报告丢失或被破坏时，有了它就保险了。IGMP_RANDOM_DELAY (13-11图)计算报告时延。

```

59 /*
60 * Macro to compute a random timer value between 1 and (IGMP_MAX_REPORTING_
61 * DELAY * countdown frequency). We generate a "random" number by adding
62 * the total number of IP packets received, our primary IP address, and the
63 * multicast address being timed-out. The 4.3 random() routine really
64 * ought to be available in the kernel!
65 */
66 #define IGMP_RANDOM_DELAY(multiaddr) \
67     /* struct in_addr multiaddr; */ \
68     ( (ipstat.ips_total + \
69         ntohl(IA_SIN(in_ifaddr)->sin_addr.s_addr) + \
70         ntohl((multiaddr).s_addr) \
71     ) \
72     % (IGMP_MAX_HOST_REPORT_DELAY * PR_FASTHZ) + 1 \
73 )

```

igmp_var.h

图13-11 IGMP_RANDOM_DELAY 函数

根据RFC 1122，报告定时器必须设成 0到10之间的随机秒数 (IGMP_MAX_HOST_REPORT_DELAY)。因为IGMP 定时器每秒被减去 5次(PR_FASTHZ)，所以IGMP_RANDOM_DELAY必须选择一个在1~50之间的随机数。如果r是把接到的所有IP分组数、主机的原始地址和多播组相加后得到的随机数，则

0 (rmod50) 49

且

1 (rmod50)+1 50

要避免为0，因为这会禁止定时器，并且不发送任何报告。

13.6 igmp_fasttimo函数

在讨论igmp_fasttimo之前，我们需要描述一下遍历in_multi结构的机制。

为找到各个in_multi结构，Net/3必须遍历每个接口的in_multi表。在遍历过程中，in_multistep结构(如图13-12所示)记录位置。

```

123 struct in_multistep {
124     struct in_ifaddr *i_ia;
125     struct in_multi *i_inm;
126 };

```

in_var.h

in_var.h

图13-12 in_multistep 函数

123-126 i_ia指向下一个in_ifaddr接口结构，i_inm指向当前接口的in_multi结构。

IN_FIRST_MULTI和IN_NEXT_MULTI宏(显示如图13-13)遍历该表。

```

147 /*
148  * Macro to step through all of the in_multi records, one at a time.
149  * The current position is remembered in "step", which the caller must
150  * provide. IN_FIRST_MULTI(), below, must be called to initialize "step"
151  * and get the first record. Both macros return a NULL "inm" when there
152  * are no remaining records.
153  */
154 #define IN_NEXT_MULTI(step, inm) \
155     /* struct in_multistep step; */ \
156     /* struct in_multi *inm; */ \
157 { \
158     if (((inm) = (step).i_inm) != NULL) \
159         (step).i_inm = (inm)->inm_next; \
160     else \
161         while ((step).i_ia != NULL) { \
162             (inm) = (step).i_ia->ia_multiaddrs; \
163             (step).i_ia = (step).i_ia->ia_next; \
164             if ((inm) != NULL) { \
165                 (step).i_inm = (inm)->inm_next; \
166                 break; \
167             } \
168         } \
169 }

```

in_var.h

```

170 #define IN_FIRST_MULTI(step, inm) \
171     /* struct in_multistep step; */ \
172     /* struct in_multi *inm; */ \
173 { \
174     (step).i_ia = in_ifaddr; \
175     (step).i_inm = NULL; \
176     IN_NEXT_MULTI((step), (inm)); \
177 }

```

in_var.h

图13-13 IN_FIRST_MULTI 和IN_NEXT_MULTI 结构

154-169 如果in_multi表有多个入口, i_inm就前进到下一个入口。当 IN_NEXT_MULTI到达多播表的最后时, i_ia就指向下一个接口, i_inm指向与该接口相关的第一个in_multi结构。如果该接口没有多播结构, while循环继续遍历整个接口表, 直到搜索完所有接口。

170-177 in_multistep数组初始化时, 指向in_ifaddr表的第一个in_ifaddr结构, i_inm设成空。IN_NEXT_MULTI找到第一个in_multi结构。

从图13-9我们知道, igmp_fasttimo是IGMP的快速超时函数, 每秒被调用 5次。igmp_fasttimo(如图13-14)递减多播报告定时器, 并在定时器超时时发送一个报告。

```

187 void
188 igmp_fasttimo()
189 {
190     struct in_multi *inm;
191     int s;
192     struct in_multistep step;
193     /*
194      * Quick check to see if any work needs to be done, in order
195      * to minimize the overhead of fasttimo processing.
196      */
197     if (!igmp_timers_are_running)
198         return;
199     s = splnet();
200     igmp_timers_are_running = 0;
201     IN_FIRST_MULTI(step, inm);
202     while (inm != NULL) {
203         if (inm->inm_timer == 0) {
204             /* do nothing */
205         } else if (--inm->inm_timer == 0) {
206             igmp_sendreport(inm);
207         } else {
208             igmp_timers_are_running = 1;
209         }
210         IN_NEXT_MULTI(step, inm);
211     }
212     splx(s);
213 }

```

igmp.c

图13-14 igmp_fasttimo 结构

187-198 如果igmp_timers_are_running为假, igmp_fasttimo立即返回, 不再浪费时间检查各个定时器。

199-213 igmp_fasttimo重新设置运行标志位, 用 IN_FIRST_MULTI初始化step和inm。igmp_fasttimo函数用while循环找到各个in_multi结构和IN_NEXT_MULTI宏。对每个结构:

- 如果定时器是0, 什么都不做。
- 如果定时器不是0, 则将其递减。如果到达0, 则发送一个IGMP组成员关系报告。
- 如果定时器还不是0, 则至少还有一个定时器在运行, 所以把 igmp_timers_are_running设成1。

igmp_sendreport函数

igmp_sendreport函数(图13-15)为一个多播组构造和发送IGMP报告报文。

```

214 static void
215 igmp_sendreport(inm)
216 struct in_multi *inm;
217 {
218     struct mbuf *m;
219     struct igmp *igmp;
220     struct ip *ip;
221     struct ip_options *imo;
222     struct ip_options simo;
223
224     MGETHDR(m, M_DONTWAIT, MT_HEADER);
225     if (m == NULL)
226         return;
227     /*
228      * Assume max_linkhdr + sizeof(struct ip) + IGMP_MINLEN
229      * is smaller than mbuf size returned by MGETHDR.
230      */
231     m->m_data += max_linkhdr;
232     m->m_len = sizeof(struct ip) + IGMP_MINLEN;
233     m->m_pkthdr.len = sizeof(struct ip) + IGMP_MINLEN;
234
235     ip = mtod(m, struct ip *);
236     ip->ip_tos = 0;
237     ip->ip_len = sizeof(struct ip) + IGMP_MINLEN;
238     ip->ip_off = 0;
239     ip->ip_p = IPPROTO_IGMP;
240     ip->ip_src.s_addr = INADDR_ANY;
241     ip->ip_dst = inm->inm_addr;
242
243     igmp = (struct igmp *) (ip + 1);
244     igmp->igmp_type = IGMP_HOST_MEMBERSHIP_REPORT;
245     igmp->igmp_code = 0;
246     igmp->igmp_group = inm->inm_addr;
247     igmp->igmp_cksum = 0;
248     igmp->igmp_cksum = in_cksum(m, IGMP_MINLEN);
249
250     imo = &simo;
251     bzero((caddr_t) imo, sizeof(*imo));
252     imo->imo_multicast_ifp = inm->inm_ifp;
253     imo->imo_multicast_ttl = 1;
254
255     /*
256      * Request loopback of the report if we are acting as a multicast
257      * router, so that the process-level routing demon can hear it.
258      */
259     {
260         extern struct socket *ip_mrouter;
261         imo->imo_multicast_loop = (ip_mrouter != NULL);
262     }
263     ip_output(m, NULL, NULL, 0, imo);
264
265     ++igmpstat.igps_snd_reports;
266 }

```

图13-15 igmp_sendreport 函数

214-232 唯一的参数inm指向被报告组的in_multi结构。igmp_sendreport分配一个新的mbuf，准备存放一个IGMP报文。igmp_sendreport为链路层首部留下空间，把mbuf

的长度和分组的长度设成 IGMP 报文的长度。

233-245 每次构造 IP 首部 and IGMP 报文的一个字段。数据报的源地址设成 `INADDR_ANY`，目的地址是被报告的多播组。`ip_output` 用输出接口的单播地址替换 `INADDR_ANY`。每个组成员和所有多播路由器都接收报告 (因为路由器接收所有 IP 多播)。

246-260 最后，`igmp_sentreport` 构造一个 `ip_moptions` 结构，并把它与报文一起传给 `ip_output`。与 `in_multi` 结构相关的接口被选做输出的接口；TTL 被设成 1，使报告只在本地网络上；如果本地系统被配置成路由器，则允许这个请求的多播环回。

进程级的多播路由器必须监听成员关系报告。在 12.14 节中我们看到，当系统被配置成多播路由器时，总是接收 IGMP 数据报。通过普通的运输层分用程序把报文传给 IGMP 的 `igmp_input` 和 `pr_input` 函数 (图 13-9)。

13.7 输入处理：igmp_input 函数

在 12.14 节中，我们描述了 `ipintr` 的多播处理部分。我们看到，多播路由器接受所有 IGMP 报文，但多播主机只接受那些到达接口是目的多播组成员的 IGMP 报文 (也即，那些接收它们的接口是组成员的查询和成员关系报告)。

标准协议分用机制把接受的报文传给 `igmp_input`。`igmp_input` 的开始和结束如图 13-16 所示。下面几节描述每种 IGMP 报文类型码。

```


52 void
53 igmp_input(m, iphlen)
54 struct mbuf *m;
55 int      iphlen;
56 {
57     struct igmp *igmp;
58     struct ip *ip;
59     int      igmplen;
60     struct ifnet *ifp = m->m_pkthdr.rcvif;
61     int      minlen;
62     struct in_multi *inm;
63     struct in_ifaddr *ia;
64     struct in_multistep step;
65     ++igmpstat.igps_rcv_total;
66     ip = mtod(m, struct ip *);
67     igmplen = ip->ip_len;
68     /*
69      * Validate lengths
70      */
71     if (igmplen < IGMP_MINLEN) {
72         ++igmpstat.igps_rcv_tooshort;
73         m_freem(m);
74         return;
75     }
76     minlen = iphlen + IGMP_MINLEN;
77     if ((m->m_flags & M_EXT || m->m_len < minlen) &&
78         (m = m_pullup(m, minlen)) == 0) {
79         ++igmpstat.igps_rcv_tooshort;
80         return;
81     }

```

igmp.c

图13-16 igmp_input 函数

```

82      /*
83       * Validate checksum
84       */
85      m->m_data += iphlen;
86      m->m_len -= iphlen;
87      igmp = mtod(m, struct igmp *);
88      if (in_cksum(m, igmplen)) {
89          ++igmpstat.igps_rcv_badsum;
90          m_freem(m);
91          return;
92      }
93      m->m_data -= iphlen;
94      m->m_len += iphlen;
95      ip = mtod(m, struct ip *);
96      switch (igmp->igmp_type) {
          
157     }
158     /*
159     * Pass all valid IGMP packets up to any process(es) listening
160     * on a raw IGMP socket.
161     */
162     rip_input(m);
163 }

```

igmp.c

图13-16 (续)

1. 验证IGMP报文

52-96 函数 `ipintr` 传递一个指向接受分组 (存放在一个 `mbuf` 中) 的指针 `m`，和数据报 IP 首部的大小 `iphlen`。

数据报的长度必须足够容纳一个 IGMP 报文 (`IGMP_MIN_LEN`)，并能被放在一个标准的 `mbuf` 首部中 (`m_pullup`)，而且还必须有正确的 IGMP 检验和。如果发现有任何错误，统计错误的个数，并自动丢弃该数据报，`igmp_input` 返回。

`igmp_input` 进程体根据 `igmp_type` 内的代码处理无效报文。记得在图 13-6 中，`igmp_type` 包含一个版本码和一个类型码。`switch` 语句基于 `igmp_type` (图 13-7) 中两个值的结合。下面几节分别讨论几种情况。

2. 把 IGMP 报文传给原始 IP

157-163 这个 `switch` 语句没有 `default` 情况。所有有效报文 (也就是，格式正确的报文) 被传给 `rip_input`，在 `rip_input` 里被提交给所有监听 IGMP 报文的进程。监听进程可以自由处理或丢弃那些具有内核不识别的版本或类型的 IGMP 报文。

`mrouted` 依靠对 `rip_input` 的调用接收成员关系查询和报告。

13.7.1 成员关系查询：IGMP_HOST_MEMBERSHIP_QUERY

RFC 1075 推荐多播路由器每 120 秒至少发布一次 IGMP 成员关系查询。把查询发到 224.0.0.1 组 (“所有主机组”)。图 13-17 显示了主机如何处理报文。

97-122 到达环回接口上的查询报文被自动丢弃 (习题 13.1)。查询报文被定义成发给 “所有

主机组”，到达其他地址的查询报文由 `igps_rcv_badqueries` 统计数量，并被丢弃。

```

97     case IGMP_HOST_MEMBERSHIP_QUERY:
98         ++igmpstat.igps_rcv_queries;

99     if (ifp == &loif)
100         break;

101     if (ip->ip_dst.s_addr != igmp_all_hosts_group) {
102         ++igmpstat.igps_rcv_badqueries;
103         m_freem(m);
104         return;
105     }
106     /*
107     * Start the timers in all of our membership records for
108     * the interface on which the query arrived, except those
109     * that are already running and those that belong to the
110     * "all-hosts" group.
111     */
112     IN_FIRST_MULTICAST(step, inm);
113     while (inm != NULL) {
114         if (inm->inm_ifp == ifp && inm->inm_timer == 0 &&
115             inm->inm_addr.s_addr != igmp_all_hosts_group) {
116             inm->inm_timer =
117                 IGMP_RANDOM_DELAY(inm->inm_addr);
118             igmp_timers_are_running = 1;
119         }
120         IN_NEXT_MULTICAST(step, inm);
121     }
122     break;

```

igmp.c

图13-17 IGMP查询报文的输入处理

接受查询报文并不会立即引起IGMP成员报告。相反，`igmp_input`为与接收查询的接口相关的各个组定时器设置一个随机的值 `IGMP_RANDOM_DELAY`。当某组的定时器超时，则 `igmp_fasttimo`发送一个成员关系报告，与此同时，其他所有收到查询的主机也进行同一动作。一旦某个主机上的某个特定组的随机定时器超时，就向该组多播一个报告。这个报告将取消其主机上的定时器，保证只有一个报告在网络上多播。路由器与其他组成员一样，接收该报告。

这个情况的一个例外就是“所有主机组”。这个组不设定定时器，也不发送报告。

13.7.2 成员关系报告：IGMP_HOST_MEMBERSHIP_REPORT

接收一个IGMP成员关系报告是我们在13.1节中提到的不会产生IGMP报文的两种事件之一。该报文的效果限于接收它的接口本地。图13-18显示了报文处理。

123-146 和发送到不正确多播组的成员关系报告一样，发到环回接口上的报告被丢弃。也就是说，报文必须寻址到报文内标识的组。

不完整地初始化的主机的源地址中可能没有网络号或主机号（或两者都没有）。`igmp_report`查看地址的A类网络部分，如果地址的网络或子网部分是0，这部分一定为0。如果是这种情况，则把源地址设成子网地址，其中包含正在接收接口的网络标识符和子网标识符。这样做的唯一原因是为了通知子网号所标识的正在接收接口上的某个进程级守护程序。

如果接收接口属于被报告的组，就把相关的报告定时器重新设成0。从而使发给该组的第一个报告能够制止其他主机发布报告。路由器只需知道网络上至少有一个接口是组的成员，

就无需维护一个明确的组成员表或计数器。

```

123     case IGMP_HOST_MEMBERSHIP_REPORT:
124         ++igmpstat.igps_rcv_reports;

125         if (ifp == &loif)
126             break;

127         if (!IN_MULTICAST(ntohl(igmp->igmp_group.s_addr)) ||
128             igmp->igmp_group.s_addr != ip->ip_dst.s_addr) {
129             ++igmpstat.igps_rcv_badreports;
130             m_freem(m);
131             return;
132         }
133         /*
134          * KLUDGE: if the IP source address of the report has an
135          * unspecified (i.e., zero) subnet number, as is allowed for
136          * a booting host, replace it with the correct subnet number
137          * so that a process-level multicast routing demon can
138          * determine which subnet it arrived from. This is necessary
139          * to compensate for the lack of any way for a process to
140          * determine the arrival interface of an incoming packet.
141          */
142         if ((ntohl(ip->ip_src.s_addr) & IN_CLASSA_NET) == 0) {
143             IFP_TO_IA(ifp, ia);
144             if (ia)
145                 ip->ip_src.s_addr = htonl(ia->ia_subnet);
146         }
147         /*
148          * If we belong to the group being reported, stop
149          * our timer for that group.
150          */
151         IN_LOOKUP_MULTI(igmp->igmp_group, ifp, inm);
152         if (inm != NULL) {
153             inm->inm_timer = 0;
154             ++igmpstat.igps_rcv_ourreports;
155         }
156         break;

```

图13-18 IGMP报告报文的输入处理

13.8 离开一个组：igmp_leavegroup函数

我们在12章中看到，当in_multi结构中的引用计数器跳到0时，in_delmulti调用igmp_leavegroup。如图13-19所示。

```

179 void
180 igmp_leavegroup(inm)
181 struct in_multi *inm;
182 {
183     /*
184      * No action required on leaving a group.
185      */
186 }

```

图13-19 igmp_leavegroup 函数

179-186 当一个接口离开一个组时, IGMP没有采取任何动作。不发明确的通知——下一次多播路由器发布 IGMP查询时, 接口不为该组生成 IGMP报告。如果没有为某个组生成报告, 则多播路由器就假定所有接口已经离开该组, 并停止把到该组的分组在网络上多播。

如果当一个报告被挂起时, 接口离开了该组 (就是说, 此时组的报告定时器正在计时), 就不再发送该报告, 因为当 `icmp_leavegroup` 返回时, `in_delmulti` (图12-36) 已经把组的定时器及其相关的 `in_multi` 结构丢掉了。

13.9 小结

本章我们讲述了 IGMP, IGMP在一个网络上的主机和路由器之间传递 IP多播成员信息。当一个接口加入一个组时, 或按照多播路由器发布的 IGMP报告查询报文的要求, 生成 IGMP成员关系报告。

设计IGMP使交换成员信息所需要的报文数最少:

- 当主机加入一个组时, 宣布它们的成员关系;
- 对成员关系查询的响应被推迟一个随机的时间, 而且第一个响应抑制了其他的响应;
- 当主机离开一个组时, 不发通知报文;
- 每分钟发的成员查询不超过一次。

多播路由器与其他路由器共享自己收集的 IGMP信息(第14章), 以便于把多播数据报传给多播目的组的远程成员。

习题

- 13.1 为什么不需要响应在环回接口上到达的 IGMP查询?
- 13.2 验证图13-15中226到229行的假设。
- 13.3 对在点到点网络接口上到达的成员关系查询, 是否有必要设置随机的延迟时间?