第11章　ICMP：Internet控制报文协议

11.1　引言

ICMP在IP系统间传递差错和管理报文，是任何IP实现必需和要求的组成部分。ICMP的规范见RFC 792 [Postel 1981b]。RFC 950 [Mogul和Postel 1985]和RFC 1256 [Deering 1991a]定义了更多的ICMP报文类型。RFC 792 [Braden 1989a]提供了重要的ICMP细节。

ICMP有自己的传输协议号(1)，允许ICMP报文在IP数据报内携带。应用程序可以直接从第32章讨论的原始IP接口发送或接收ICMP报文。

我们可把ICMP报文分成两类：差错和查询。查询报文是用一对请求和回答定义的。ICMP差错报文通常包含了引起错误的IP数据报的第一个分片的IP首部(和选项)，加上该分片数据部分的前8个字节。标准假定这8个字节包含了该分组运输层首部的所有分用信息，这样运输层协议可以向正确的进程提交ICMP差错报文。

TCP和UDP端口号在它们首部的前8个字节内出现。

图11-1显示了所有目前定义的ICMP报文。双线上面的是ICMP请求和回答报文；双线下面的是ICMP差错报文。

type和code	描　述	PRC_
ICMP_ECHO	回显请求	
ICMP_ECHOREPLY	回显回答	
ICMP_TSAMP	时间戳请求	
ICMP_TSTAMPREPLY	时间戳回答	
ICMP_MASKREQ	地址掩码请求	
ICMP_MASKREPLY	地址掩码回答	
ICMP_IREQ	信息请求(过时的)	
ICMP_IREQREPLY	信息回答(过时的)	
ICMP_ROUTERADVERT	路由器通告	
ICMP_ROUTESOLICIT	路由器请求	
ICMP_REDIRECT	有更好的路由	
ICMP_REDIRECT_NET	网络有更好的路由	PRC_REDIRECT_HOST
ICMP_REDIRECT_HOST	主机有更好的路由	PRC_REDIRECT_HOST
ICMP_REDIRECT_TOSNET	TOS和网络有更好的路由	PRC_REDIRECT_HOST
ICMP_REDIRECT_TOSHOST	TOS和主机有更好的路由	PRC_REDIRECT_HOST
其他	不识别码	
ICMP_UNREACH	目的主机不可达	
ICMP_UNREACH_NET	网络不可达	PRC_UNREACH_NET
ICMP_UNREACH_HOST	主机不可达	PRC_UNREACH_HOST

图11-1　ICMP报文类型和代码

type和code	描　述	PRC_
ICMP_UNREACH_PROTOCOL	目的主机上协议不能用	PRC_UNREACH_PROTOCOL
ICMP_UNREACH_PORT	目的主机上端口没有被激活	PRC_UNREACH_PORT
ICMP_UNREACH_SRCFAIL	源路由失败	PRC_UNREACH_SRCFAIL
ICMP_UNREACH_NEEDFRAG	需要分片并设置DF比特	PRC_MSGSIZE
ICMP_UNREACH_NET_UNKNOWN	目的网络未知	PRC_UNREACH_NET
ICMP_UNREACH_HOST_UNKNOWN	目的主机未知	PRC_UNREACH_HOST
ICMP_UNREACH_ISOLATED	源主机被隔离	PRC_UNREACH_HOST
ICMP_UNREACH_NET_PROHIB	从管理上禁止与目的网络通信	PRC_UNREACH_NET
ICMP_UNREACH_HOST_PROHIB	从管理上禁止与目的主机通信	PRC_UNREACH_HOST
ICMP_UNREACH_TOSNET	对服务类型，网络不可达	PRC_UNREACH_NET
ICMP_UNREACH_TOSHOST	对服务类型，主机不可达	PRC_UNREACH_HOST
13	用过滤从管理上禁止通信	
14	主机优先违规	
15	事实上优先切断	
其他	不识别码	
ICMP_TIMXCEED	超时	
ICMP_TIMXCEED_INTRANS	传送过程中IP生存期到期	PRC_TIMXCEED_INTRANS
ICMP_TIMXCEED_REASS	重装生存期到期	PRC_TIMXCEED_REASS
其他	不识别码	
ICMP_PRRAMPROB	IP首部的问题	
0	未指明首部差错	PRC_PARAMPROB
ICMP_PRRAMPROB_OPTABSENT	丢失需要的选项	PRC_PARAMPROB
其他	无效字节的字节偏移	
ICMP_SOURCEQUENCH	要求放慢发送	PRC_QUENCH
其他	不识别类型	

图11-1 (续)

图11-1和图11-2中含有大量信息：

- PRC_栏显示了Net/3处理的与协议无关的差错码(11.6节)和ICMP报文之间的映射。对请求和回答，这一列是空的。因为在这种情况下不会产生差错。如果对一个 ICMP差错，这一行为空，说明Net/3不识别该码，并自动丢弃该差错报文。
- 图11-3显示了我们讨论图11-2所列函数的位置。
- icmp_input栏是icmp_input为每个ICMP报文调用的函数。
- UDP栏是为UDP插口处理ICMP报文的函数。
- TCP栏是为TCP插口处理ICMP报文的函数。注意，是tcp_quench处理ICMP源站抑制差错，而不是tcp_notify。
- 如果errno栏为空，内核不向进程报告ICMP报文。
- 表的最后一行显示，在用于接收 ICMP报文的进程的接收点上，不识别的 ICMP报文被提交给原来的IP协议。

在Net/3中，ICMP是作为IP之上的一个运输层协议实现的，它不产生差错或请求；它代表

其他协议格式化并发送报文。 ICMP传递到达的差错，并向适当的传输协议或等待 ICMP报文的进程发出回答。另一方面， ICMP用一个合适的ICMP回答响应大多数ICMP请求。图11-4对此作了总结。

type 和 code	icmp_input	UDP	TCP	errno
ICMP_ECHO	icmp_reflect			
ICMP_ECHOREPLY	rip_input			
ICMP_TSTAMP	icmp_reflect			
ICMP_TSTAMPREPLY	rip_input			
ICMP_MASKREQ	icmp_reflect			
ICMP_MASKREPLY	rip_input			
ICMP_IREQ	rip_input			
ICMP_IREQREPLY	rip_input			
ICMP_ROUTERADVERT	rip_input			
ICMP_ROUTERSOLICIT	rip_input			
ICMP_REDIRECT				
ICMP_REDIRECT_NET	pfctlinput	in_rtchange	in_rtchange	
ICMP_REDIRECT_HOST	pfctlinput	in_rtchange	in_rtchange	
ICMP_REDIRECT_TOSNET	pfctlinput	in_rtchange	in_rtchange	
ICMP_REDIRECT_TOSHOST	pfctlinput	in_rtchange	in_rtchange	
其他	rip_input			
ICMP_UNREACH				
ICMP_UNREACH_NET	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_HOST	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_PROTOCOL	pr_ctlinput	udp_notify	tcp_notify	ECONNREFUSED
ICMP_UNREACH_PORT	pr_ctlinput	udp_notify	tcp_notify	ECONNREFUSED
ICMP_UNREACH_SRCFAIL	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_NEEDFRAG	pr_ctlinput	udp_notify	tcp_notify	EMSGSIZE
ICMP_UNREACH_NET_UNKNOWN	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_HOST_UNKNOWN	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_ISOLATED	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_NET_PROHIB	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_HOST_PROHIB	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_TOSNET	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
ICMP_UNREACH_TOSHOST	pr_ctlinput	udp_notify	tcp_notify	EHOSTUNREACH
13	rip_input			
14	rip_input			
15	rip_input			
其他	rip_input			
ICMP_TIMXCEED				
ICMP_TIMXCEED_INTRANS	pr_ctlinput	udp_notify	tcp_notify	
ICMP_TIMXCEED_REASS	pr_ctlinput	udp_notify	tcp_notify	
其他	rip_input			
ICMP_PARAMPROB				
0	pr_ctlinput	udp_notify	tcp_notify	ENOPROTOOPT
ICMP_PARAMPROB_OPTABSENT	pr_ctlinput	udp_notify	tcp_notify	ENOPROTOOPT
其他	rip_input			
ICMP_SOURCEQUENCH	pr_ctlinput	udp_notify	tcp_quench	
其他	rip_input			

图11-2 ICMP报文类型和代码(续)

函　数	描　述	引　用
icmp_reflect	为ICMP生成回答	11.12节
in_rtchange	更新IP路由表	图22-34
pfctlinput	向所有协议报告差错	7.7节
pr_ctlinput	向与插口有关的协议报告差错	7.4节
rip_input	进程不识别的ICMP报文	32.5节
tcp_notify	向进程报告差错或忽略	图27-12
tcp_quench	放慢输出	图27-13
udp_notify	向进程报告差错	图23-31

图11-3　ICMP输入处理时调用的函数

ICMP报文类型	到　达	输　出
请求	向ICMP请求生成回答	由某个进程生成
回答	传给原始IP	由内核生成
差错	传给传输协议和原始IP	由IP或传输协议生成
未知	传给原始IP	由某个进程生成

图11-4　ICMP报文处理

11.2　代码介绍

图11-5的两个文件中有本章讨论的ICMP数据结构、统计量和处理的程序。

文　件	描　述
netinet/ip_icmp.h	ICMP结构定义
netinet/ip_icmp.c	ICMP处理

图11-5　本章定义的文件

11.2.1　全局变量

本章介绍的全局变量如图11-6所示。

变　量	类　型	描　述
icmpmaskrepl	int	使ICMP地址掩码回答的返回有效
icmpstat	struct icmpstat	ICMP统计量(图11-7)

图11-6　本章介绍的全局变量

11.2.2　统计量

统计量是由图11-7所示的`icmpstat`结构的成员收集的。

icmpstat成员	描　述	SNMP使用的
icps_oldicmp	因为数据报是一个ICMP报文而丢弃的差错数	•
icps_oldshort	因为IP数据报太短而丢弃的差错数	•

图11-7　本章收集的统计信息

icmpstat成员	描 述	SNMP使用的
icps_badcode	由于无效码而丢弃的ICMP报文数	•
icps_badlen	由于无效的ICMP体而丢弃的ICMP报文数	•
icps_checksum	由于坏的ICMP检验和而丢弃的ICMP报文数	•
icps_tooshort	由于ICMP首部太短而丢弃的报文数	•
icps_outhist[]	输出计数器数组;每种ICMP类型对应一个	•
icps_inhist[]	输入计数器数组;每种ICMP类型对应一个	•
icps_error	icmp_error的调用(重定向除外)数	
icps_reflect	内核反映的ICMP报文数	

图11-7 (续)

在分析程序时，我们会看到计数器是递增的。

图11-8显示的是执行netstat -s命令输出的统计信息的例子。

netstat -s 输出	icmpstat 成员
84124 calls to icmp_error	icps_error
0 errors not generated 'cuz old message was icmp	icps_oldicmp
Output histogram:	icps_outhist[]
echo reply: 11770	ICMP_ECHOREPLY
destination unreachable: 84118	ICMP_UNREACH
time exceeded: 6	ICMP_TIMXCEED
6 messages with bad code fields	icps_badcode
0 messages < minimum length	icps_badlen
0 bad checksums	icps_checksum
143 messages with bad length	icps_tooshort
Input histogram:	icps_inhist[]
echo reply: 793	ICMP_ECHOREPLY
destination unreachable: 305869	ICMP_UNREACH
source quench: 621	ICMP_SOURCEQUENCH
routing redirect: 103	ICMP_REDIRECT
echo: 11770	ICMP_ECHO
time exceeded: 25296	ICMP_TIMXCEED
11770 message responses generated	icps_reflect

图11-8 ICMP统计信息示例

11.2.3 SNMP变量

图11-9显示了SNMP ICMP组的变量与Net/3收集的统计量之间的关系。

SNMP变量	icmpstat成员	描 述
icmpInMsgs	见正文	
icmpInErrors	icps_badcode + icps_badlen + icps_checksum + icps_tooshort	收到的ICMP报文数 由于错误丢弃的ICMP报文数
icmpInDestUnreachs icmpInTimeExcds icmpInParmProbs icmpInSrcQuenchs icmpInRedirects icmpInEchos	icps_inhist[]计数器	每一类收到的ICMP报文数

图11-9 ICMP组内的简单SNMP变量

icmpInEchoReps icmpInTimestamps icmpInTimestampReps icmpInAddrMasks icmpInAddrMaskReps		
icmpOutMsgs icmpOutErrors	见正文 icps_oldicmp + icps_oldshort	发送的ICMP报文数 由于一个错误而没有发送的ICMP错误数
icmpOutDestUnreachs icmpOutTimeExcds icmpOutParmProbs icmpOutSrcQuenchs icmpOutRedirects icmpOutEchos icmpOutEchoReps icmpOutTimestamps icmpOutTimestampReps icmpOutAddrMasks icmpOutAddrMaskReps	icps_outhist[]计数器	每一类发送的ICMP报文数

图11-9 (续)

icmpInMsgs是icps_inhist数组和icmpInErrors中的计数之和，icmpOutMsgs是icps_outist数组和icmpOutErrors中的计数之和。

11.3 icmp结构

Net/3通过图11-10中的icmp结构访问某个ICMP报文。

42-45 icmp_type标识特定报文，icmp_code进一步指定该报文(图11-1的第1栏)。计算icmp_cksum的算法与IP首部检验和相同，保护整个ICMP 报文(像IP一样，不仅仅保护首部)。

46-79 联合icmp_hun(首部联合)和icmp_dun(数据联合)按照icmp_type和icmp_code访问多种ICMP报文。每种ICMP报文都使用icmp_hun；只有一部分报文用icmp_dun。没有使用的字段必须设置为0。

80-86 我们已经看到，利用其他嵌套的结构(例如 mbuf、le_softc和ether_arp)，#define宏可以简化对结构成员的访问。

图11-11显示了ICMP报文的整体结构，并再次强调 ICMP报文是封装在IP数据报里的。我们将在分析程序时，分析所遇报文的特定结构。

ip_icmp.h

```
42 struct icmp {
43     u_char  icmp_type;          /* type of message, see below */
44     u_char  icmp_code;          /* type sub code */
45     u_short icmp_cksum;         /* ones complement cksum of struct */
46     union {
47         u_char  ih_pptr;        /* ICMP_PARAMPROB */
48         struct in_addr ih_gwaddr;   /* ICMP_REDIRECT */
49         struct ih_idseq {
50             n_short icd_id;
51             n_short icd_seq;
52         } ih_idseq;
53         int     ih_void;

54         /* ICMP_UNREACH_NEEDFRAG -- Path MTU Discovery (RFC1191) */
55         struct ih_pmtu {
```

图11-10 icmp 结构

```
56              n_short ipm_void;
57              n_short ipm_nextmtu;
58          } ih_pmtu;
59      } icmp_hun;
60  #define icmp_pptr    icmp_hun.ih_pptr
61  #define icmp_gwaddr  icmp_hun.ih_gwaddr
62  #define icmp_id      icmp_hun.ih_idseq.icd_id
63  #define icmp_seq     icmp_hun.ih_idseq.icd_seq
64  #define icmp_void    icmp_hun.ih_void
65  #define icmp_pmvoid  icmp_hun.ih_pmtu.ipm_void
66  #define icmp_nextmtu     icmp_hun.ih_pmtu.ipm_nextmtu
67      union {
68          struct id_ts {
69              n_time  its_otime;
70              n_time  its_rtime;
71              n_time  its_ttime;
72          } id_ts;
73          struct id_ip {
74              struct ip idi_ip;
75              /* options and then 64 bits of data */
76          } id_ip;
77          u_long  id_mask;
78          char    id_data[1];
79      } icmp_dun;
80  #define icmp_otime   icmp_dun.id_ts.its_otime
81  #define icmp_rtime   icmp_dun.id_ts.its_rtime
82  #define icmp_ttime   icmp_dun.id_ts.its_ttime
83  #define icmp_ip      icmp_dun.id_ip.idi_ip
84  #define icmp_mask    icmp_dun.id_mask
85  #define icmp_data    icmp_dun.id_data
86  };
```

ip_icmp.h

图11-10 (续)

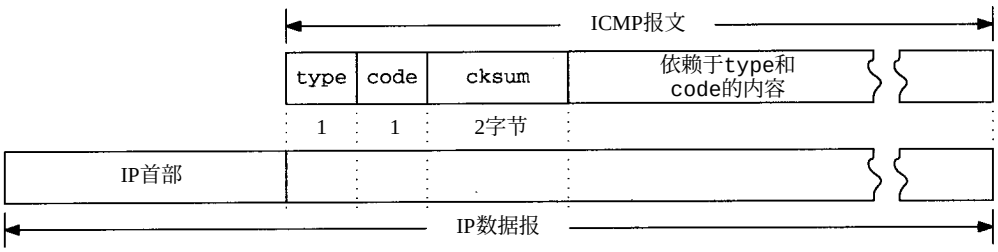

图11-11 一个ICMP报文(省略icmp_)

11.4 ICMP 的protosw结构

inetsw[4](图11-13)的protosw结构描述了ICMP，并支持内核和进程对协议的访问。图11-12显示了该结构。在内核里，icmp_input处理到达的ICMP报文，进程产生的外出ICMP报文由rip_output处理。以rip_开头的三个函数将在第32章中讨论。

成 员	inetsw[4]	描 述
pr_type	*SOCK_RAW*	ICMP提供原始分组服务
pr_domain	*&inetdomain*	ICMP是Internet域的一部分

图11-12 ICMP 的inetsw 项

成 员	inetsw[4]	描 述
pr_protocol	*IPPROTO_ICMP (1)*	出现在IP首部的ip_p字段中
pr_flags	*PR_ATOMIC|PR_ADDR*	插口层标志，ICMP不使用
pr_input	*icmp_input*	从IP层接收ICMP报文
pr_output	*rip_output*	将ICMP报文发送到IP层
pr_ctlinput	*0*	ICMP不使用
pr_ctloutput	*rip_ctloutput*	响应来自一个进程的管理请求
pr_usrreq	*rip_usrreq*	响应来自一个进程的通信请求
pr_init	*0*	ICMP不使用
pr_fasttimo	*0*	ICMP不使用
pr_slowtimo	*0*	ICMP不使用
pr_drain	*0*	ICMP不使用
pr_sysctl	*0*	ICMP不使用

图11-12 (续)

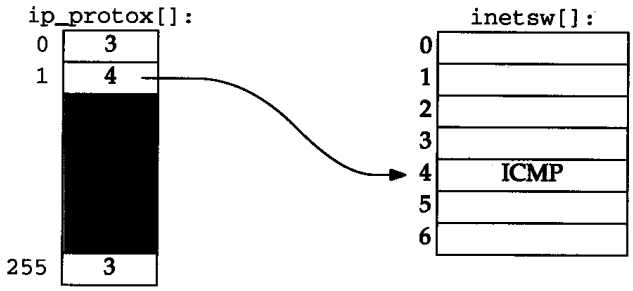

图11-13 数值为1的ip_p 选择了inetsw[4]

11.5 输入处理：icmp_input函数

回想起ipintr对数据报进行分用是根据IP首部中的传输协议编号ip_p。对于ICMP报文，ip_p是1，并通过ip_protox选择inetsw[4]。

当一个ICMP报文到达时，IP层通过inetsw[4]的pr_input函数，间接调用icmp_input(图10-11)。

我们将看到，在icmp_input中，每一个ICMP报文要被处理3次：被icmp_input处理一次；被与ICMP差错报文中的IP分组相关联的传输协议处理一次；被记录收到ICMP报文的进程处理一次。ICMP输入处理过程的总的构成情况如图11-14所示。

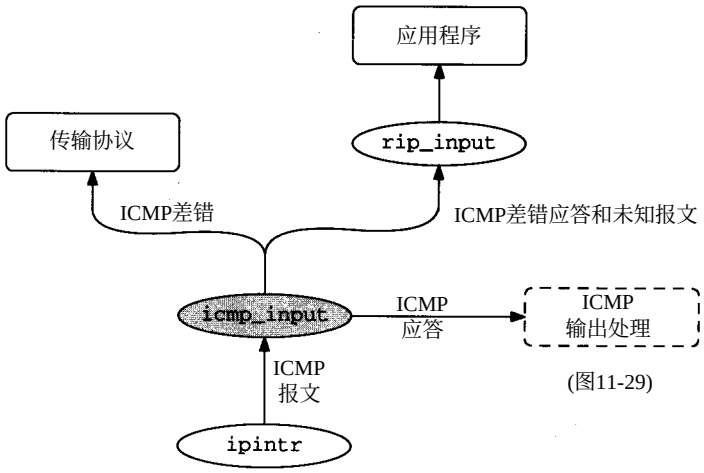

图11-14 ICMP的输入处理过程

我们将在以下5节(11.6～11.10)讨论icmp_input：(1) 验证收到的报文；(2) ICMP差错报文；(3) ICMP请求报文；(4) ICMP重定向报文；(5) ICMP回答报文。icmp_input函数的第一部分如图11-15所示。

ip_icmp.c

```
131 static struct sockaddr_in icmpsrc = { sizeof (struct sockaddr_in), AF_INET };
132 static struct sockaddr_in icmpdst = { sizeof (struct sockaddr_in), AF_INET };
133 static struct sockaddr_in icmpgw = { sizeof (struct sockaddr_in), AF_INET };
134 struct sockaddr_in icmpmask = { 8, 0 };

135 void
136 icmp_input(m, hlen)
137 struct mbuf *m;
138 int     hlen;
139 {
140     struct icmp *icp;
141     struct ip *ip = mtod(m, struct ip *);
142     int     icmplen = ip->ip_len;
143     int     i;
144     struct in_ifaddr *ia;
145     void    (*ctlfunc) (int, struct sockaddr *, struct ip *);
146     int     code;
147     extern u_char ip_protox[];

148     /*
149      * Locate icmp structure in mbuf, and check
150      * that not corrupted and of at least minimum length.
151      */
152     if (icmplen < ICMP_MINLEN) {
153         icmpstat.icps_tooshort++;
154         goto freeit;
155     }
156     i = hlen + min(icmplen, ICMP_ADVLENMIN);
157     if (m->m_len < i && (m = m_pullup(m, i)) == 0) {
158         icmpstat.icps_tooshort++;
159         return;
160     }
161     ip = mtod(m, struct ip *);
162     m->m_len -= hlen;
163     m->m_data += hlen;
164     icp = mtod(m, struct icmp *);
165     if (in_cksum(m, icmplen)) {
166         icmpstat.icps_checksum++;
167         goto freeit;
168     }
169     m->m_len += hlen;
170     m->m_data -= hlen;

171     if (icp->icmp_type > ICMP_MAXTYPE)
172         goto raw;
173     icmpstat.icps_inhist[icp->icmp_type]++;
174     code = icp->icmp_code;
175     switch (icp->icmp_type) {
```

图11-15 icmp_input 函数

```
317     default:
318         break;
319     }
320   raw:
321     rip_input(m);
322     return;

323   freeit:
324     m_freem(m);
325 }
```

ip_icmp.c

图11-15 (续)

1. 静态结构

131-134 因为icmp_input是在中断时调用的，此时堆栈的大小是有限的。所以，为了在每次调用icmp_input时，避免动态分配造成的延迟，以及使堆栈最小，这4个结构是动态分配的。icmp_input把这4个结构用作临时变量。

icmpsrc的命名容易引起误解，因为icmp_input把它用作临时sockaddr_in变量，而它也从未包含过源站地址。在Net/2版本的icmp_input中，在报文被raw_input函数提交给原始IP之前，报文的源站地址在函数的最后被复制到icmpsrc中。而Net/3调用只需要一个指向该分组的指针的rip_input，而不是raw_input。虽然有这个改变，但是icmpsrc仍然保留了在Net/2中的名字。

2. 确认报文

135-139 icmp_input希望收到的ICMP报文(m)中含有一个指向该数据报的指针，以及该数据报IP首部的字节长度(hlen)。图11-16列出了几个在icmp_input里用于简化检测无效ICMP报文的常量。

常量／宏	值	描　述
ICMP_MINLEN	8	ICMP报文大小的最小值
ICMP_TSLEN	20	ICMP时间戳报文大小
ICMP_MASKLEN	12	ICMP地址掩码报文大小
ICMP_ADVLENMIN	36	ICMP差错(建议)报文大小的最小值 (*IP* + *ICMP* + *BADIP* = 20 + 8 + 8 = 36)
ICMP_ADVLEN(p)	36 + *optsize*	ICMP差错报文的大小，包含无效分组p的IP选项的*optsize*字节

图11-16 ICMP引用的用来验证报文的常量

140-160 icmp_input从ip_len取出ICMP报文的大小，并把它存放在icmplen中。第8章讲过，ipintr从ip_len中排除了IP首部的长度。如果报文长度太短，不是有效报文，就生成icps_tooshort，并丢弃该报文。如果在第一个mbuf中，ICMP首部和IP首部不是连续的，则由m_pullup保证ICMP首部以及封闭的IP分组的IP首部在同一个mbuf中。

3. 验证检验和

161-170 icmp_input隐藏mbuf中的IP首部，并用in_cksum验证ICMP的检验和。如果报文被破坏，就增加icps_checksum，并丢弃该报文。

4. 验证类型

171-175 如果报文类型(icmp_type)不在识别范围内，icmp_input就跳过switch执行

raw语句(图11-9)。如果在识别范围内，icmp_input复制icmp_code，switch按照icmp_type处理该报文。

在ICMP switch语句处理完后，icmp_input向rip_input发送ICMP报文，后者把ICMP报文发布给准备接收的进程。只有那些被破坏的报文(长度或检验和出错)以及只由内核处理的ICMP请求报文才不传给rip_input。在这两种情况下，icmp_input都立即返回，并跳过raw处的源程序。

5. 原始ICMP输入

317-325 icmp_input把到达的报文传给rip_input，rip_input依据报文里含有的协议及源站和目的站地址信息(32章)，把报文发布给正在监听的进程。

原始IP机制允许进程直接发送和接收ICMP报文，这样做有几个原因：

- 新ICMP报文可由进程处理而无需修改内核(例如，路由器通告，图11-28)。
- 可以用进程而无需用内核模块来实现发送ICMP请求和处理回答的机制(ping和traceroute)。
- 进程可以增加对报文的内核处理。与此类似，内核在更新完它的路由表后，会把ICMP重定向报文传给一个路由守护程序。

11.6 差错处理

我们首先考虑ICMP差错报文。当主机发出的数据报无法成功地提交给目的主机时，它就接收这些报文。目的主机或中间的路由器生成这些报文，并将它们返回到原来的系统。图11-17显示了多种ICMP差错报文的格式。

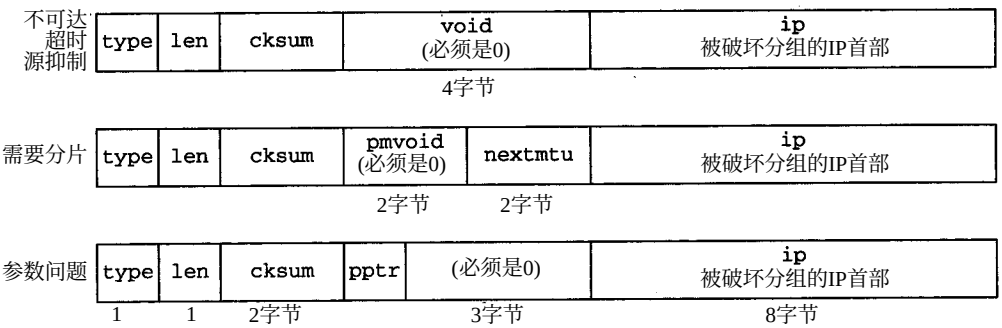

图11-17 ICMP差错报文(省略icmp_)

图11-18中的源程序来自图11-15中的switch语句。

ip_icmp.c

```
176     case ICMP_UNREACH:
177         switch (code) {
178         case ICMP_UNREACH_NET:
179         case ICMP_UNREACH_HOST:
180         case ICMP_UNREACH_PROTOCOL:
181         case ICMP_UNREACH_PORT:
182         case ICMP_UNREACH_SRCFAIL:
183             code += PRC_UNREACH_NET;
184             break;
185         case ICMP_UNREACH_NEEDFRAG:
```

图11-18 icmp_input 函数：差错报文

```
186             code = PRC_MSGSIZE;
187             break;

188         case ICMP_UNREACH_NET_UNKNOWN:
189         case ICMP_UNREACH_NET_PROHIB:
190         case ICMP_UNREACH_TOSNET:
191             code = PRC_UNREACH_NET;
192             break;

193         case ICMP_UNREACH_HOST_UNKNOWN:
194         case ICMP_UNREACH_ISOLATED:
195         case ICMP_UNREACH_HOST_PROHIB:
196         case ICMP_UNREACH_TOSHOST:
197             code = PRC_UNREACH_HOST;
198             break;

199         default:
200             goto badcode;
201         }
202         goto deliver;

203     case ICMP_TIMXCEED:
204         if (code > 1)
205             goto badcode;
206         code += PRC_TIMXCEED_INTRANS;
207         goto deliver;
208     case ICMP_PARAMPROB:
209         if (code > 1)
210             goto badcode;
211         code = PRC_PARAMPROB;
212         goto deliver;

213     case ICMP_SOURCEQUENCH:
214         if (code)
215             goto badcode;
216         code = PRC_QUENCH;

217       deliver:
218         /*
219          * Problem with datagram; advise higher level routines.
220          */
221         if (icmplen < ICMP_ADVLENMIN || icmplen < ICMP_ADVLEN(icp) ||
222             icp->icmp_ip.ip_hl < (sizeof(struct ip) >> 2)) {
223             icmpstat.icps_badlen++;
224             goto freeit;
225         }
226         NTOHS(icp->icmp_ip.ip_len);
227         icmpsrc.sin_addr = icp->icmp_ip.ip_dst;
228         if (ctlfunc = inetsw[ip_protox[icp->icmp_ip.ip_p]].pr_ctlinput)
229             (*ctlfunc) (code, (struct sockaddr *) &icmpsrc,
230                         &icp->icmp_ip);
231         break;

232       badcode:
233         icmpstat.icps_badcode++;
234         break;
```

ip_icmp.c

图11-18 (续)

176-216 对ICMP差错的处理是最少的，因为这主要是运输层协议的责任。imcp_input把icmp_type和icmp_code映射到一个与协议无关的差错码集上，该差错码是由PRC_常量

(图11-19)表示的。PRC_常量有一个隐含的顺序，正好与ICMP的code相对应。这就解释了为什么code是按一个PRC_常量递增的。

217-225 如果识别出类型和码，icmp_input就跳到deliver。如果没有识别出来，icmp_input就跳到badcode。

如果对所报告的差错而言，报文长度不正确，icps_badlen的值就加1，并丢弃该报文。Net/3总是丢弃无效的ICMP报文，也不生成有关该无效报文的ICMP差错。这样，就避免在两个有缺陷的实现之间形成无限的差错报文序列。

226-231 icmp_input调用运输层协议的pr_ctlinput函数，该函数根据原始数据报的ip_p，把到达分组分用到正确的协议，从而构造出原始的IP数据报。差错码(code)、原始IP数据报的目的地址(icmpsrc)以及一个指向无效数据报的指针(icmp_ip)被传给pr_ctlinput(如果是为该协议定义的)。图23-31和图27-12讨论这些差错。

232-234 最后，icps_badcode的值增加1，并终止switch语句的执行。

常 量	描 述
PRC_HOSTDEAD	主机似乎已关闭
PRC_IFDOWN	网络接口关闭
PRC_MSGSIZE	无效报文大小
PRC_PRRAMPROB	首部不正确
PRC_QUENCH	某人说要放慢
PRC_QUENCH2	阻塞比特要求放慢
PRC_REDIRECT_HOST	主机路由选择重定向
PRC_REDIRECT_NET	网络路由选择重定向
PRC_REDIRECT_TOSHOST	TOS和主机的重定向
PRC_REDIRECT_TOSNET	TOS和网络的重定向
PRC_ROUTEDEAD	如果可能，选择新的路由
PRC_TIMXCEED_INTRANS	传送过程中分组生命期到期
PRC_TIMXCEED_REASS	分片在重装过程中生命期到期
PRC_UNREACH_HOST	没有到主机的路由
PRC_UNREACH_NET	没有到网络的路由
PRC_UNREACH_PORT	目的主机称端口未激活
PRC_UNREACH_PROTOCOL	目的主机称协议不可用
PRC_UNREACH_SRCFAIL	源路由失败

图11-19 与协议无关的差错码

尽管PRC_常量表面上与协议无关，但它们主要还是基于Internet协议族。其结果是，当某个Internet协议族以外的协议把自己的差错映射到PRC_常量时，会失去可指定性。

11.7 请求处理

Net/3响应具有正确格式的ICMP请求报文，但把无效ICMP请求报文传给rip_input。第32章讨论了应用程序如何生成ICMP请求报文。

除路由器通告报文外，大多数Net/3所接收的ICMP请求报文都生成回答报文。为避免为回答报文分配新的mbuf，icmp_input把请求的缓存转换成回答的缓存，并返回给发送方。我

们将分别讨论各个请求。

11.7.1 回显询问：ICMP_ECHO和ICMP_ECHOREPLY

尽管ICMP非常简单，但是ICMP回显请求和回答却是网络管理员最有力的诊断工具。发出ICMP回显请求称为“ ping ”一个主机，也就是调用 ping程序一次。许多系统提供该程序来手工发送ICMP回显请求。卷1的第7章详细讨论了ping。

ping程序的名字依照了声纳脉冲 (soar ping)，用其他物体对声纳脉冲的反射所产生的回声确定它们的位置。卷 1把这个名字解释成Packet InterNet Groper，是不正确的。

图11-20是ICMP回显请求和回答报文的结构。

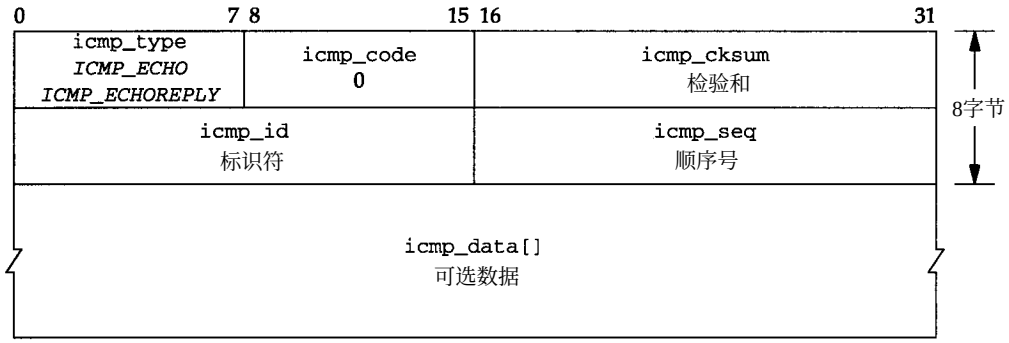

图11-20 ICMP回显请求和回答

icmp_code总是0。 icmp_id和icmp_seq设置成请求的发送方，回答中也不做修改。源系统可以用这些字段匹配请求和回答。 icmp_data中到达的所有数据也被反射。图 11-21是ICMP回显处理和icmp_input实现反射ICMP请求的源程序。

ip_icmp.c

```
235     case ICMP_ECHO:
236         icp->icmp_type = ICMP_ECHOREPLY;
237         goto reflect;

                /* other ICMP request processing */

277       reflect:
278         ip->ip_len += hlen;     /* since ip_input deducts this */
279         icmpstat.icps_reflect++;
280         icmpstat.icps_outhist[icp->icmp_type]++;
281         icmp_reflect(m);
282         return;
```

ip_icmp.c

图11-21 icmp_input 函数：回显请求和回答

235-237 通过把icmp_type变成ICMP_ECHOREPLY，并跳转到reflect发送回答，icmp_input把回显请求转换成了回显回答。

277-282 在为每个ICMP 请求构造完回答之后，icmp_input执行reflect处的程序。在这里，存储数据报正确的长度被恢复，在 icps_reflect和icps_outhist[]中分别计算请求的数量和ICMP报文的类型。icps_reflect(11.12节)把回答发回给请求方。

11.7.2 时间戳询问：ICMP_TSTAMP和ICMP_TSTAMPREPLY

ICMP时间戳报文如图11-22所示。

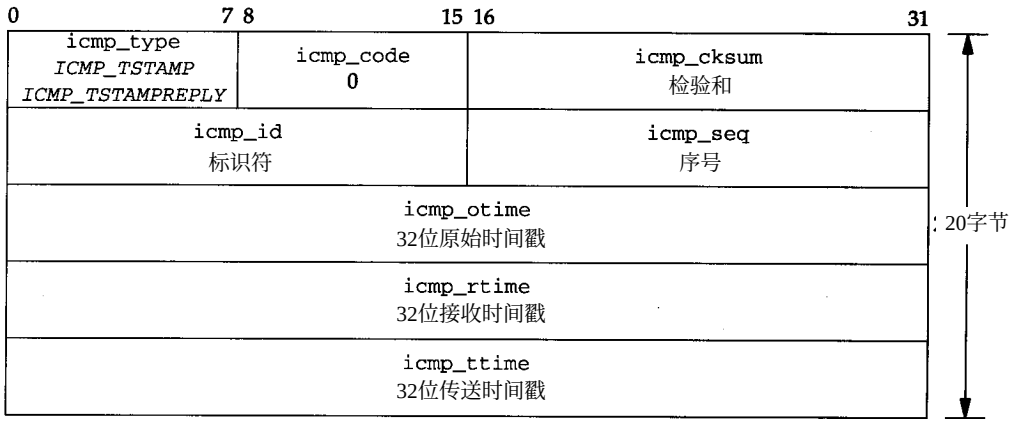

图11-22 ICMP时间戳请求和回答

icmp_code总是0。icmp_id和icmp_seq的作用与它们在ICMP回显报文中的一样。请求的发送方设置icmp_otime(发出请求的时间)；icmp_rtime(收到请求的时间)和icmp_ttime(发出回答的时间)由回答的发送方设置。所有时间都是从UTC午夜开始的毫秒数。如果时间值没有以标准单位记录，就把高位置位，与IP时间戳选项一样。

图11-23是实现时间戳报文的程序。

ip_icmp.c

```
238     case ICMP_TSTAMP:
239         if (icmplen < ICMP_TSLEN) {
240             icmpstat.icps_badlen++;
241             break;
242         }
243         icp->icmp_type = ICMP_TSTAMPREPLY;
244         icp->icmp_rtime = iptime();
245         icp->icmp_ttime = icp->icmp_rtime;  /* bogus, do later! */
246         goto reflect;
```

ip_icmp.c

图11-23 icmp_input 函数：时间戳请求和回答

238-246 icmp_input对ICMP的响应，包括：把icmp_type改成ICMP_TSTAMPREPLY，记录当前icmp_rtime和icmp_ttime，并跳转到reflect发送回答。

很难精确地设置icmp_rtime和icmp_ttime。当系统执行这段程序时，报文可能已经在IP输入队列中等待处理，这时设置icmp_rtime已经太晚了。类似地，数据报也可能在要求处理时在网络接口的传输队列中被延迟，这时设置icmp_ttime又太早了。为了把时间戳设置得更接近真实的接收和发送时间，必须修改每个网络的接口驱动程序，使其能理解ICMP报文(习题11.8)。

11.7.3 地址掩码询问：ICMP_MASKREQ和ICMP_MASKREPLY

ICMP 地址掩码请求和回答如图11-24所示。

RFC 950 [Mogul和Postel 1985]在原来的ICMP规范说明中增加了地址掩码报文，使系统能发现某个网络上使用的子网掩码。

除非系统被明确地配置成地址掩码的授权代理，否则， RFC 1122禁止向其发送掩码回答。这样，就避免系统与所有向它发出请求的系统共享不正确的地址掩码。如果没有管理员授权回答，系统也要忽略地址掩码请求。

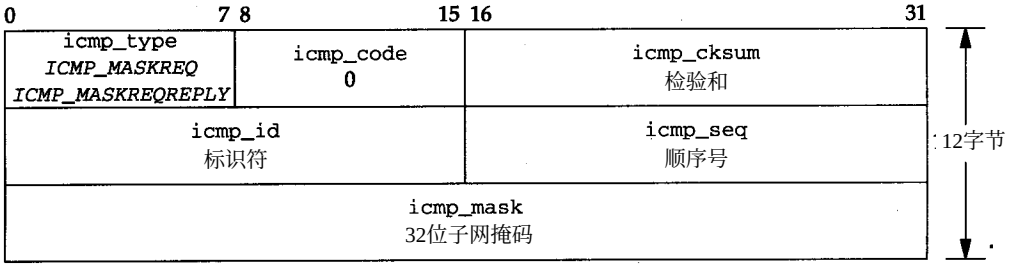

图11-24 ICMP地址掩码请求和回答

如果全局整数icmpmaskrepl非零， Net/3会响应地址掩码请求。 icmpmaskrepl的默认值是0， icmp_sysctl可以通过systctl(8)程序(11.14节)修改它。

Net/2系统中没有控制回答地址掩码请求的机制。其结果是，必须非常正确地配置Net/2接口的地址掩码；该信息是与网络上所有发出地址掩码请求的系统共享的。

地址掩码报文的处理如图11-25所示。

ip_icmp.c

```
247     case ICMP_MASKREQ:
248 #define satosin(sa) ((struct sockaddr_in *)(sa))
249         if (icmpmaskrepl == 0)
250             break;
251         /*
252          * We are not able to respond with all ones broadcast
253          * unless we receive it over a point-to-point interface.
254          */
255         if (icmplen < ICMP_MASKLEN)
256             break;
257         switch (ip->ip_dst.s_addr) {

258         case INADDR_BROADCAST:
259         case INADDR_ANY:
260             icmpdst.sin_addr = ip->ip_src;
261             break;

262         default:
263             icmpdst.sin_addr = ip->ip_dst;
264         }
265         ia = (struct in_ifaddr *) ifaof_ifpforaddr(
266                         (struct sockaddr *) &icmpdst, m->m_pkthdr.rcvif);
267         if (ia == 0)
268             break;
269         icp->icmp_type = ICMP_MASKREPLY;
270         icp->icmp_mask = ia->ia_sockmask.sin_addr.s_addr;
271         if (ip->ip_src.s_addr == 0) {
272             if (ia->ia_ifp->if_flags & IFF_BROADCAST)
273                 ip->ip_src = satosin(&ia->ia_broadaddr)->sin_addr;
274             else if (ia->ia_ifp->if_flags & IFF_POINTOPOINT)
275                 ip->ip_src = satosin(&ia->ia_dstaddr)->sin_addr;
276         }
```

ip_icmp.c

图11-25 icmp_input 函数：地址掩码请求和回答

247-256 如果没有配置响应掩码请求，或者该请求太短，这段程序就中止 switch的执行，并把报文传给rip_input(图11-15)。

在这里Net/3无法增加 icps_badlen。对其他ICMP 长度差错，它却增加 icps_badlen。

1. 选择子网掩码

257-267 如果地址掩码请求被发到0.0.0.0或255.255.255.255，源地址被保存在icmpdst中。在这里，ifaof_offoraddr把icmpdst作为源站地址，在同一网络上查找 in_ofaddr结构。如果源站地址是0.0.0.0或255.255.255.255，ifaof_offoraddr返回一个指针，该指针指向与接收接口相关的第一个IP地址。

default情况(针对单播或有向广播)为ifaof_ifpforaddr保存目的地址。

2. 转换成回答

269-270 通过改变icmp_type，并把所选子网掩码 ia_sockmask复制到icmp_mask，就完成了把请求转换成回答的工作。

3. 选择目的地址

271-276 如果请求的源站地址全 0(“ 该网络上的这台主机 ”，只在引导时用作源站地址，RFC 1122)，并且源站不知道自己的地址， Net/3必须广播这个回答，使源站系统接收到这个报文。在这种情况下，如果接收接口位于某个广播或点到点网络上，该回答的目的地址将分别是ia_broadaddr和ia_dstaddr。icmp_input把回答的目的地址放在 ip_src里，因为reflect处的程序(图11-21)会把源站和目的站地址倒过来。不改变单播请求的地址。

11.7.4 信息询问：ICMP_IREQ和ICMP_IREQREPLY

ICMP信息报文已经过时了。它们企图广播一个源和目的站地址字段的网络部分为全 0的请求，使系统发现连接的 IP网络的数量。响应该请求的主机将返回一个填好网络号的报文。主机还需要其他办法找到地址的主机部分。

RFC 1122推荐主机不要实现 ICMP信息报文，因为 RARP(RFC 903 [Finlayson et al., 1984])和BOOTP(RFC 951 [Croft和Gilmore 1985])更适于发现地址。RFC 1541 [Droms 1993]描述的一个新协议，动态主机配置协议(Dynamic Host Configuration Protocol，DHCP)，可能会取代或增强BOOTP的功能。它现在是一个建议的标准。

Net/2响应ICMP信息请求报文。但是，Net/3把它们传给rip_input。

11.7.5 路由器发现：ICMP_ROUTERADVERT和ICMP_ROUTERSOLICIT

RFC 1256 定义了ICMP路由器发现报文。Net/3内核不直接处理这些报文，而由rip_input把它们传给一个用户级守护程序，由它发送和响应这种报文。

卷1的9.6节讨论了这种报文的设计和运行。

11.8 重定向处理

图11-26显示了ICMP重定向报文的格式。

icmp_input中要讨论的最后一个case是ICMP_REDIRECT。如8.5节的讨论，当分组

被发给错误的路由器时，产生重定向报文。该路由器把分组转发给正确的路由器，并发回一个ICMP重定向报文，系统把信息记入它自己的路由表。

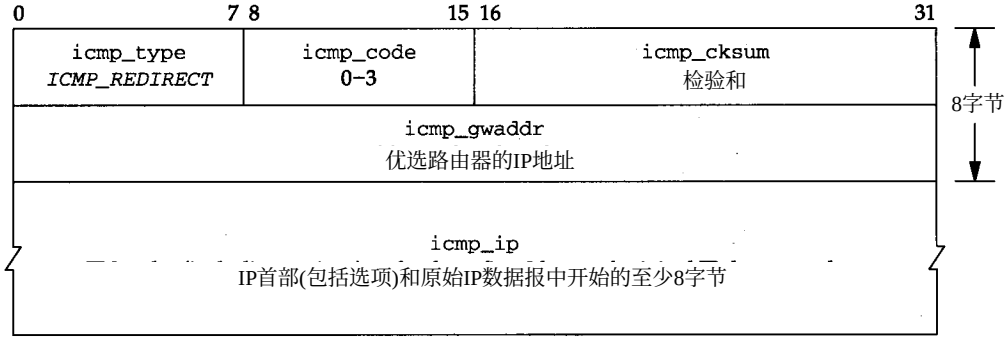

图11-26 ICMP重定向报文

图11-27显示了icmp_input用来处理重定向报文的程序。

ip_icmp.c

```
283     case ICMP_REDIRECT:
284         if (code > 3)
285             goto badcode;
286         if (icmplen < ICMP_ADVLENMIN || icmplen < ICMP_ADVLEN(icp) ||
287             icp->icmp_ip.ip_hl < (sizeof(struct ip) >> 2)) {
288             icmpstat.icps_badlen++;
289             break;
290         }
291         /*
292          * Short circuit routing redirects to force
293          * immediate change in the kernel's routing
294          * tables.  The message is also handed to anyone
295          * listening on a raw socket (e.g. the routing
296          * daemon for use in updating its tables).
297          */
298         icmpgw.sin_addr = ip->ip_src;
299         icmpdst.sin_addr = icp->icmp_gwaddr;
300         icmpsrc.sin_addr = icp->icmp_ip.ip_dst;
301         rtredirect((struct sockaddr *) &icmpsrc,
302                    (struct sockaddr *) &icmpdst,
303                    (struct sockaddr *) 0, RTF_GATEWAY | RTF_HOST,
304                    (struct sockaddr *) &icmpgw, (struct rtentry **) 0);
305         pfctlinput(PRC_REDIRECT_HOST, (struct sockaddr *) &icmpsrc);
306         break;
```

ip_icmp.c

图11-27 icmp_input 函数：重定向报文

1. 验证

283-290 如果重定向报文中含有未识别的ICMP 码，icmp_input就跳到badcode(图11-18的232行)；如果报文具有无效长度或封闭的IP分组具有无效首部长度，则中止switch。图11-16显示了ICMP差错报文的最小长度是36(ICMP_ADVLENMIN)。ICMP_ADVLEN(icp)是当icp所指向的分组有IP选项时，ICMP差错报文的最小长度。

291-300 icmp_input分别把重定向报文的源站地址(发送该报文的网关)、为原始分组推荐的路由器(第一跳目的地)和原始分组的最终目的地址分配给 icmpgw、icmpdst和icmpsrc。

这里，icmpsrc并不包含源站地址 —— 这是方便存放目的地址的位置，无需再定义一个sockaddr结构。

2. 更新路由

301-306 Net/3按照RFC 1122的推荐，等价地对待网络重定向和主机重定向。重定向信息被传给rtredirect，由这个函数更新路由表。重定向的目的地址(保存在 icmpsrc)被传给pfctlinput，由它通告重定向的所有协议域(7.3节)，使协议有机会把缓存的到目的站的路由作废。

按照RFC 1122，应该把网络重定向作为主机重定向对待，因为当目的网络划分了子网时，它们会提供不正确的路由信息。事实上， RFC 1009要求，在网络划分子网的情况下，不发送网络重定向。不幸的是，许多路由器违背了这个要求。 Net/3从不发重定向报文。

ICMP重定向报文是IP路由选择体系结构的基本组成部分。尽管被划分到差错报文类，但它却是在任何有多个路由器的网络正常运行时出现的。第 18章更详细讨论了IP路由选择问题。

11.9 回答处理

内核不处理任何ICMP回答报文。ICMP请求由进程产生，内核从不产生请求。所以，内核把它接收的所有回答传给等待 ICMP报文的进程。另外， ICMP路由器发现报文被传给rip_input。

ip_icmp.c

```
307         /*
308          * No kernel processing for the following;
309          * just fall through to send to raw listener.
310          */
311     case ICMP_ECHOREPLY:
312     case ICMP_ROUTERADVERT:
313     case ICMP_ROUTERSOLICIT:
314     case ICMP_TSTAMPREPLY:
315     case ICMP_IREQREPLY:
316     case ICMP_MASKREPLY:
317     default:
318         break;
319     }
320   raw:
321     rip_input(m);
322     return;
```

ip_icmp.c

图11-28 icmp_input 函数：回答报文

307-322 内核无需对ICMP回答报文做出任何反应，所以在raw处的switch语句后继续执行(图11-15)。注意，switch语句的default情况(未识别的ICMP报文)也把控制传给在raw处的代码。

11.10 输出处理

有几种方法产生外出的ICMP报文。第8章讲到IP调用icmp_error来产生和发送ICMP 差错报文。icmp_reflect发送ICMP 回答报文，同时，进程也可能通过原始 ICMP 协议生成ICMP 报文。图11-29显示了这些函数与ICMP外出处理之间的关系。

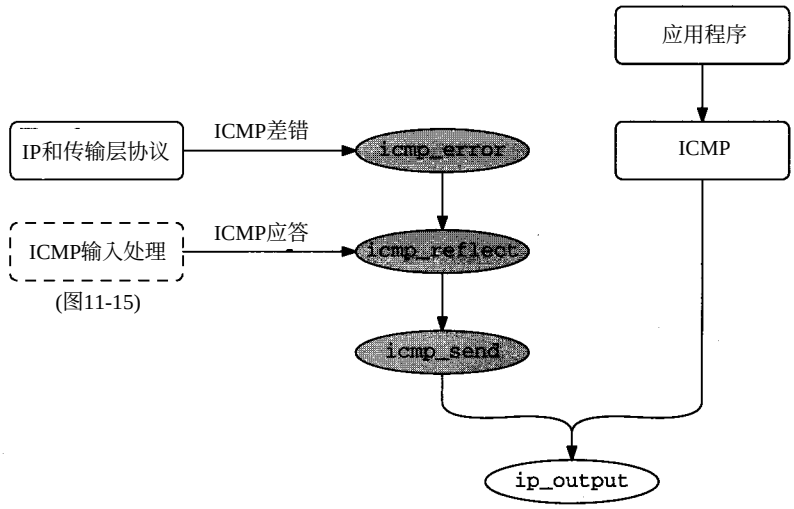

图11-29 ICMP外出处理

11.11 icmp_error函数

icmp_error在IP或运输层协议的请求下，构造一个ICMP差错请求报文，并把它传给icmp_reflect，在那里该报文被返回无效数据报的源站。我们分三部分分析这个函数：

ip_icmp.c

```
46 void
47 icmp_error(n, type, code, dest, destifp)
48 struct mbuf *n;
49 int     type, code;
50 n_long  dest;
51 struct ifnet *destifp;
52 {
53     struct ip *oip = mtod(n, struct ip *), *nip;
54     unsigned oiplen = oip->ip_hl << 2;
55     struct icmp *icp;
56     struct mbuf *m;
57     unsigned icmplen;

58     if (type != ICMP_REDIRECT)
59         icmpstat.icps_error++;
60     /*
61      * Don't send error if not the first fragment of message.
62      * Don't error if the old packet protocol was ICMP
63      * error message, only known informational types.
64      */
65     if (oip->ip_off & ~(IP_MF | IP_DF))
66         goto freeit;
67     if (oip->ip_p == IPPROTO_ICMP && type != ICMP_REDIRECT &&
68         n->m_len >= oiplen + ICMP_MINLEN &&
69         !ICMP_INFOTYPE(((struct icmp *)((caddr_t) oip + oiplen))->icmp_type)){
70         icmpstat.icps_oldicmp++;
71         goto freeit;
72     }
73     /* Don't send error in response to a multicast or broadcast packet */
74     if (n->m_flags & (M_BCAST | M_MCAST))
75         goto freeit;
```

ip_icmp.c

图11-30 icmp_error 函数：验证

- 确认该报文(图11-30)；
- 构造首部(图11-32)；并
- 把原来的数据报包含进来(图11-33)。

46-57 参数是：n，指向包含无效数据报缓存链的指针；type和code，ICMP差错类型和代码；dest，ICMP重定向报文中的下一跳路由器地址；以及destifp，指向原始IP分组外出接口的指针。mtod把缓存链指针n转换成oip，oip是指向缓存中ip结构的指针。原始IP分组的字节长度保存在ioplen中。

58-75 icps_error统计除重定向报文外的所有ICMP差错。Net/3不把重定向报文看作错误，而且icps_error也不是一个SNMP变量。

icmp_error丢弃无效数据报oip，并且在以下情况下，不发送差错报文：

- 除IP_MF和IP_DF外，ip_off的某些位非零(习题11.10)。这表明oip不是数据报的第一个分片，而且ICMP决不能为跟踪数据报的分片而生成差错报文。
- 无效数据报本身是一个ICMP差错报文。如果icmp_type是ICMP请求或响应类型，则ICMP_INFOTYPE返回真；如果icmp_type是一个差错类型，则ICMP_INFOTYPE返回假。

Net/3不考虑ICMP重定向报文差错，尽管RFC 1122要求考虑。

- 数据报作为链路层广播或多播到达(由M_BCAST和M_MCAST标志表明)。

在以下两种其他情况下，不能发送ICMP差错报文：

- 该数据报是发给IP广播和IP多播地址的。
- 数据报的源站地址不是单播IP地址(也即，这个源站地址是一个全零地址、环回地址、广播地址、多播地址或E类地址)。

Net/3无法检查第一种情况。icmp_reflect函数强调了第二种情况(11.12节)。

有趣的是，Net/2的Deering多播扩展并不丢弃第一种类型的数据报。因为Net/3的多播程序来自Deering多播扩展，所以，检测似乎被删去了。

这些限制的目的是为了避免有错的广播数据报触发网络上所有主机都发出ICMP差错报文。当网络上所有主机同时要发送差错报文时，产生的广播风暴会使整个网络的通信崩溃。

这些规则适用于ICMP差错报文，但不适用于ICMP回答。如RFC 1122和RFC 1127的讨论，允许响应广播请求，但既不推荐也不鼓励。Net/3只响应具有单播源地址的广播请求，因为ip_output会把返回到广播地址的ICMP报文丢弃(图11-39)。

图11-31是ICMP差错报文的构造。

图11-32的程序构造差错报文。

76-106 icmp_error以下面的方式构造ICMP差错报文的首部：

- m_gethdr分配一个新的分组首部缓存。MH_ALIGN定位缓存的数据指针，使无效数据报的ICMP首部、IP首部(和选项)和最多8字节的数据被放在缓存的最后。
- icmp_type、icmp_code、icmp_gwaddr(用于重定向)、icmp_pptr(用于参数问题)和icmp_nextmtu(用于要求分片报文)被初始化。icmp_nextmtu字段实现了RFC 1191中描述的要求分片报文的扩展。卷1的24.2节描述的“路径MTU发现算法”依赖于这个报文。

一旦构造好ICMP首部，就必须把原始数据报的一部分附到首部上，如图11-33所示。

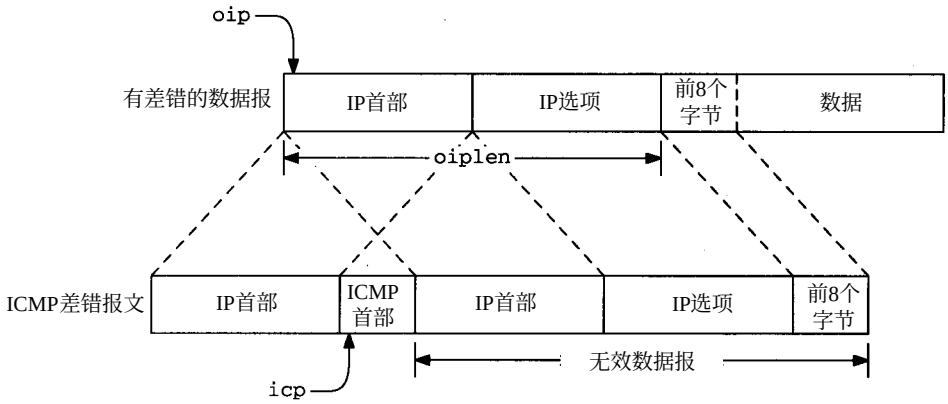

图11-31 ICMP差错报文的构造

ip_icmp.c

```
 76      /*
 77       * First, formulate icmp message
 78       */
 79      m = m_gethdr(M_DONTWAIT, MT_HEADER);
 80      if (m == NULL)
 81          goto freeit;
 82      icmplen = oiplen + min(8, oip->ip_len);
 83      m->m_len = icmplen + ICMP_MINLEN;
 84      MH_ALIGN(m, m->m_len);
 85      icp = mtod(m, struct icmp *);
 86      if ((u_int) type > ICMP_MAXTYPE)
 87          panic("icmp_error");
 88      icmpstat.icps_outhist[type]++;
 89      icp->icmp_type = type;
 90      if (type == ICMP_REDIRECT)
 91          icp->icmp_gwaddr.s_addr = dest;
 92      else {
 93          icp->icmp_void = 0;
 94          /*
 95           * The following assignments assume an overlay with the
 96           * zeroed icmp_void field.
 97           */
 98          if (type == ICMP_PARAMPROB) {
 99              icp->icmp_pptr = code;
100              code = 0;
101          } else if (type == ICMP_UNREACH &&
102                     code == ICMP_UNREACH_NEEDFRAG && destifp) {
103              icp->icmp_nextmtu = htons(destifp->if_mtu);
104          }
105      }
106      icp->icmp_code = code;
```

ip_icmp.c

图11-32 icmp_error 函数：报文首部构造

107-125 无效数据报的IP首部、选项和数据(一共是icmplen个字节)被复制到ICMP差错报文中。同时，首部的长度被加回无效数据报的ip_len中。

在udp_usrreq中，UDP也把首部长度加回到无效数据报的ip_len。其结果是一个ICMP报文，该报文具有无效分组 IP首部内的不正确的数据报长度。作者发现，许多基于Net/2程序的系统都有这个错误，Net/1系统没有这个问题。

ip_icmp.c

```
107     bcopy((caddr_t) oip, (caddr_t) & icp->icmp_ip, icmplen);
108     nip = &icp->icmp_ip;
109     nip->ip_len = htons((u_short) (nip->ip_len + oiplen));

110     /*
111      * Now, copy old ip header (without options)
112      * in front of icmp message.
113      */
114     if (m->m_data - sizeof(struct ip) < m->m_pktdat)
115                 panic("icmp len");
116     m->m_data -= sizeof(struct ip);
117     m->m_len += sizeof(struct ip);
118     m->m_pkthdr.len = m->m_len;
119     m->m_pkthdr.rcvif = n->m_pkthdr.rcvif;
120     nip = mtod(m, struct ip *);
121     bcopy((caddr_t) oip, (caddr_t) nip, sizeof(struct ip));
122     nip->ip_len = m->m_len;
123     nip->ip_hl = sizeof(struct ip) >> 2;
124     nip->ip_p = IPPROTO_ICMP;
125     nip->ip_tos = 0;
126     icmp_reflect(m);

127   freeit:
128     m_freem(n);
129 }
```

ip_icmp.c

图11-33 icmp_error 函数：包含原始数据报

因为MH_ALIGN把ICMP报文分配在缓存的最后，所以缓存的前面应该有足够的空间存放IP首部。无效数据报的IP首部(除选项外)被复制到ICMP报文的前面。

Net/2版本的这部分有一个错误：函数的最后一个 bcopy移动oiplen个字节，其中包括无效数据报的选项。应该只复制没有选项的标准首部。

在恢复正确的数据报长度(ip_len)、首部长度(ip_hl)和协议(ip_p)后，IP首部就完整了。TOS字段(ip_tos)被清除。

RFC 792和RFC 1122推荐在ICMP报文中，把TOS字段设为0。

126-129 完整的报文被传给icmp_reflect，由icmp_reflect把它发回源主机。丢掉无效数据报。

11.12 icmp_reflect函数

icmp_reflect把ICMP回答或差错发回给请求或无效数据报的源站。必须牢记，icmp_reflect在发送数据报之前，把它的源站地址和目的地址倒过来。与 ICMP 报文的源站和目的站地址有关的规则非常复杂，图 11-34对其中几个函数的作用作了小结。

我们分三部分讨论icmp_reflect函数：源站和目的站地址选择、选项构造及组装和发送。图11-35显示了该函数的第一部分。

1. 设置目的地址

329-345 icmp_reflect一开始，就复制 ip_dst，并把请求或差错报文的源站地址ip_src移到ip_dst。icmp_error和icmp_reflect保证：ip_src对差错报文而言是有效的目的地址。ip_output丢掉所有发往广播地址的分组。

函 数	小 结
icmp_input	在地址掩码请求中，用接收接口的广播或目的地址代替全0地址
icmp_error	把作为链路级广播或多播发送的数据报引起的差错报文丢弃。应该丢弃(但没有)发往IP广播或多播地址的数据报引起的报文
icmp_reflect	丢弃报文，而不是把它返回给多播或实验地址 把非单播目的地址转换成接收接口的地址，对返回的报文来说，目的地址就是一个有效的源地址 交换源站和目的站的地址
ip_output	按照ICMP的请求丢弃输出的广播(也就是说，丢弃由发往广播地址的分组产生的差错报文)

图11-34 ICMP丢弃和地址小结

2. 选择源站地址

346-371 icmp_reflect在in_ifaddr中找到具有单播或广播地址的接口，该接口地址与原始数据报的目的地址匹配，这样，icmp_reflect就为报文选好了源地址。在多接口主机上，匹配的接口可能不是接收该数据报的接口。如果没有匹配，就选择正在接收的接口的in_ifaddr结构，或者in_ifaddr中的第一个地址(如果该接口没有被配置成IP可用的)。该函数把ip_src设成所选的地址，并把ip_ttl改为255(MAXTTL)，因为这是一个新的数据报。

RFC 1700推荐把所有IP分组的TTL字段设成64。但是现在，许多系统把ICMP报文的TTL设成255。

TTL的取值有一个拆衷。小的TTL避免分组在路由回路里面循环，但也有可能使分组无法到达远一点的节点(有很多跳)。大的TTL允许分组到达远距离的主机，但却让分组在路由回路里循环较长时间。

ip_icmp.c

```
329 void
330 icmp_reflect(m)
331 struct mbuf *m;
332 {
333     struct ip *ip = mtod(m, struct ip *);
334     struct in_ifaddr *ia;
335     struct in_addr t;
336     struct mbuf *opts = 0, *ip_srcroute();
337     int     optlen = (ip->ip_hl << 2) - sizeof(struct ip);

338     if (!in_canforward(ip->ip_src) &&
339         ((ntohl(ip->ip_src.s_addr) & IN_CLASSA_NET) !=
340          (IN_LOOPBACKNET << IN_CLASSA_NSHIFT))) {
341         m_freem(m);             /* Bad return address */
342         goto done;              /* Ip_output() will check for broadcast */
343     }
344     t = ip->ip_dst;
345     ip->ip_dst = ip->ip_src;
346     /*
347      * If the incoming packet was addressed directly to us,
348      * use dst as the src for the reply.  Otherwise (broadcast
349      * or anonymous), use the address which corresponds
350      * to the incoming interface.
351      */
352     for (ia = in_ifaddr; ia; ia = ia->ia_next) {
```

图11-35 icmp_reflect 函数：地址选择

```
353         if (t.s_addr == IA_SIN(ia)->sin_addr.s_addr)
354             break;
355         if ((ia->ia_ifp->if_flags & IFF_BROADCAST) &&
356             t.s_addr == satosin(&ia->ia_broadaddr)->sin_addr.s_addr)
357             break;
358     }
359     icmpdst.sin_addr = t;
360     if (ia == (struct in_ifaddr *) 0)
361         ia = (struct in_ifaddr *) ifaof_ifpforaddr(
362                         (struct sockaddr *) &icmpdst, m->m_pkthdr.rcvif);
363     /*
364      * The following happens if the packet was not addressed to us,
365      * and was received on an interface with no IP address.
366      */
367     if (ia == (struct in_ifaddr *) 0)
368         ia = in_ifaddr;
369     t = IA_SIN(ia)->sin_addr;
370     ip->ip_src = t;
371     ip->ip_ttl = MAXTTL;
```

ip_icmp.c

图11-35 (续)

RFC 1122提出，对到达的回显请求或时间戳请求，要求把其中的源路由选项及记录路由和时间戳选项的建议，附到回答报文中。在这个过程期间，源路由必须被逆转过来。 RFC 1122没有涉及在其他ICMP回答报文中如何处理这些选项。 Net/3把这些规则应用于地址掩码请求，因为它在构造地址掩码回答后调用了`icmp_reflect`(图11-21)。

程序的下一部分(图11-36)为ICMP报文构造选项。

ip_icmp.c

```
372     if (optlen > 0) {
373         u_char *cp;
374         int     opt, cnt;
375         u_int   len;
376         /*
377          * Retrieve any source routing from the incoming packet;
378          * add on any record-route or timestamp options.
379          */
380         cp = (u_char *) (ip + 1);
381         if ((opts = ip_srcroute()) == 0 &&
382             (opts = m_gethdr(M_DONTWAIT, MT_HEADER))) {
383             opts->m_len = sizeof(struct in_addr);
384             mtod(opts, struct in_addr *)->s_addr = 0;
385         }
386         if (opts) {
387             for (cnt = optlen; cnt > 0; cnt -= len, cp += len) {
388                 opt = cp[IPOPT_OPTVAL];
389                 if (opt == IPOPT_EOL)
390                     break;
391                 if (opt == IPOPT_NOP)
392                     len = 1;
393                 else {
394                     len = cp[IPOPT_OLEN];
395                     if (len <= 0 || len > cnt)
396                         break;
397                 }
398                 /*
```

图11-36 `icmp_reflect` 函数：选项构造

```
399                  * Should check for overflow, but it "can't happen"
400                  */
401                 if (opt == IPOPT_RR || opt == IPOPT_TS ||
402                     opt == IPOPT_SECURITY) {
403                     bcopy((caddr_t) cp,
404                           mtod(opts, caddr_t) + opts->m_len, len);
405                     opts->m_len += len;
406                 }
407             }
408             /* Terminate & pad, if necessary */
409             if (cnt = opts->m_len % 4) {
410                 for (; cnt < 4; cnt++) {
411                     *(mtod(opts, caddr_t) + opts->m_len) =
412                         IPOPT_EOL;
413                     opts->m_len++;
414                 }
415             }
416         }
```

ip_icmp.c

图11-36 (续)

3. 取得逆转后的源路由

372-385 如果到达的数据报没有选项，控制被传给 430行(图11-37)。icmp_error传给icmp_reflect的差错报文从来没有IP选项，所以后面的程序只用于那些被转换成回答并直接传给icmp_reflect的ICMP请求。

cp指向回答的选项的开始。ip_srcroute逆转并返回所有在ipintr处理数据报时保存下来的源路由选项。如果ip_srcoute返回0，即请求中没有源路由选项，icmp_reflect分配并初始化一个mbuf，作为空的ipoption结构。

4. 加上记录路由和时间戳选项

386-416 如果opts指向某个缓存，for循环搜索原始IP首部的选项，在ip_srcroute返回的源路由后面加上记录路由和时间戳选项。

在ICMP报文发送之前必须移走原始首部里的选项。这由图 11-37中的程序完成。

ip_icmp.c

```
417         /*
418          * Now strip out original options by copying rest of first
419          * mbuf's data back, and adjust the IP length.
420          */
421         ip->ip_len -= optlen;
422         ip->ip_hl = sizeof(struct ip) >> 2;
423         m->m_len -= optlen;
424         if (m->m_flags & M_PKTHDR)
425             m->m_pkthdr.len -= optlen;
426         optlen += sizeof(struct ip);
427         bcopy((caddr_t) ip + optlen, (caddr_t) (ip + 1),
428               (unsigned) (m->m_len - sizeof(struct ip)));
429     }
430     m->m_flags &= ~(M_BCAST | M_MCAST);
431     icmp_send(m, opts);
432 done:
433     if (opts)
434         (void) m_free(opts);
435 }
```

ip_icmp.c

图11-37 icmp_reflect 函数：最后的组装

5. 移走原始选项

417-429 icmp_reflect把ICMP报文移到IP首部的后面，这样就从原始请求中移走了选项。如图11-38所示。新选项在opts所指向的mbuf里，被ip_output再次插入。

6. 发送报文和清除

430-435 在报文和选项被传给icmp_send之前，要明确地清除广播和多播标志位。此后释放掉存放选项的缓存。

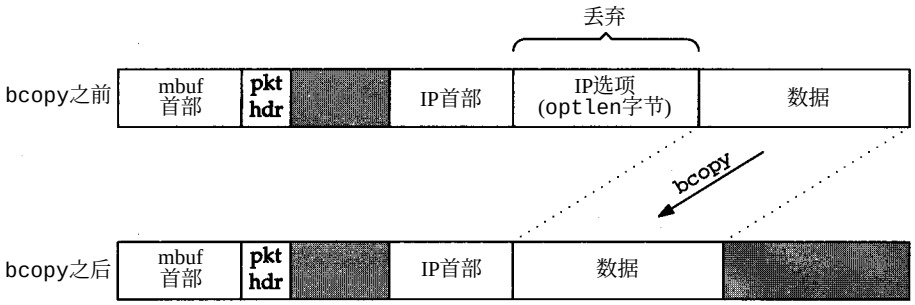

图11-38 icmp_reflect ：移走选项

11.13 icmp_send函数

icmp_send(图11-39)处理所有输出的ICMP报文，并在把它们传给IP层之前计算ICMP检验和。

ip_icmp.c

```
440 void
441 icmp_send(m, opts)
442 struct mbuf *m;
443 struct mbuf *opts;
444 {
445     struct ip *ip = mtod(m, struct ip *);
446     int     hlen;
447     struct icmp *icp;

448     hlen = ip->ip_hl << 2;
449     m->m_data += hlen;
450     m->m_len -= hlen;
451     icp = mtod(m, struct icmp *);
452     icp->icmp_cksum = 0;
453     icp->icmp_cksum = in_cksum(m, ip->ip_len - hlen);
454     m->m_data -= hlen;
455     m->m_len += hlen;
456     (void) ip_output(m, opts, NULL, 0, NULL);
457 }
```

ip_icmp.c

图11-39 icmp_send 函数

440-457 与icmp_input检测ICMP检验和一样， Net/3调整缓存的数据指针和长度，隐藏IP首部，让in_cksum只看到ICMP报文。计算好的检验和放在首部的 icmp_cksum，然后把数据报和所有选项传给ip_output。ICMP层并不维护路由高速缓存，所以icmp_send只传给ip_output一个空指针(第4个参数)，而不是控制标志。特别是不传 IP_ALLOWBROADCAST，所以ip_output丢弃所有具有广播目的地址的 ICMP报文(也就是说，到达原始数据报的具有无效的源地址)。

11.14 icmp_sysctl函数

IP的icmp_sysctl函数只支持图11-40列出的选项。系统管理员可以用sysctl程序修改该选项。

Sysctl常量	Net/3变量	描　述
ICMPCTL_MASKREPL	icmpmaskrepl	系统是否响应ICMP地址掩码请求

图11-40 icmp_sysctl 参数

图11-41显示了icmp_sysctl函数。

ip_icmp.c

```
467 int
468 icmp_sysctl(name, namelen, oldp, oldlenp, newp, newlen)
469 int    *name;
470 u_int   namelen;
471 void   *oldp;
472 size_t *oldlenp;
473 void   *newp;
474 size_t  newlen;
475 {

476     /* All sysctl names at this level are terminal. */
477     if (namelen != 1)
478         return (ENOTDIR);

479     switch (name[0]) {
480     case ICMPCTL_MASKREPL:
481         return (sysctl_int(oldp, oldlenp, newp, newlen, &icmpmaskrepl));
482     default:
483         return (ENOPROTOOPT);
484     }
485     /* NOTREACHED */
486 }
```

ip_icmp.c

图11-41 icmp_sysctl 函数

467-478 如果缺少所要求的ICMP sysctl名，就返回ENOTDIR。

479-486 ICMP级以下没有选项，所以，如果不识别选项，该函数就调用sysctl_int修改icmpmaskrepl或返回ENOPROTOOPT。

11.15 小结

ICMP协议是作为IP上面的运输层实现的，但它与IP层紧密结合一起。我们看到，内核直接响应ICMP请求报文，但把差错与回答传给合适的运输层协议或应用程序处理。当一个ICMP重定向报文到达时，内核立刻重定向表，并且也把重定向传给所有等待的进程，比如典型地传给一个路由守护程序。

我们将在23.9和27.6节看到UDP和TCP协议如何响应ICMP差错报文，在第32章看到进程如何产生ICMP请求。

习题

11.1 一个目的地址是0.0.0.0的请求所产生的ICMP地址掩码回答报文的源地址是什么？

11.2 试描述一个具有假的单播源地址的分组在链路级的广播会如何影响网络上另一个主机的运行。

11.3 RFC 1122建议，如果新的第一跳路由器与旧的第一跳路由器位于不同的子网，或者如果发送报文的路由器不是报文最终目的地的当前第一跳路由器，那么主机应该丢弃ICMP重定向报文。为什么要采纳这个建议？

11.4 如果ICMP信息请求是过时的，为什么`icmp_inout`要把它传给`rip_input`而不是丢弃它呢？

11.5 我们指出，Net/3在把IP分组放入一个ICMP差错报文之前，并不把它的偏移和长度字段转换成网络字节序。为什么这对IP位移字段来说是无关紧要的？

11.6 描述某种情况，使图11-25的`ifaof_ifpforaddr`返回一个空指针。

11.7 在一次时间戳询问中，时间戳后面的数据会怎么样？

11.8 实现以下改变，改进ICMP时间戳程序：
在缓存分组首部加上一个时间戳字段，让设备驱动程序把接收分组的确切时间记录在这个字段内，并用ICMP 时间戳程序把该值复制到`icmp_rtime`字段。
在输出端，让ICMP时间戳程序保存分组中的某个字节偏移，该位置用于保存时间戳里的当前时间。修改设备驱动程序，在发送分组之前插入时间戳。

11.9 修改`icmp_error`，使ICMP差错报文中返回最多64字节(像Solaris 2.x一样)的原始数据。

11.10 图11-30中，`ip_off`的高位被置位的分组会发生什么情况？

11.11 为什么图11-39中丢弃了`ip_output`返回的值？