

# 第9章 IP选项处理

## 9.1 引言

第8章中提到，IP输入函数(ipintr)将在验证分组格式(检验和，长度等)之后，确定分组是否到达目的地之前，对选项进行处理。这表明，分组所遇到的每个路由器以及最终的目的主机都要对分组的选项进行处理。

RFC 791和1122指定了IP选项和处理规则。本章将讨论大多数 IP选项的格式和处理。我们也将显示运输协议如何指定 IP数据报内的IP选项。

IP分组内可以包含某些在分组被转发或被接收之前处理的可选字段。 IP实现可以用任意顺序处理选项；Net/3按照选项在分组中出现的顺序处理选项。图 9-1显示，标准IP首部之后最多可跟40字节的选项。

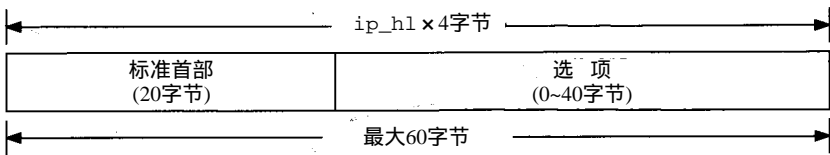


图9-1 一个IP首部可以有0~40字节的IP选项

## 9.2 代码介绍

两个首部描述了IP选项的数据结构。选项处理的代码出现在两个 C文件中。图 9-2列出了相关的文件。

文 件	描 述
netinet/ip.h	ip_timestamp结构
netinet/ip_var.h	ipoption结构
netinet/ip_input.c	选项处理
netinet/ip_output.c	ip_insertoptions函数

图9-2 本章讨论的文件

### 9.2.1 全局变量

图9-3描述了两个全局变量支持源路由的逆 (reversal)。

变 量	数据类型	描 述
ip_nhops	int	以前的源路由跳计数
ip_srcrt	struct ip_srcrt	以前的源路由

图9-3 本章引入的全局变量

### 9.2.2 统计量

选项处理代码更新的唯一的统计量是ipstat结构中的ips\_badoptioins，如图8-4所示。

### 9.3 选项格式

IP选项字段可能包含0个或多个单独选项。选项有两种类型，单字节和多字节，如图9-4中所示。

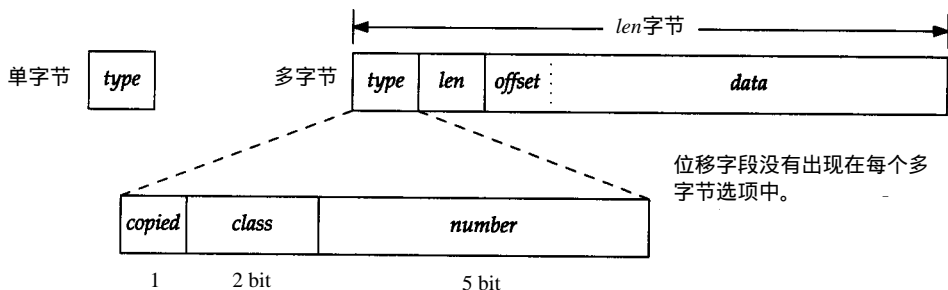


图9-4 单字节和多字节IP选项的结构

所有选项都以1字节类型(*type*)字段开始。在多字节选项中，类型字段后面紧接着一个长度(*len*)字段，其他的字节是数据(*data*)。许多选项数据字段的第一个字节是1字节的位移(*offset*)字段，指向数据字段内的某个字节。长度字节的计算覆盖了类型、长度和数据字段。类型被继续分成三个子字段：1 bit 备份(*copied*)标志、2 bit 类(*class*)字段和5 bit 数字(*number*)字段。图9-5列出了目前定义的IP选项。前两个选项是单字节选项；其他的是多字节选项。

常 量	类 型		长度 (字节)	Net/3	描 述
	十进制	二进制			
<i>IPOPT_EOL</i>	0-0-0 0	0-00-00000	1	•	选项表的结尾(EOL)
<i>IPOPT_NOP</i>	0-0-1 1	0-00-00001	1	•	无操作(NOP)
<i>IPOPT_RR</i>	0-0-7 7	0-00-00111	可变	•	记录路由
<i>IPOPT_TS</i>	0-2-4 68	0-10-00100	可变	•	时戳
<i>IPOPT_SECURITY</i>	1-0-2 130	1-00-00010	11	•	基本的安全
<i>IPOPT_LSRR</i>	1-0-3 131	1-00-00011	可变	•	宽松源路由和记录路由(LSRR)
	1-0-5 133	1-00-00101	可变	•	扩展的安全
<i>IPOPT_SATID</i>	1-0-8 136	1-00-01000	4	•	流标识符
<i>IPOPT_SSRR</i>	1-0-9 137	1-00-01001	可变	•	严格源路由和记录路由(SSRR)

图9-5 RFC 791定义的IP选项

第1列显示了Net/3的选项常量，第2列和第3列是该类型的十进制和二进制值，第4列是选项的长度。Net/3列显示的是在Net/3中由ip\_dooptions实现的选项。IP必须自动忽略所有它不识别的选项。我们不描述Net/3没有实现的选项：安全和流ID。流ID选项是过时的，安全选项主要只由美国军方使用。RFC 791中有更多的讨论。

当Net/3对一个有选项的分组进行分片时(10.4节)，它将检查*copied*标志位。该标志位指出是否把所有选项都备份到每个分片的IP首部。*class*字段把相关的

<i>class</i>	描 述
0	控制
1	保留
2	查错和措施
3	保留

图9-6 IP选项内的  
*class*字段

选项按如图9-6所示进行分组。图9-5中，除时戳选项具有`class`为2外，所有选项都是`class`为0。

## 9.4 ip\_dooptions函数

在图8-13中，我们看到`ipintr`在检测分组的目的地址之前调用`ip_dooptions`。`ip_dooptions`被传给一个指针`m`，该指针指向某个分组，`ip_dooptions`处理分组中它知道的选项。如果`ip_dooptions`转发该分组，如在处理LSRR和SSRR选项时，或由于某个差错而丢掉该分组时，它返回1。如果它不转发分组，`ip_dooptions`返回0，由`ipintr`继续处理该分组。

`ip_dooptions`是一个长函数，所以我们分步地显示。第一部分初始化一个`for`循环，处理首部中的各选项。

当处理每个选项时，`cp`指向选项的第一个字节。图9-7显示，当可用时，如何从`cp`的常量位移访问`type`、`length`和`offset`字段。

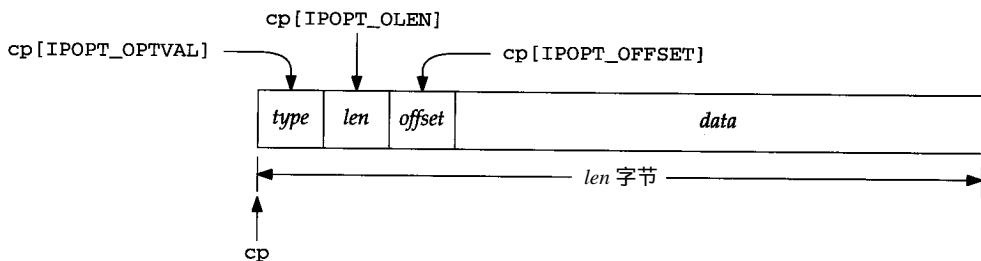


图9-7 用常量位移访问IP选项字段

RFC把位移(`offset`)字段描述作指针(`pointer`)，指针比位移的描述性略强一些。`offset`的值是某个字节在该选项内的序号(从`type`开始，序号为1)，不是从`type`开始的、且以零开始的计数。位移的最小值是4(`IPOPT_MINOFF`)，它指向的是多字节选项中数据字段的第一个字节。

图9-8显示了`ip_dooptions`函数的整体结构。

555-566 `ip_dooptions`把ICMP差错类型`type`初始化为`ICMP_PARAMPROB`，对任何没有特定差错类型的差错，这是一个一般值。对于`ICMP_PARAMPROB`，`code`指的是出错字节在分组内的位移。这是默认的ICMP差错报文。某些选项将改变这些值。

`ip`指向一个20字节大小的`ip`结构，所以`ip+1`指向的是跟在IP首部后面的下一个`ip`结构。因为`ip_dooptions`需要IP首部后面字节的地址，所以就把结果指针转换成指向一个无符号字节(`u_char`)的指针。因此，`cp`指向标准IP首部以外的第一个字节，就是IP选项的第一个字节。

### 1. EOL和NOP过程

567-582 `for`循环按照每个选项在分组中出现的顺序分别对它们进行处理。EOL选项以及一个无效的选项长度(也即选项长度表明选项数据超过了IP首部)都将终止该循环。当出现NOP选项时，忽略它。`switch`语句的`default`情况隐含要求系统忽略未知的选项。

下面的内容描述了`switch`语句处理的每个选项。如果`ip_dooptions`在处理分组选项时没有出错，就把控制交给`switch`下面的代码。

### 2. 源路由转发

719-724 如果分组需要被转发，SSRR或LSRR选项处理代码就把forward置位。分组被传给ip\_forward，并且第2个参数为1，表明分组是按源路由选择的。

```

553 int
554 ip_dooptions(m)
555 struct mbuf *m;
556 {
557     struct ip *ip = mtod(m, struct ip *);
558     u_char *cp;
559     struct ip_timestamp *ipt;
560     struct in_ifaddr *ia;
561     int opt, optlen, cnt, off, code, type = ICMP_PARAMPROB, forward = 0;
562     struct in_addr *sin, dst;
563     n_time ntime;

564     dst = ip->ip_dst;
565     cp = (u_char *) (ip + 1);
566     cnt = (ip->ip_hl << 2) - sizeof(struct ip);
567     for (; cnt > 0; cnt -= optlen, cp += optlen) {
568         opt = cp[IPOPT_OPTVAL];
569         if (opt == IPOPT_EOL)
570             break;
571         if (opt == IPOPT_NOP)
572             optlen = 1;
573         else {
574             optlen = cp[IPOPT_OLEN];
575             if (optlen <= 0 || optlen > cnt) {
576                 code = &cp[IPOPT_OLEN] - (u_char *) ip;
577                 goto bad;
578             }
579         }
580         switch (opt) {
581             default:
582                 break;

583             /* option processing */

584         }
585     }
586     if (forward) {
587         ip_forward(m, 1);
588         return (1);
589     }
590     return (0);

591 bad:
592     ip->ip_len -= ip->ip_hl << 2; /* XXX icmp_error adds in hdr length */
593     icmp_error(m, type, code, 0, 0);
594     ipstat.ips_badoptions++;
595     return (1);
596 }

```

ip\_input.c

图9-8 ip\_dooptions 函数

我们在8.5节中讲到，并不为源路由选择分组生成ICMP重定向——这就是为什么在传给ip\_forward时设置第2个参数的原因。

如果转发了分组，则 `ip_dooptions` 返回1。如果分组中没有源路由，则返回 0 给 `ipintr`，表明需要对该数据报进一步处理。注意，只有当系统被配置成路由器时 (`ipforwarding` 等于1)，才发生源路由转发。

从某种程度上说，这是一个有些矛盾的策略，但却是 RFC 1122 的书面要求。

RFC 1127 [Braden 1989c] 把它作为一个公开问题加以阐述。

### 3. 差错处理

725-730 如果在 `switch` 语句里出现了错误，`ip_dooptions` 就跳到 `bad`。从分组长度中把IP首部长度减去，因为 `icmp_error` 假设首部长度不包含在分组长度里。`icmp_error` 发出适当的差错报文，`ip_dooptions` 返回1，避免 `ipintr` 处理被丢弃的分组。

下一节描述Net/3处理的所有选项。

## 9.5 记录路由选项

记录路由选项使得分组在穿过互联网时所经过的路由被记录在分组内部。项的大小是源主机在构造时确定的，必须足够保存所有预期的地址。我们记得在 IP 分组的首部，选项最多只能有40字节。记录路由选项可以有3个字节的开销，后面紧跟地址的列表（每个地址4字节）。如果该选项是唯一的选项，则最多可以有9个 ( $3 + 4 \times 9 = 39$ ) 地址出现。一旦分配给该选项的空间被填满，就按通常的情况对分组进行转发，中间的系统就不再记录地址。

图9-9说明了一个记录路由选项的格式，图9-10是其源程序。

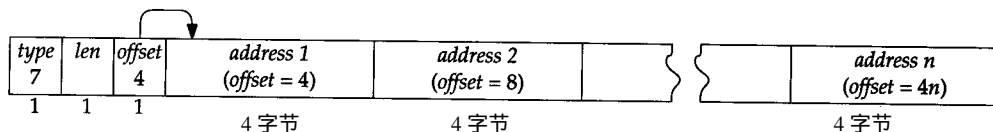


图9-9 记录路由选项，其中  $n$  必须  $\leq 9$

```

647      case IPOPT_RR:
648          if ((off = cp[IPOPT_OFFSET]) < IPOPT_MINOFF) {
649              code = &cp[IPOPT_OFFSET] - (u_char *) ip;
650              goto bad;
651          }
652          /*
653           * If no space remains, ignore.
654           */
655          off--;
656          if (off > optlen - sizeof(struct in_addr))
657              break;
658          bcopy((caddr_t) (&ip->ip_dst), (caddr_t) &ipaddr.sin_addr,
659              sizeof(ipaddr.sin_addr));
660          /*
661           * locate outgoing interface; if we're the destination,
662           * use the incoming interface (should be same).
663           */
664          if ((ia = (INA) ifa_ifwithaddr((SA) &ipaddr)) == 0 &&
665              (ia = ip_rtaddr(ipaddr.sin_addr)) == 0) {
666              type = ICMP_UNREACH;
667              code = ICMP_UNREACH_HOST;

```

图9-10 函数 `ip_dooptions`：记录路由选项的处理

```

668             goto bad;
669         }
670         bcopy((caddr_t) & (IA_SIN(ia)->sin_addr),
671             (caddr_t) (cp + off), sizeof(struct in_addr));
672         cp[IPOPT_OFFSET] += sizeof(struct in_addr);
673         break;

```

ip\_input.c

图9-10 (续)

647-657 如果位移选项太小, 则 ip\_dooptions 就发送一个 ICMP 参数问题差错。如果变量 code 被设置成分组内无效选项的字节位移量, 并且 bad 标号(图9-8)语句的执行产生错误, 则发出的 ICMP 参数问题差错报文中就具有该 code 值。如果选项中没有附加地址的空间, 则忽略该选项, 并继续处理下一个选项。

#### 记录地址

658-673 如果 ip\_dst 是某个系统地址(分组已到达目的地), 则把接收接口的地址记录在选项中; 否则把 ip\_rtaddr 提供的外出接口的地址记录下来。把位移更新为选项中下一个可用地址位置。如果 ip\_rtaddr 无法找到到目的地的路由, 就发送一个 ICMP 主机不可达差错报文。

卷1的7.3节举了一些记录路由选项的例子。

#### ip\_rtaddr 函数

函数 ip\_rtaddr 查询路由缓存, 必要时查询完整的路由表, 来找到到给定 IP 地址的路由。它返回一个指向 in\_ifaddr 结构的指针, 该指针与该路由的外出接口有关。图 9-11 显示了该函数。

```

735 struct in_ifaddr *
736 ip_rtaddr(dst)
737 struct in_addr dst;
738 {
739     struct sockaddr_in *sin;
740     sin = (struct sockaddr_in *) &ipforward_rt.ro_dst;
741     if (ipforward_rt.ro_rt == 0 || dst.s_addr != sin->sin_addr.s_addr) {
742         if (ipforward_rt.ro_rt) {
743             RTFREE(ipforward_rt.ro_rt);
744             ipforward_rt.ro_rt = 0;
745         }
746         sin->sin_family = AF_INET;
747         sin->sin_len = sizeof(*sin);
748         sin->sin_addr = dst;
749         rtalloc(&ipforward_rt);
750     }
751     if (ipforward_rt.ro_rt == 0)
752         return ((struct in_ifaddr *) 0);
753     return ((struct in_ifaddr *) ipforward_rt.ro_rt->rt_ifa);
754 }

```

ip\_input.c

图9-11 函数 ip\_rtaddr : 寻找外出的接口

#### 1. 检查IP转发缓存

735-741 如果路由缓存为空, 或者如果 ip\_rtaddr 的唯一参数 dest 与路由缓存中的目的

地不匹配，则必须查询路由表选择一个外出的接口。

## 2. 确定路由

742-750 旧的路由(如果有的话)被丢弃，并把新的路由储存在 \*sin(这是转发缓存的 ro\_dst成员)。rtalloc搜索路由表，寻找到目的地的路由。

## 3. 返回路由信息

751-754 如果没有路由可用，就返回一个空指针；否则，就返回一个指针，指向与所选路由相关联的接口地址结构。

## 9.6 源站和记录路由选项

通常是在中间路由器所选择的路径上转发分组。源站和记录路由选项允许源站明确指定一条到目的地的路由，覆盖掉中间路由器的路由选择决定。而且，在分组到达目的地的过程中，把该路由记录下来。

严格路由包含了源站和目的站之间的每个中间路由器的地址；宽松路由只指定某些中间路由器的地址。在宽松路由中，路由器可以自由选择两个系统之间的任何路径；而在严格路由中，则不允许路由器这样做。我们用图 9-12 说明源路由处理。

A、B和C是路由器，而HS和HD是源和目的主机。因为每个接口都有自己的 IP地址，所以我们看到路由器A有三个地址： $A_1$ 、 $A_2$ 和 $A_3$ 。同样，路由器B和C也有多个地址。图 9-13 显示了源站和记录路由选项的格式。

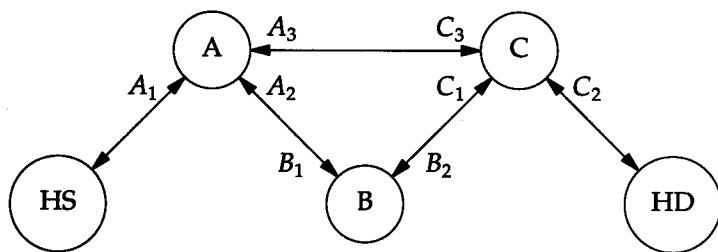


图9-12 源路由举例

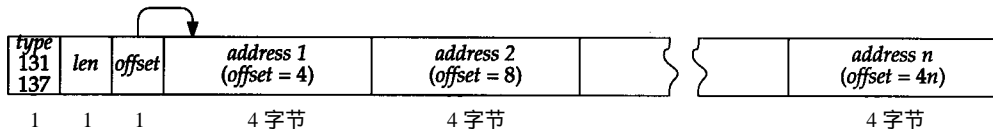


图9-13 严格和宽松源路由选项

IP首部的源和目的地址以及在选项中列出的位移和地址表，指定了路由以及分组目前在该路由中所处的位置。图 9-14 显示，当分组按照这个宽松源路由从 HS 经 A、B、C 到 HD 时，信息是如何改变的。每行代表当分组被第 1 列显示的系统发送时的状态。最后一行显示分组被 HD 接收。图 9-15 显示了相关的代码。

符号“·”表示位移与路由中地址的相对位置。注意，每个系统都把出接口的地址放到选项去。特别地，原来的路由指定  $A_3$  为第一跳目的地，但是外出接口  $A_2$  被记录在路由中。通过这种方法，分组所采用的路由被记录在选项中。被记录的路由将被目的地系统倒转过来放到所有应答分组上，让它们沿着原始的路由的逆方向发送。

除了 UDP，Net/3 在应答时总是把收到的源路由逆转过来。

系统	IP首部		源路由选项		
	ip_src	ip_dst	位移	地址	
HS	HS	A <sub>3</sub>	4	• B <sub>1</sub>	C <sub>1</sub> HD
A	HS	B <sub>1</sub>	8	A <sub>2</sub> • C <sub>1</sub>	HD
B	HS	C <sub>1</sub>	12	A <sub>2</sub> B <sub>2</sub> • HD	
C	HS	HD	16	A <sub>2</sub> B <sub>2</sub> C <sub>2</sub> •	
HD	HS	HD	16	A <sub>2</sub> B <sub>2</sub> C <sub>2</sub> •	

图9-14 当分组通过该路由时，源路由选项被修改。

```

583      /*
584      * Source routing with record.
585      * Find interface with current destination address.
586      * If none on this machine then drop if strictly routed,
587      * or do nothing if loosely routed.
588      * Record interface address and bring up next address
589      * component. If strictly routed make sure next
590      * address is on directly accessible net.
591      */
592      case IPOPT_LSRR:
593      case IPOPT_SSRR:
594          if ((off = cp[IPOPT_OFFSET]) < IPOPT_MINOFF) {
595              code = &cp[IPOPT_OFFSET] - (u_char *) ip;
596              goto bad;
597          }
598          ipaddr.sin_addr = ip->ip_dst;
599          ia = (struct in_ifaddr *)
600              ifa_ifwithaddr((struct sockaddr *) &ipaddr);
601          if (ia == 0) {
602              if (opt == IPOPT_SSRR) {
603                  type = ICMP_UNREACH;
604                  code = ICMP_UNREACH_SRCFAIL;
605                  goto bad;
606              }
607              /*
608               * Loose routing, and not at next destination
609               * yet; nothing to do except forward.
610               */
611              break;
612          }
613          off--; /* 0 origin */
614          if (off > optlen - sizeof(struct in_addr)) {
615              /*
616               * End of source route. Should be for us.
617               */
618              save_rte(cp, ip->ip_src);
619              break;
620          }
621          /*
622           * locate outgoing interface
623           */
624          bcopy((caddr_t) (cp + off), (caddr_t) &ipaddr.sin_addr,
625              sizeof(ipaddr.sin_addr));
626          if (opt == IPOPT_SSRR) {
627 #define INA struct in_ifaddr *
628 #define SA struct sockaddr *
629             if ((ia = (INA) ifa_ifwithdstaddr((SA) &ipaddr)) == 0)

```

图9-15 函数ip\_dooptions : LSRR和SSRR选项处理



```

630         ia = (INA) ifa_ifwithnet((SA) & ipaddr);
631     } else
632         ia = ip_rtaddr(ipaddr.sin_addr);
633     if (ia == 0) {
634         type = ICMP_UNREACH;
635         code = ICMP_UNREACH_SRCFAIL;
636         goto bad;
637     }
638     ip->ip_dst = ipaddr.sin_addr;
639     bcopy((caddr_t) & (IA_SIN(ia)->sin_addr),
640          (caddr_t) (cp + off), sizeof(struct in_addr));
641     cp[IPOPT_OFFSET] += sizeof(struct in_addr);
642     /*
643      * Let ip_intr's mcast routing check handle mcast pkts
644      */
645     forward = !IN_MULTICAST(ntohl(ip->ip_dst.s_addr));
646     break;

```

ip\_input.c

图9-15 (续)

583-612 如果选项位移小于4(IPOPT\_MINOFF),则Net/3发送一个ICMP参数问题差错,并带上相应的code值。如果分组的目的地址与本地地址没有一个匹配,且选项是严格源路由(IPOPT\_SSRR),则发送一个源路由失败差错。如果本地地址不在路由中,则上一个系统把分组发送到错误的主机上了。对宽松路由(IPOPT\_LSRR)来说,这不是错误;仅意味着IP必须把分组转发到目的地。

#### 1. 源路由的结束

613-620 减小off,把它转换成从选项开始的字节位移。如果IP首部的ip\_dst是某个本地地址,并且off所指向的超过了源路由的末尾,源路由中没有地址了,则分组已经到达了目的地。save\_rte复制在静态结构ip\_srcrt中的路由,并保存在全局ip\_nhops(图9-18)里路由中的地址个数。

ip\_srcrt被定义成为一个外部静态结构,因为它只能被在ip\_input.c中定义的函数访问。

#### 2. 为下一跳更新分组

621-637 如果ip\_dst是一个本地地址,并且offset指向选项内的一个地址,则该系统是源路由中指定的一个中间系统,分组也没有到达目的地。在严格路由中,下一个系统必须位于某个直接相连的网络上。ifa\_ifwithdst和ifa\_ifwithnet通过在配置的接口中搜索匹配的目的地(一个点到点的接口)或匹配的网络地址(广播接口)来寻找一条到下一个系统的路由。而在宽松路由中,ip\_rtaddr(图9-11)通过查询路由表来寻找到下一个系统的路由。如果没有找到到下一系统的接口或路由,就发送一个ICMP源路由失败差错报文。

638-644 如果找到一个接口或一条路由,则ip\_dooptions把ip\_dst设置成off指向的IP地址。在源路由选项内,用外出接口的地址代替中间系统的地址,把位移增加,指向路由中的下一个地址。

#### 3. 多播目的地

645-646 如果新的目的地址不是多播地址,就将forward设置成1,表明在处理完所有选项后,应该把分组转发而不是返回给ipintr。

源路由中的多播地址允许两个多播路由器通过不支持多播的中间路由器进行通信。第14章详细描述了这一技术。

卷I的8.5节有更多的源路由选项的例子。

### 9.6.1 save\_rte函数

RFC 1122要求, 在最终目的地, 运输协议必须能够使用分组中被记录下来的路由。运输协议必须把该路由倒过来并附在所有应答的分组上。图 9-18中显示的save\_rte函数, 把源路由保存在如图9-16所示的ip\_srcrt结构中。

```

57 int      ip_nhops = 0;
58 static struct ip_srcrt {
59     struct in_addr dst;          /* final destination */
60     char      nop;               /* one NOP to align */
61     char      srcopt[IPOPT_OFFSET + 1]; /* OPTVAL, OLEN and OFFSET */
62     struct in_addr route[MAX_IPOPTLEN / sizeof(struct in_addr)];
63 } ip_srcrt;

```

ip\_input.c

图9-16 结构ip\_srcrt

Route的声明是不正确的, 尽管这不是个恶性错误。应该是

```
Struct in_addr route[(MAX_IPOPTLEN - 3)/sizeof(struct in_addr)];
```

对图9-26和图9-27的讨论详细地说明了这个问题。

57-63 该代码定义了ip\_srcrt结构,并声明了静态变量ip\_srcrt。只有两个函数访问ip\_srcrt: save\_rte, 把到达分组中的源路由复制到ip\_srcrt中; ip\_srcroute, 创建一个与源路由方向相逆的路由。图 9-17说明了源路由处理过程。

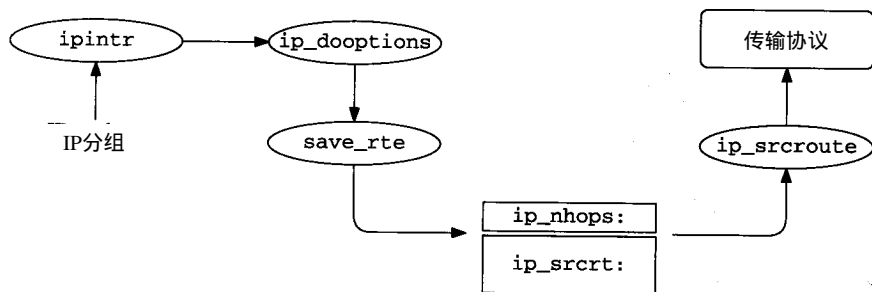


图9-17 对求逆后的源路由的处理

```

759 void
760 save_rte(option, dst)
761 u_char *option;
762 struct in_addr dst;
763 {
764     unsigned olen;
765     olen = option[IPOPT_OLEN];
766     if (olen > sizeof(ip_srcrt) - (1 + sizeof(dst)))
767         return;
768     bcopy((caddr_t) option, (caddr_t) ip_srcrt.srcopt, olen);
769     ip_nhops = (olen - IPOPT_OFFSET - 1) / sizeof(struct in_addr);
770     ip_srcrt.dst = dst;
771 }

```

ip\_input.c

图9-18 函数save\_rte

759-771 当一个源路由选择的分组到达目的地时，`ip_dooptions`调用`save_rte`。`option`是一个指向分组的源路由选项的指针，`dst`是从分组首部来的`ip_src`（也就是，返回路由的目的地，图9-12中的HS）。如果选项的长度超过`ip_srcrt`结构，`save_rte`立即返回。

永远也不可能发生这种情况，因为`ip_srcrt`结构比最大选项长度(40字节)要大。

`save_rte`把该选项复制到`ip_srcrt`，计算并保存`ip_nhops`中源路由的跳数，把返回路由的目的地保存在`dst`中。

### 9.6.2 `ip_srcroute`函数

当响应某个分组时，ICMP和标准的运输层协议必须把分组带的任意源路由逆转。逆转源路由是通过`ip_srcroute`保存的路由构造的，如图9-19所示。

```

777 struct mbuf *
778 ip_srcroute()
779 {
780     struct in_addr *p, *q;
781     struct mbuf *m;
782
783     if (ip_nhops == 0)
784         return ((struct mbuf *) 0);
785     m = m_get(M_DONTWAIT, MT_SOOPTS);
786     if (m == 0)
787         return ((struct mbuf *) 0);
788
789 #define OPTSIZ (sizeof(ip_srcrt.nop) + sizeof(ip_srcrt.srcopt))
790
791     /* length is (nhops+1)*sizeof(addr) + sizeof(nop + srcopt header) */
792     m->m_len = ip_nhops * sizeof(struct in_addr) + sizeof(struct in_addr) +
793         OPTSIZ;
794
795     /*
796      * First save first hop for return route
797      */
798     p = &ip_srcrt.route[ip_nhops - 1];
799     *(mtod(m, struct in_addr *)) = *p--;
800
801     /*
802      * Copy option fields and padding (nop) to mbuf.
803      */
804     ip_srcrt.nop = IPOPT_NOP;
805     ip_srcrt.srcopt[IPOPT_OFFSET] = IPOPT_MINOFF;
806     bcopy((caddr_t) &ip_srcrt.nop,
807         mtod(m, caddr_t) + sizeof(struct in_addr), OPTSIZ);
808     q = (struct in_addr *) (mtod(m, caddr_t) +
809         sizeof(struct in_addr) + OPTSIZ);
810 #undef OPTSIZ
811
812     /*
813      * Record return path as an IP source route,
814      * reversing the path (pointers are now aligned).
815      */
816     while (p >= ip_srcrt.route) {
817         *q++ = *p--;
818     }
819
820     /*
821      * Last hop goes to final destination.

```

图9-19 `ip_srcroute` 函数

```

815     */
816     *q = ip_srcrt.dst;
817     return (m);
818 }

```

ip\_input.c

图9-19 (续)

777-783 ip\_srcroute把保存在ip\_srcrt结构中的源路由逆转后, 返回与 ipoption 结构(图9-26)格式类似的结果。如果 ip\_nhops 是0, 则没有保存的路由, 所以 ip\_srcroute返回一个指针。

记得在图8-13中, 当一个无效分组到达时, ipintr把ip\_nhops清零。运输层协议必须调用ip\_srcroute, 并在下一个分组到达之前自己保存逆转后的路由。正如以前我们注意到的, 这样做是正确的, 因为 ipintr在处理分组时, 在IP输入队列的下一个分组被处理之前都会调用运输层(TCP或UDP)的。

为源路由分配存储器缓存

784-786 如果ip\_nhops非0, ip\_srcroute就分配一个mbuf, 并把m\_len设置成足够大, 以便包含第一跳目的地、选项首部信息(OPTSZ)以及逆转后的路由。如果分配失败, 则返回一个空指针, 跟没有源路由一样。

p被初始化为指向到达路由的末尾, ip\_srcroute把最后记录的地址复制到mbuf的前面, 在这里它为外出的第一跳目的地开始逆转后的路由。然后该函数把一份 NOP选项(习题9.4)和源路由信息复制到mbuf中。

805-818 While循环把其余的IP地址从源路由中以相反的顺序复制到mbuf中。路由的最后一个地址被设置成到达分组中被 save\_rte放在ip\_srcrt.dst中的源站地址。返回一个指向mbuf的指针。图9-20说明了对图9-12的路由如何构造逆转的路由。

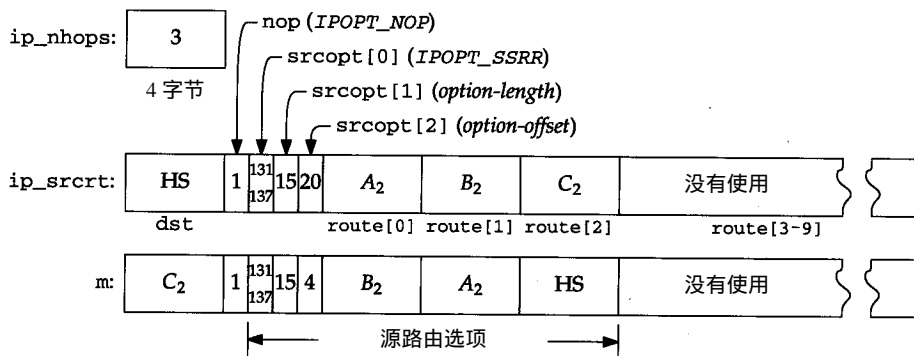


图9-20 ip\_srcroute 逆转ip\_srcrt 中的路由

## 9.7 时间戳选项

当分组穿过一个互联网时, 时间戳选项使各个系统把它当前的时间表示记录在分组的选项内。时间是以从UTC的午夜开始计算的毫秒计, 被记录在一个32 bit的字段里。

如果系统没有准确的UTC(几分钟以内)或没有每秒更新至少15次, 就不把它作为标准时间。非标准时间必须把时间戳字段的高比特位置位。

有三种时间戳选项类型，Net/3通过如图9-22所示的ip\_timestamp结构访问。

114-133 如同ip结构(图8-10)一样，#ifs保证比特字段访问选项中正确的比特位。图9-21中列出了由ipt\_flg指定的三种时戳选项类型。

ipt_flg	值	描 述
IPOPT_TS_TSONLY	0	记录时间戳
IPOPT_TS_TSANDADDR	1	记录地址和时间戳
	2	保留
IPOPT_TS_PRESPEC	3	只在预先指定的系统记录时间戳
	4-15	保留

图9-21 ipt\_flg 可能的值

```

114 struct ip_timestamp {
115     u_char  ipt_code;           /* IPOPT_TS */
116     u_char  ipt_len;           /* size of structure (variable) */
117     u_char  ipt_ptr;           /* index of current entry */
118 #if BYTE_ORDER == LITTLE_ENDIAN
119     u_char  ipt_flg:4;         /* flags, see below */
120     u_char  ipt_oflw:4;        /* overflow counter */
121 #endif
122 #if BYTE_ORDER == BIG_ENDIAN
123     u_char  ipt_oflw:4;        /* overflow counter */
124     u_char  ipt_flg:4;        /* flags, see below */
125 #endif
126     union ipt_timestamp {
127         n_long  ipt_time[1];
128         struct ipt_ta {
129             struct in_addr ipt_addr;
130             n_long  ipt_time;
131         } ipt_ta[1];
132     } ipt_timestamp;
133 };

```

ip.h

ip.h

图9-22 ip\_timestamp 结构和常量

初始主机必须构造一个具有足够大的数据区存放可能的时间戳和地址的时间戳选项。对于ipt\_flg为3的时间戳选项，初始主机在构造该选项时，填写要记录时间戳的系统的地址。图9-23显示了三种时间戳选项的结构。

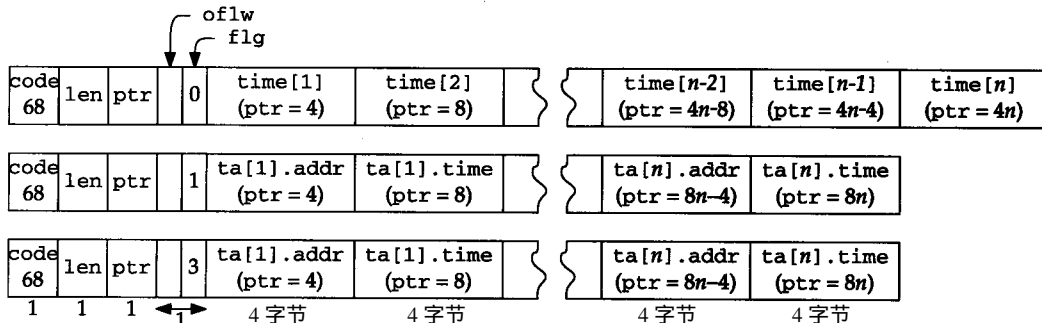


图9-23 三种时间戳选项(省略ipt\_)

因为IP选项只能有40个字节，所以时戳选项限制只能有9个时戳(ipt\_flg等于0)或4个地

址和时间戳对(`ipt_flg`等于1或3)。图9-24显示了对三种不同的时戳选项类型的处理。

674-684 如果选项长度小于5个字节(时戳选项的最小长度),则`ip_dooptions`发出一个ICMP参数问题差错报文。`oflw`字段统计由于选项数据区满而无法登记时戳的系统个数。如果数据区满,则`oflw`加1;当它本身超过16(它是一个4 bit的字段)而溢出时,发出一个ICMP参数问题差错报文。

```

674         case IPOPT_TS:
675             code = cp - (u_char *) ip;
676             ipt = (struct ip_timestamp *) cp;
677             if (ipt->ipt_len < 5)
678                 goto bad;
679             if (ipt->ipt_ptr > ipt->ipt_len - sizeof(long)) {
680                 if (++ipt->ipt_oflw == 0)
681                     goto bad;
682                 break;
683             }
684             sin = (struct in_addr *) (cp + ipt->ipt_ptr - 1);
685             switch (ipt->ipt_flg) {

686         case IPOPT_TS_TSONLY:
687             break;

688         case IPOPT_TS_TSANDADDR:
689             if (ipt->ipt_ptr + sizeof(n_time) +
690                 sizeof(struct in_addr) > ipt->ipt_len)
691                 goto bad;
692             ipaddr.sin_addr = dst;
693             ia = (INA) ifaof_ifpforaddr((SA) & ipaddr,
694                                         m->m_pkthdr.rcvif);
695             if (ia == 0)
696                 continue;
697             bcopy((caddr_t) & IA_SIN(ia)->sin_addr,
698                   (caddr_t) sin, sizeof(struct in_addr));
699             ipt->ipt_ptr += sizeof(struct in_addr);
700             break;

701         case IPOPT_TS_PRESPEC:
702             if (ipt->ipt_ptr + sizeof(n_time) +
703                 sizeof(struct in_addr) > ipt->ipt_len)
704                 goto bad;
705             bcopy((caddr_t) sin, (caddr_t) & ipaddr.sin_addr,
706                   sizeof(struct in_addr));
707             if (ifa_ifwithaddr((SA) & ipaddr) == 0)
708                 continue;
709             ipt->ipt_ptr += sizeof(struct in_addr);
710             break;

711         default:
712             goto bad;
713     }
714     ntime = iptime();
715     bcopy((caddr_t) & ntime, (caddr_t) cp + ipt->ipt_ptr - 1,
716           sizeof(n_time));
717     ipt->ipt_ptr += sizeof(n_time);
718 }
719 }

```

ip\_input.c

ip\_input.c

图9-24 函数`ip_dooptions` : 时间戳选项处理

### 1. 只有时间戳

685-687 对于ipt\_flg为0的时间戳选项(IPOPT\_TS\_TSONLY), 所有的工作都在switch语句之后再作。

### 2. 时间戳和地址

688-700 对于ipt\_flg为1的时间戳选项(IPOPT\_TS\_TSANDADDR), 接收接口的地址被记录下来(如果数据区还有空间), 选项的指针前进一步。因为Net/3支持一个接收接口上的多IP地址, 所以ip\_dooptions调用ifaof\_ifpforaddr选择与分组的初始目的地址(也就是在任何源路由选择发生之前的目的地)最匹配的地址。如果没有匹配, 则跳过时间戳选项(INA和SA定义如图9-15所示)。

### 3. 预定地址上的时间戳

701-710 如果ipt\_flg为3(IPOPT\_TS\_PRESPEC), ifa\_ifwithaddr确定选项中指定的下一个地址是否与系统的某个地址匹配。如果不匹配, 该选项要求在这个系统上不处理; continue使ip\_dooptions继续处理下一个选项。如果下一个地址与系统的某个地址匹配, 则选项的指针前进到下一个位置, 控制交给switch的后面。

### 4. 插入时间戳

711-713 default截获无效的ipt\_flg值, 并把控制传递到bad。

714-719 时间戳用switch语句后面的代码写入到选项中。iptime返回自从UTC午夜起到现在的时间数, ip\_dooptions记录此时间戳, 并增加此选项相对于下一个位置的偏移。

## iptime函数

图9-25显示了iptime的实现。

```
458 n_time  
459 iptime()  
460 {  
461     struct timeval atv;  
462     u_long t;  
  
463     microtime(&atv);  
464     t = (atv.tv_sec % (24 * 60 * 60)) * 1000 + atv.tv_usec / 1000;  
465     return (htonl(t));  
466 }
```

ip\_icmp.c

ip\_icmp.c

图9-25 函数iptime

458-466 microtime返回从UTC1970年1月1日午夜以来的时间, 放在timeval结构中。从午夜以来的毫秒数用atv计算, 并以网络字节序返回。

卷1的7.4节有几个时间戳选项的例子。

## 9.8 ip\_insetoptions函数

我们在8.6节看到, ip\_output函数接收一个分组和选项。当ip\_forward调用该函数时, 选项已经是分组的一部分, 所以ip\_forward总是把一个空选项指针传给ip\_output。但是, 运输层协议可能会把由ip\_insetoptions(由图8-22中的ip\_output调用)合并到分组中的选项传递给ip\_forward。

ip\_insetoptions希望选项在ipoption结构中被格式化,如图9-26所示。

```

92 struct ipoption {
93     struct in_addr ipopt_dst; /* first-hop dst if source routed */
94     char ipopt_list[MAX_IPOPTLEN]; /* options proper */
95 };

```

ip\_var.h

图9-26 结构ipoption

92-95 该结构只有两个成员: ipopt\_dst, 如果选项表中有源路由, 则其中有第一跳目的地, ipopt\_list, 是一个最多40(MAX\_IPOPTLEN)字节的选项矩阵, 其格式我们在本章中已做了描述。如果选项表中没有源路由, 则 ipopt\_dst全为0。

注意, ip\_srcrt结构(图9-16)和由 ip\_srcroute(图9-19)返回的 mbuf都符合由 ipoption结构所指定的格式。图9-27把结构ip\_srcrt和ipoption作了比较。

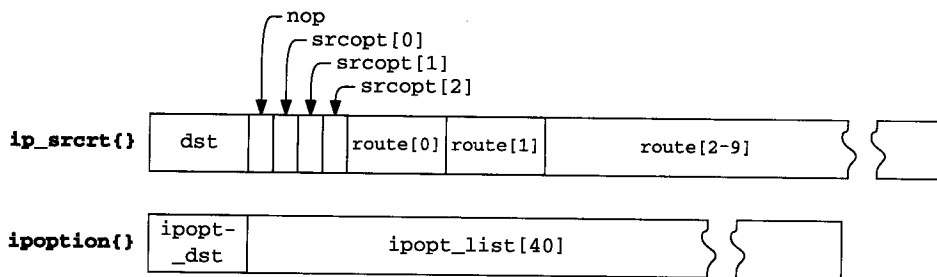


图9-27 结构ip\_srcrt 和ipoption

结构ip\_srcrt比ipoption多4个字节。路由矩阵的最后一个入口(route[9])永远都不会填上, 因为这样的话, 源路由选项将会有 44字节长, 比IP首部所能容纳的要大(图9-16)。

函数ip\_insetoptions如图9-28所示。

```

352 static struct mbuf *
353 ip_insetoptions(m, opt, phlen)
354 struct mbuf *m;
355 struct mbuf *opt;
356 int *phlen;
357 {
358     struct ipoption *p = mtod(opt, struct ipoption *);
359     struct mbuf *n;
360     struct ip *ip = mtod(m, struct ip *);
361     unsigned optlen;

362     optlen = opt->m_len - sizeof(p->ipopt_dst);
363     if (optlen + (u_short) ip->ip_len > IP_MAXPACKET)
364         return (m); /* XXX should fail */
365     if (p->ipopt_dst.s_addr)
366         ip->ip_dst = p->ipopt_dst;
367     if (m->m_flags & M_EXT || m->m_data - optlen < m->m_pktdat) {
368         MGETHDR(n, M_DONTWAIT, MT_HEADER);
369         if (n == 0)

```

ip\_output.c

图9-28 函数ip\_insetoptions



```

370         return (m);
371         n->m_pkthdr.len = m->m_pkthdr.len + optlen;
372         m->m_len -= sizeof(struct ip);
373         m->m_data += sizeof(struct ip);
374         n->m_next = m;
375         m = n;
376         m->m_len = optlen + sizeof(struct ip);
377         m->m_data += max_linkhdr;
378         bcopy((caddr_t) ip, mtod(m, caddr_t), sizeof(struct ip));
379     } else {
380         m->m_data -= optlen;
381         m->m_len += optlen;
382         m->m_pkthdr.len += optlen;
383         ovbcopy((caddr_t) ip, mtod(m, caddr_t), sizeof(struct ip));
384     }
385     ip = mtod(m, struct ip *);
386     bcopy((caddr_t) p->ipt_list, (caddr_t) (ip + 1), (unsigned) optlen);
387     *phlen = sizeof(struct ip) + optlen;
388     ip->ip_len += optlen;
389     return (m);
390 }

```

ip\_output.c

图9-28 (续)

352-364 ip\_insertoptions有三个参数：m，外出的分组；opt，在结构中格式化的选项；phlen，一个指向整数的指针，在这里返回新首部的长度（在插入选项之后）。如果插入选项分组长度超过最大分组长度 65 535(IP\_MAXPACKET)字节，则自动将选项丢弃。ip\_dooptions认为ip\_insertoptions永远都不会失败，所以无法报告差错。幸好，很少有应用程序会试图发送最大长度的数据报，更别说选项了。

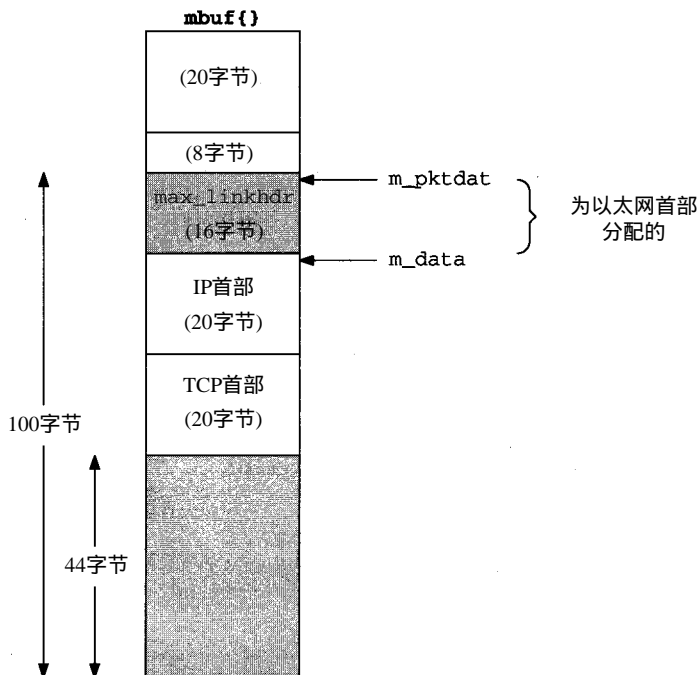


图9-29 函数ip\_insertoptions : TCP报文段

365-366 如果`ipopt_dst.s_addr`指定了一个非零地址,则选项中包括了源路由,并且分组首部的`ip_dst`被源路由中的第一跳目的地代替。

在26.2节中,我们将看到TCP调用`MGETHDR`为IP和TCP首部分配一个单独的mbuf。图9-29显示了在第367~378行代码执行之前,一个TCP报文段的mbuf结构。

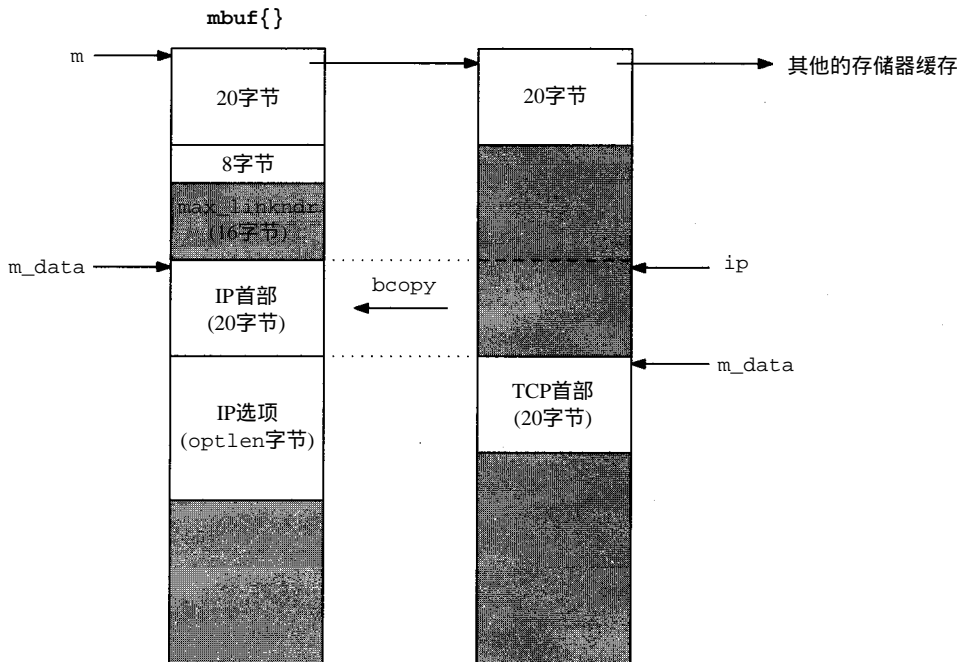


图9-30 函数`ip_inseroptions` : 在选项被复制后的TCP报文段

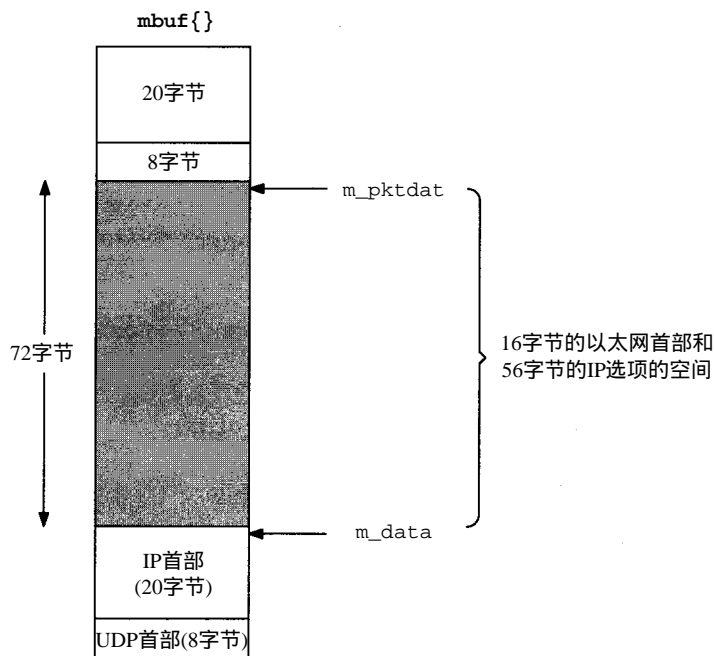


图9-31 函数`ip_inseroptions` : UDP报文段

如果被插入的选项占据了多于 16 的字节数，则第 367 行的测试为真，并调用 MGETHDR 分配另一个 mbuf。图 9-30 显示了选项被复制到新的 mbuf 后，该缓存的结构。

367-378 如果分组首部被存放在一簇，或者第一个缓存中没有多余选项的空间，则 ip\_insetoptions 分配一个新的分组首部 mbuf，初始化它的长度，从旧的缓存中把该 IP 首部截取下来，并把该首部从旧缓存中移动到新缓存中。

如 23.6 节中所述，UDP 使用 M\_PREPEND 把 UDP 和 IP 首部放置到缓存的最后，与数据分离。如图 9-31 所示。因为首部是放在缓存的最后，所以在缓存中总有空间存放选项，对 UDP 来说，第 367 行的条件总为假。

379-384 如果分组在缓存数据区的开始部分有存放选项的空间，则修改 m\_data 和 m\_len，以包含 optlen 更多的字节。并且当前的 IP 首部被 ovbcopy（能够处理源站和目的站的重叠问题）移走，为选项腾出位置。

385-390 ip\_insetoptions 现在可以把 ipoption 结构的成员 ipopt\_list 直接复制到紧接在 IP 首部后面的缓存中。把新的首部长度存放在 \*phlen 中，修改数据报长度 (ip\_len)，并返回一个指向分组首部缓存的指针。

## 9.9 ip\_pcbopts 函数

函数 ip\_pcbopts 把 IP 选项表及 IP\_OPTIONS 插口选项转换成 ip\_output 希望的格式：ipoption 结构。如图 9-32 所示。

ip\_output.c

```

559 int
560 ip_pcbopts(pcbopt, m)
561 struct mbuf **pcbopt;
562 struct mbuf *m;
563 {
564     cnt, optlen;
565     u_char *cp;
566     u_char opt;
567     /* turn off any old options */
568     if (*pcbopt)
569         (void) m_free(*pcbopt);
570     *pcbopt = 0;
571     if (m == (struct mbuf *) 0 || m->m_len == 0) {
572         /*
573          * Only turning off any previous options.
574          */
575         if (m)
576             (void) m_free(m);
577         return (0);
578     }
579     if (m->m_len % sizeof(long))
580         goto bad;
581     /*
582      * IP first-hop destination address will be stored before
583      * actual options; move other options back
584      * and clear it when none present.
585      */
586     if (m->m_data + m->m_len + sizeof(struct in_addr) >= &m->m_dat[MLEN])
587         goto bad;
588     cnt = m->m_len;
589     m->m_len += sizeof(struct in_addr);

```

图 9-32 函数 ip\_pcbopts

```

590     cp = mtod(m, u_char *) + sizeof(struct in_addr);
591     ovbcopy(mtod(m, caddr_t), (caddr_t) cp, (unsigned) cnt);
592     bzero(mtod(m, caddr_t), sizeof(struct in_addr));

593     for (; cnt > 0; cnt -= optlen, cp += optlen) {
594         opt = cp[IPOPT_OPTVAL];
595         if (opt == IPOPT_EOL)
596             break;
597         if (opt == IPOPT_NOP)
598             optlen = 1;
599         else {
600             optlen = cp[IPOPT_OLEN];
601             if (optlen <= IPOPT_OLEN || optlen > cnt)
602                 goto bad;
603         }
604         switch (opt) {
605             default:
606                 break;

607             case IPOPT_LSRR:
608             case IPOPT_SSRR:
609                 /*
610                  * user process specifies route as:
611                  * ->A->B->C->D
612                  * D must be our final destination (but we can't
613                  * check that since we may not have connected yet).
614                  * A is first hop destination, which doesn't appear in
615                  * actual IP option, but is stored before the options.
616                  */
617                 if (optlen < IPOPT_MINOFF - 1 + sizeof(struct in_addr))
618                     goto bad;
619                 m->m_len -= sizeof(struct in_addr);
620                 cnt -= sizeof(struct in_addr);
621                 optlen -= sizeof(struct in_addr);
622                 cp[IPOPT_OLEN] = optlen;
623                 /*
624                  * Move first hop before start of options.
625                  */
626                 bcopy((caddr_t) & cp[IPOPT_OFFSET + 1], mtod(m, caddr_t),
627                     sizeof(struct in_addr));
628                 /*
629                  * Then copy rest of options back
630                  * to close up the deleted entry.
631                  */
632                 ovbcopy((caddr_t) (&cp[IPOPT_OFFSET + 1] +
633                     sizeof(struct in_addr)),
634                     (caddr_t) & cp[IPOPT_OFFSET + 1],
635                     (unsigned) cnt + sizeof(struct in_addr));
636                 break;
637         }
638     }
639     if (m->m_len > MAX_IPOPTLEN + sizeof(struct in_addr))
640         goto bad;
641     *pcbopt = m;
642     return (0);

643 bad:
644     (void) m_free(m);
645     return (EINVAL);
646 }

```

ip\_output.c

图9-32 (续)

559-562 第一个参数，`pcbopt`引用指向当前选项表的指针。然后该函数用一个指向新的选项表的指针来代替该指针，这个新选项表是由第二个参数 `m`指向的缓存链所指定的选项构造而来。该过程所准备的选项表，将被包含在 `IP_OPTIONS`插口选项中，除了LSRR和SSRR选项的格式外，看起来象一个标准的IP选项表。对这些选项，第一跳目的地址是作为路由的第一个地址出现的。图9-14显示，在外出的分组中，第一跳目的地址是作为目的地址出现的，而不是路由的第一个地址。

#### 1. 扔掉以前的选项

563-580 所有被`m_free`和`*pcbopt`扔掉的选项都被清除。如果该过程传来一个空缓存或者根本不传递缓存，则该函数不安装任何新的选项，并立即返回。

如果新选项表没有填充到4 bit的边界，则`ip_pcbopts`跳到`bad`，扔掉该表，并返回`EINVAL`。

该函数的其余部分重新安排该表，使其看起来象一个`ipoption`结构。图9-33显示了这个过程。

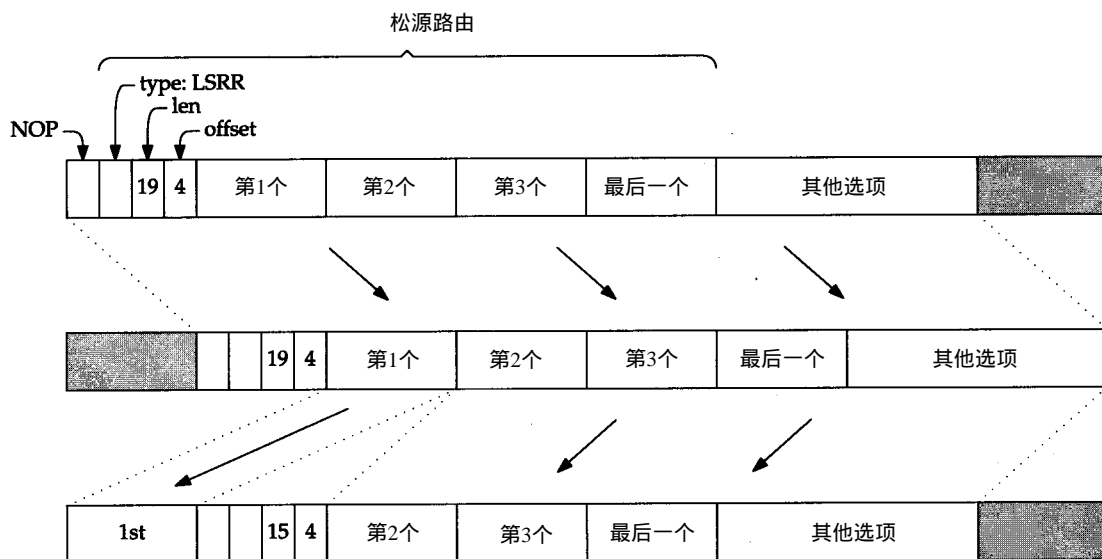


图9-33 `ip_pcbopts` 选项表处理

#### 2. 为第一跳目的地腾出位置

581-592 如果缓存中没有位置，则把所有的数据都向缓存的末尾移动4个字节(是一个`in_addr`结构的大小)。ovbcopy完成复制。bzero清除缓存开始的4个字节。

#### 3. 扫描选项表

593-606 for循环扫描选项表，寻找LSRR和SSRR选项。对多字节选项，该循环也验证选项的长度是否合理。

#### 4. 重新安排LSRR和SSRR选项

607-638 当该循环找到一个LSRR或SSRR选项时，它把缓存的大小、循环变量和选项长度减去4，因为选项的每一个地址将被移走，并被移到缓存的前面。

`bcopy`把第一个地址移走，`ovbcopy`把选项的其他部分移动4个字节，来填补第一个地

址留下的空隙。

#### 5. 清除

639-646 循环结束后,选项表的大小(包括第一跳地址)必须不超过44 (`MAX_IPOPTLEN + 4`)字节。更长的选项表无法放入IP分组的首部。该表被保存在 `*pcbopt` 中,函数返回。

### 9.10 一些限制

除了管理和诊断工具生成的IP数据报外,很少出现选项。卷1讨论了两个最常用的工具, `ping`和`tracert`。使用IP选项的应用程序很难写。编程接口的文档很少,也没有很好地标准化。许多厂商提供的应用程序,比如 `Telnet`和`FTP`,并没有为用户提供方法,来指定如源路由等的选项。

在大的互联网上,记录路由、时间戳和源路由选项的用途被IP首部的最大长度所限制。许多路由含有的跳数远多于40选项字节所能表示的。当多选项在同一分组中出现时,所能得到的空间是几乎没有用的。IPv6用一个更为灵活的选项首部设计强调了这个问题。

在分片过程中,IP只把某些选项复制到非初始片上,因为重组时会扔掉非初始片上的选项。在目的主机上,运输层协议只能用到初始片上的选项(10.6节)。但有些选项,如源路由,即使在目的主机上,被非初始片丢弃,依然必须被复制到每个分片。

### 9.11 小结

本章中我们显示了IP选项的格式和处理过程。我们没有讨论安全和流ID选项,因为Net/3没有实现它们。

我们看到,多字节选项的大小是由源主机在构造它们时确定的。最大选项首部长度只有40字节,这严格限制了IP选项的使用。

源路由选项要求最多的支持。到达的源路由被 `save_rte`保存,并保留在 `ip_srcroute`中。通常不转发分组的主机可能转发源路由选择的分组,但是RFC 1122默认要求不允许这种功能。Net/3没有对这种特性的判断,总是转发源路由选择的分组。

最后,我们看到 `ip_insertoptions`是如何把选项插入到一个外出的分组中去的。

### 习题

- 9.1 如果一个分组中有两个不同的源路由选项会发生什么情况?
- 9.2 一些商用路由器可以被配置成按照分组的IP目的地址扔掉它们。通过种方式,可以把一台或一组主机通过路由器隔离在更大的互联网之外。请描述源路由选择的分组如何绕过这个机制。假定网络中至少有一个主机,路由器没有阻塞,并转发源路由选择的数据报。
- 9.3 某些主机可能没有被配置成默认路由。这样主机就无法路由选择到其他与它直接相连的网络。请描述源路由如何与这种类型的主机通信。
- 9.4 为什么NOP采用如图9-16所示的 `ip_srcrt`结构?
- 9.5 时间戳选项中非标准时间值会和标准时间值混淆吗?
- 9.6 `ip_dooptions`在处理其他选项之前要把分组的地址保存在 `dest`中(图9-8)。为什么?