TCP/IP Illustrated, Volume 2

The Implementation

Gary R. Wright
W. Richard Stevens

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

TCP/IP Illustrated

The Implementation

Volume 2

W. Richard Stevens

Gary R. Wright

Addison-Wesley Professional

The IP header is completed by restoring the correct datagram length (`ip_len`), header length (`ip_hl`), and protocol (`ip_p`), and clearing the TOS field (`ip_tos`).

RFCs 792 and 1122 recommend that the TOS field be set to 0 for ICMP messages.

`126-129`

The completed message is passed to `icmp_reflect`, where it is sent back to the source host. The invalid datagram is discarded.

11.12. `icmp_reflect` Function

`icmp_reflect` sends ICMP replies and errors back to the source of the request or back to the source of the invalid datagram. It is important to remember that `icmp_reflect` reverses the source and destination addresses in the datagram before sending it. The rules regarding source and destination addresses of ICMP messages are complex. Figure 11.34 summarizes the actions of several functions in this area.

Figure 11.34. ICMP discard and address summary.

Function	Summary
`icmp_input`	Replace an all-0s source address in address mask requests with the broadcast or destination address of the receiving interface.
`icmp_error`	Discard error messages caused by datagrams sent as link-level broadcasts or multicasts. Should discard (but does not) messages caused by datagrams sent to IP broadcast or multicast addresses.
`icmp_reflect`	Discard messages instead of returning them to a multicast or experimental address. Convert nonunicast destinations to the address of the receiving interface, which makes the destination address a valid source address for the return message. Swap the source and destination addresses.
`ip_output`	Discards outgoing broadcasts at the request of ICMP (i.e., discards errors generated by packets sent to a broadcast address)

We describe the `icmp_reflect` function in three parts: source and destination address selection, option construction, and assembly and transmission. Figure 11.35 shows the first part of the function.

Figure 11.35. `icmp_reflect` function: address selection.

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) {
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

Set destination address

329-345

`icmp_reflect` starts by making a copy of **`ip_dst`** and moving **`ip_src`**, the source of the request or error datagram, to **`ip_dst`**. `icmp_error` and `icmp_reflect` ensure that **`ip_src`** is a valid destination address for the error message. `ip_output` discards any packets sent to a broadcast address.

Select source address

346-371

`icmp_reflect` selects a source address for the message by searching `in_ifaddr` for the interface with a unicast or broadcast address matching the destination address of the original datagram. On a multihomed host, the matching interface may not be the interface on which the datagram was received. If there is no match, the `in_ifaddr` structure of the receiving interface is selected or, failing that (the interface may not be configured for IP), the first address in `in_ifaddr.` The function sets `ip_src` to the selected address and changes `ip_ttl` to 255 (`MAXTTL`) because the error is a new datagram.

RFC 1700 recommends that the TTL field of all IP packets be set to 64. Many systems, however, set the TTL of ICMP messages to 255 nowadays.

There is a tradeoff associated with TTL values. A small TTL prevents a packet from circulating in a routing loop but may not allow a packet to reach a site far (many hops) away. A large TTL allows packets to reach distant hosts but lets packets circulate in routing loops for a longer period of time.

RFC 1122 *requires* that source route options, and *recommends* that record route and timestamp options, from an incoming echo request or timestamp request, be attached to a reply. The source route must be reversed in the process. RFC 1122 is silent on how these options should be handled on other types of ICMP replies. Net/3 applies these rules to the address mask request, since it calls `icmp_reflect` (Figure 11.21) after constructing the address mask reply.

The next section of code (Figure 11.36) constructs the options for the ICMP message.

Figure 11.36. `icmp_reflect` function: option construction.

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                 /*
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

Get reversed source route

372-385

If the incoming datagram did not contain options, control passes to line 430 (Figure 11.37). The error messages that `icmp_error` sends to `icmp_reflect` never have IP options, and so the following code applies only to ICMP requests that are converted to replies and passed directly to `icmp_reflect`.

Figure 11.37. `icmp_reflect` function: final assembly.

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

`cp` points to the start of the options for the *reply*. `ip_srcroute` reverses and returns any source route option saved when `ipintr` processed the datagram. If `ip_srcroute` returns 0, the request did not contain a source route option so `icmp_reflect` allocates and initializes an mbuf to serve as an empty `ipoption` structure.

Add record route and timestamp options

`386-416`

If `opts` points to an mbuf, the `for` loop searches the options from the *original* IP header and appends the record route and timestamp options to the source route returned by `ip_srcroute.`

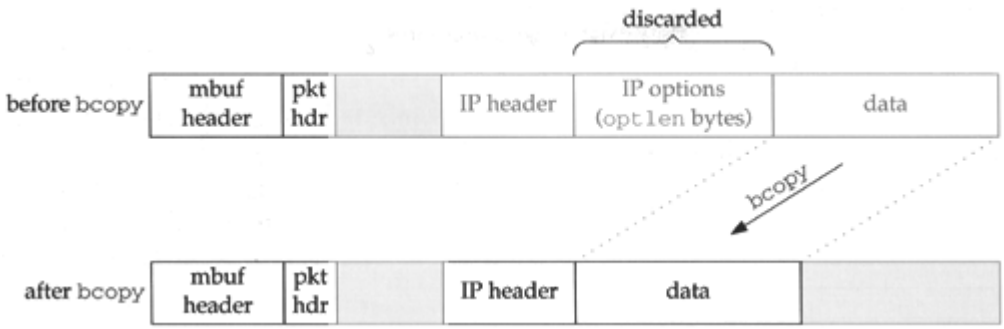
The options in the original header must be removed before the ICMP message can be sent. This is done by the code shown in Figure 11.37.

Remove original options

`417-429`

`icmp_reflect` removes the options from the original request by moving the ICMP message up to the end of the IP header. This is shown in Figure 11.38. The new options, which are in the mbuf pointed to by `opts,` are reinserted by `ip_output.`

Figure 11.38. `icmp_reflect:` removal of options.



Send message and cleanup

`430-435`

The broadcast and multicast flags are explicitly cleared before passing the message and options to `icmp_send`, after which the mbuf containing the options is released.

11.13. `icmp_send` Function

`icmp_send` (Figure 11.39) processes all outgoing ICMP messages and computes the ICMP checksum before passing them to the IP layer.

Figure 11.39. `icmp_send` function.

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

`440-457`

As it does when checking the ICMP checksum in `icmp_input`, Net/3 adjusts the mbuf data pointer and length to hide the IP header and lets `in_cksum` look only at the ICMP message. The computed checksum is placed in the header at **`icmp_cksum`** and the datagram and any options are