Chapter 8 – Traceroute

Disadvantages of using record route option in Ping

Not all routers support record route.
Not enough slots in the header.
Receiver must be kind enough to return the information by either:
- Sending an explicit reply to some address or
- Reflecting the record route option (done by most pings.)

Alternate approach used by *traceroute*

TTL field is used in the Internet as a ‘‘hop counter’’.
When TTL reaches 0 an ICMP time exceeded error message is sent.

Traceroute
- sends UPD datagrams –– to a port on which hopefully no one is listening!
- receives ICPM error messages of class
  - time exceeded – type 11 code 0
    - (other time exceeded code is 1 .. reassembly timeout)
  - unreachable port if the packet reaches the destination

Output format

Initial TTL value
Name and IP address of ICMP responder
Response times for each of 3 datagrams sent
- * => no response in 5 seconds.

```
[L:\ACAD\CS825] ==> tracerte watson.ibm.com

 0  * citron.cs.clemson.edu (130.127.48.1)  0 ms  0 ms
 1  citron.cs.clemson.edu (130.127.48.1)  0 ms  0 ms  0 ms
 2  130.127.44.1 (130.127.44.1)  0 ms  31 ms  0 ms
 3  130.127.2.1 (130.127.2.1)  0 ms  0 ms  0 ms
 4  clemson-gw.clemson.edu (130.127.8.5)  0 ms  31 ms  0 ms
 5  gnu1-clem-c1.sura.net (192.221.4.33)  0 ms  31 ms  0 ms
 6  atu1-gnu1-c3mb.sura.net (192.221.1.1)  31 ms  63 ms  31 ms
 7  cpe1-fddi1.Atlanta.mci.net (192.221.42.100)  219 ms 219 ms 219 ms
 8  border1-hssi1-0.Atlanta.mci.net (204.70.16.5)  62 ms 63 ms 31 ms
 9  core-fddi-0.Atlanta.mci.net (204.70.2.49)  31 ms 63 ms 31 ms
10  core2-aip-4.Atlanta.mci.net (204.70.1.70)  32 ms 31 ms 31 ms
11  core1-hssi-2.Greensborough.mci.net (204.70.1.126)63 ms 31 ms  63 ms
12  core2-hssi-3.Washington.mci.net (204.70.1.130)  62 ms *  125 ms
13  core1-aip-4.Washington.mci.net (204.70.1.73)  94 ms 125 ms  94 ms
14  border2-fddi-0.Washington.mci.net (204.70.3.2) 125 ms  63 ms  62 ms
15  mae-east-plusplus.Washington.mci.net (204.70.57.10) 63 ms  62 ms *
16  mae-east.ans.net (192.41.177.140)  63 ms *  62 ms
17  t3-3.cnss56.Washington-DC.t3.ans.net(140.222.56.4)63 ms  94 ms 63 m
18  t3-0.cnss32.New-York.t3.ans.net (140.222.32.1)  93 ms  63 ms  62 ms
19  cnss37.New-York.t3.ans.net (140.222.32.197)  62 ms  63 ms  93 ms
20  enss164.t3.ans.net (199.222.70.114)  94 ms  62 ms  63 ms
21  watson.ibm.com (129.34.139.4)  94 ms  63 ms  93 ms
```

Another example: (The source here was a PPP link)

```
 0  130.127.48.249 (130.127.48.249)  125 ms  125 ms  93 ms
 1  130.127.48.249 (130.127.48.249)  125 ms  125 ms  94 ms
 2  citron.cs.clemson.edu (130.127.48.1)  125 ms  125 ms  125 ms
 3  math_rtr.clemson.edu (130.127.44.9)  125 ms  94 ms  125 ms
 4  fredholm.math.clemson.edu (130.127.112.20)  156 ms *  125 ms
```

Another example: (These were done in Feb 2001)

```
/local/mac2/r128 ==> /usr/sbin/traceroute www.cdrom.com
traceroute to web1.cdrom.com (209.155.82.19), 30 hops max, 40 byte packets
 1  cs-gateway.cs.clemson.edu (130.127.48.1)  2.709 ms  1.102 ms  0.983 ms
 2  border-atm-r01.clemson.edu (130.127.12.6)  0.838 ms  0.748 ms  0.822 ms
 3  atm2-0-2.r01.scgnvl.infoave.net(165.166.125.137)7.547 ms 7.160 ms 6.950 ms
 4  atm1-0-7.r02.gaatln.infoave.net (165.166.126.141)23.115 ms 22.734 23.157
 5  atm1-0-0-100.ar3.ATL1.gblx.net(64.211.166.65)23.804ms 25.112 ms  24.332 ms
 6  pos2-2-155M.cr2.ATL1.gblx.net (206.132.115.141)23.617ms 24.532ms 23.641 ms
 7  pos10-3-622M.cr1.IAD3.gblx.net (208.48.234.221)39.046ms 40.404ms 38.924 ms
 8  208.49.231.54 (208.49.231.54)  35.261 ms  34.714 ms  34.821 ms
 9  vva1-sfo2.ATM.us.crl.com (165.113.0.253)122.601 ms 121.206 ms  123.277 ms
10  209.155.82.19 (209.155.82.19)122.712 ms 123.311 ms  123.973 ms
/local/mac2/r128 ==>

traceroute to www.orst.edu (128.193.4.112), 30 hops max, 40 byte packets
 1  cs-gateway.cs.clemson.edu (130.127.48.1)1.217 ms  0.889 ms  1.028 ms
 2  border-atm-r01.clemson.edu (130.127.12.6)  0.821 ms  0.824 ms  0.858 ms
 3  sox-atl.clemson.edu (130.127.3.6)  16.955 ms  16.771 ms  16.633 ms
 4  199.77.193.10 (199.77.193.10)  16.756 ms  17.171 ms  17.130 ms
 5  ipls-atla.abilene.ucaid.edu (198.32.8.41)  26.936 ms  27.136 ms  26.666 ms
 6  kscy-ipls.abilene.ucaid.edu (198.32.8.5)  35.758 ms  35.690 ms  35.840 ms
 7  dnvr-kscy.abilene.ucaid.edu (198.32.8.13)  46.363 ms  46.257 ms  46.646 ms
 8  ogig-den.oregon-gigapop.net (198.32.163.13)77.828 ms  78.067 ms  77.914 ms
 9  0car-0gw.oregon-gigapop.net (198.32.163.26)78.345 ms  78.024 ms  78.063 ms
10  eugn-core2-gw.nero.net (207.98.64.21)  78.407 ms  78.444 ms  79.047 ms
11  eugn-core1-gw.nero.net (207.98.64.162)  79.044 ms  79.022 ms  78.763 ms
12  corvallis-hub.nero.net (207.98.64.6)  80.845 ms  80.628 ms  81.470 ms
13  orstbrdr-gw.orst.edu (199.201.139.1)  80.990 ms  82.953 ms  80.709 ms
14  orstsw1-gw.ORST.EDU (128.193.6.21)  80.524 ms  80.732 ms  80.637 ms
15  www.orst.edu (128.193.4.112)  80.704 ms  81.041 ms  80.702 ms

misc/doc ==> traceroute www.nyu.edu
traceroute to WWWSERVER.nyu.edu (128.122.108.9), 30 hops max, 40 byte packets
 1  cs-gateway.cs.clemson.edu (130.127.48.1)  0.620 ms  0.626 ms  0.415 ms
 2  border-atm-r01.clemson.edu (130.127.12.6)  1.297 ms  0.858 ms  0.921 ms
 3  sox-atl.clemson.edu (130.127.3.6)  17.273 ms  17.316 ms  17.173 ms
 4  atla.abilene.sox.net (199.77.193.10)  17.494 ms  17.257 ms  17.625 ms
 5  ipls-atla.abilene.ucaid.edu (198.32.8.41)  27.737 ms  27.229 ms  27.435 ms
 6  clev-ipls.abilene.ucaid.edu (198.32.8.26)  34.214 ms  33.504 ms  33.505 ms
 7  199.109.2.1 (199.109.2.1)  38.283 ms  38.091 ms  38.027 ms
 8  * * *
 9  NYUGWA-FA-1-0-0.NYU.NET (192.76.177.65)  44.867 ms  44.628 ms  44.309 ms
10  WWHGWF-FDDI-0-0.NYU.NET (128.122.253.70)  45.439 ms  46.545 ms  45.940 ms
11  * * WWHGWF-FDDI-0-0.NYU.NET (128.122.253.70)  48.394 ms !X
12  * WWHGWF-FDDI-0-0.NYU.NET (128.122.253.70)  47.461 ms !X *
13  * WWHGWF-FDDI-0-0.NYU.NET (128.122.253.70)  46.939 ms !X *
14  WWHGWF-FDDI-0-0.NYU.NET (128.122.253.70)  45.387 ms !X * *
```

Traceroute code

Here is the obligatory copyright:

```
/*-
 * Copyright (c) 1990, 1993
 * The Regents of the University of California.  All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by  Van Jacobson.
 *
 * Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions  are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 * This product includes software developed by the University of
 * California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
*/


#include <sys/socket.h>
#include <sys/file.h>
#include <sys/ioctl.h>

#if __linux__
#include <endian.h>
#endif
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>

#include <arpa/inet.h>

#include <netdb.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define  MAXPACKET   65535 /* max ip packet size */
#ifndef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN  64
#endif

#ifndef FD_SET
#define NFDBITS         (8*sizeof(fd_set))
#define FD_SETSIZE       NFDBITS
#define FD_SET(n, p)    ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n,p)     ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p)  ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p)      bzero((char *)(p), sizeof(*(p)))
#endif

#define Fprintf (void)fprintf
#define Sprintf (void)sprintf
#define Printf  (void)printf
```

```
/*
 * format of a (udp) probe packet.
 */
struct opacket {
   struct ip ip;        /* ip header.                          */
   struct udphdr udp;   /* udp header                          */
   u_char seq;          /* sequence number of this packet      */
   u_char ttl;          /* ttl packet left with                */
   struct timeval tv;   /* time packet left                    */
};

u_char   packet[512];      /* last inbound (icmp) packet      */
struct opacket *outpacket; /* last output (udp) packet        */

int         wait_for_reply(int, struct sockaddr_in *, int);
void        send_probe(int, int);
double      deltaT(struct timeval *, struct timeval *);
int         packet_ok(u_char *, int, struct sockaddr_in *, int);
void        print(u_char *, int, struct sockaddr_in *);
void        tvsub(struct timeval *, struct timeval *);
char        *inetname(struct in_addr);
void        usage();

int s;                          /* receive (icmp) socket file desc. */
int sndsock;                    /* send (udp) socket file descriptor    */
struct timezone tz;             /* leftover                         */

struct sockaddr whereto;        /* Who to try to reach              */
int datalen;                    /* How much data                    */

char *source = 0;
char *hostname;

int nprobes = 3;
int max_ttl = 30;
u_short ident;
u_short port = 32768+666;       /* start udp dest port # for probe pkts   */
int options;                    /* socket options                         */
int verbose;
int waittime = 5;               /* time to wait for response (in seconds) */
int nflag;                      /* print addresses numerically            */

int main(argc, argv)
   int argc;
   char *argv[];
{
   extern char *optarg;
   extern int optind;
   struct hostent *hp;
   struct protoent *pe;
   struct sockaddr_in from, *to;
   int ch, i, on, probe, seq, tos, ttl;

      :
      :
   ident =  ident = (getpid() & 0xffff) | 0x8000;  /* Source port */
```

Create the socket used for reading the ICMP replies.

```
  if ((pe = getprotobyname("icmp")) == NULL) {
      Fprintf(stderr, "icmp: unknown protocol\n");
      exit(10);
   }
   if ((s = socket(AF_INET, SOCK_RAW, pe->p_proto)) < 0) {
      perror("traceroute: icmp socket");
      exit(5);
   }
```

Create the socket used for sending the UDP datagrams..
In this case (unlike ping) it is necessary that IP headers be built by the application.

```
  if ((sndsock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)  {
      perror("traceroute: raw socket");
      exit(5);
   }
```

The following stuff is required to adapt to the differences among sockets of the world.. Solaris sockets support / require? *IP_HDRINCL* but Linux sockets don't ! Your mileage may vary ... see socket.h and ip.h and man setsockopt for your particular environment.

```
#ifdef SO_SNDBUF
   if (setsockopt(sndsock, SOL_SOCKET, SO_SNDBUF, (char *)&datalen,
              sizeof(datalen)) < 0) {
      perror("traceroute: SO_SNDBUF");
      exit(6);
   }
#endif SO_SNDBUF
#ifdef IP_HDRINCL
   if (setsockopt(sndsock, IPPROTO_IP, IP_HDRINCL, (char *)&on,
              sizeof(on)) < 0) {
      perror("traceroute: IP_HDRINCL");
      exit(6);
   }
#endif IP_HDRINCL
   if (options & SO_DEBUG)
      (void) setsockopt(sndsock, SOL_SOCKET, SO_DEBUG,
               (char *)&on, sizeof(on));
   if (options & SO_DONTROUTE)
      (void) setsockopt(sndsock, SOL_SOCKET, SO_DONTROUTE,
               (char *)&on, sizeof(on));

   if (source) {
      (void) bzero((char *)&from, sizeof(struct sockaddr));
      from.sin_family = AF_INET;
      from.sin_addr.s_addr = inet_addr(source);
      if (from.sin_addr.s_addr == -1) {
         Printf("traceroute: unknown host %s\n", source);
         exit(1);
      }
      outpacket->ip.ip_src = from.sin_addr;
#ifndef IP_HDRINCL
      if (bind(sndsock, (struct sockaddr *)&from, sizeof(from)) < 0) {
         perror ("traceroute: bind:");
         exit (1);
      }
#endif IP_HDRINCL
   }
```

Send the UDP packet toward the traceroute target.

```
void send_probe(int seq,  int ttl)
{
   struct opacket       *op = outpacket;
   struct ip            *ip = &op->ip;
   struct udphdr        *up = &op->udp;
   int                  i;
```

Fill in IP header and UDP header −−− note that no checksums are computed!
(Other fields were filled in in the main function.)

```
   ip->ip_off     = 0;
   ip->ip_hl      = sizeof(*ip) >> 2;
   ip->ip_p       = IPPROTO_UDP;
   ip->ip_len     = datalen;
   ip->ip_ttl     = ttl;
   ip->ip_v       = IPVERSION;
   ip->ip_id      = htons(ident+seq);

   up->uh_sport   = htons(ident);
   up->uh_dport   = htons(port+seq);
   up->uh_ulen    = htons((u_short)(datalen - sizeof(struct ip)));
   up->uh_sum     = 0;

   op->seq  = seq;
   op->ttl  = ttl;

   (void)gettimeofday(&op->tv, &tz);

   i = sendto(sndsock, (char *)outpacket,
                  datalen, 0, &whereto,  sizeof(struct sockaddr));
```

The value returned by *sendto* is the number of bytes actually transmitted.
It should *always* be verified that it is correct.

```
   if (i < 0 || i != datalen)
   {
      if (i<0)
         perror("sendto");
      Printf("traceroute: wrote %s %d chars, ret=%d\n", hostname,
         datalen, i);
      (void) fflush(stdout);
   }
}
```

Receive the ICMP reply

```
int wait_for_reply(
int                sock,
struct sockaddr_in *from,
int                reset_timer)
{
   fd_set fds;
   static struct timeval wait;
   int cc = 0;
   int fromlen = sizeof (*from);

   FD_ZERO(&fds);
   FD_SET(sock, &fds);
```

Traceroute could hang if someone else has a ping running and our ICMP reply gets dropped but we don't realize it because we keep waking up to handle those other ICMP packets that keep coming in. To fix this, "reset_timer" will only be true if the last packet that came in was for us or if this is the first time we're waiting for a reply since sending out a probe. Note that this takes advantage of the select() feature on Linux where the remaining timeout is written to the struct timeval area.

```
   if (reset_timer) {
      wait.tv_sec = waittime;
      wait.tv_usec = 0;
   }

   if (select(sock+1, &fds, (fd_set *)0, (fd_set *)0, &wait) > 0)
      cc=recvfrom(s, (char *)packet, sizeof(packet), 0,
             (struct sockaddr *)from, &fromlen);

   return(cc);
}
```

Verify that the packet received was destination unreachable because TTL reached 0.

The received data should have the following layout:

IP header used to return the ICMP message
ICMP header (8 bytes)
ICMP error data (at least 28 bytes)
Original IP header
Firstt 8 bytes of data (the UDP header)

```
int packet_ok(buf, cc, from, seq)
   u_char *buf;
   int cc;
   struct sockaddr_in *from;
   int seq;
{
   register struct icmp *icp;
   u_char type, code;
   int hlen;
   struct ip *ip;
```

Verify that the packet is sufficiently long (*cc* maintains the remaining packet length)

```
   ip = (struct ip *) buf;
   hlen = ip->ip_hl << 2;
   if (cc < hlen + ICMP_MINLEN) {
      if (verbose)
         Printf("packet too short (%d bytes) from %s\n", cc,
            inet_ntoa(from->sin_addr));
      return (0);
   }
```

Adjust length, set the ICMP header pointer, and extract ICMP type and code.

```
   cc -= hlen;
   icp = (struct icmp *)(buf + hlen);

   type = icp->icmp_type;
   code = icp->icmp_code;
```

Acceptable type/code combinations include:

Time exceeded in transit (hop counter reached 0)

Destination unreachable (hopefully port unreachable on last hop)

Under these error conditions a further check is made to verify that the header returned is the one that was sent.

```
   if ((type == ICMP_TIMXCEED && code == ICMP_TIMXCEED_INTRANS) ||
        type == ICMP_UNREACH) {
      struct ip *hip;
      struct udphdr *up;

      hip = &icp->icmp_ip;
      hlen = hip->ip_hl << 2;
      up = (struct udphdr *)((u_char *)hip + hlen);
      if (hlen + 12 <= cc && hip->ip_p == IPPROTO_UDP &&
          up->uh_sport == htons(ident) &&
          up->uh_dport == htons(port+seq))
         return (type == ICMP_TIMXCEED? -1 : code+1);
   }
   if (verbose) {
      int i;
      u_long *lp = (u_long *)&icp->icmp_ip;
      Printf("\n%d bytes from %s to %s", cc,
         inet_ntoa(from->sin_addr), inet_ntoa(ip->ip_dst));
      Printf(": icmp type %d (%s) code %d\n", type, pr_type(type),
             icp->icmp_code);
      for (i = 4; i < cc ; i += sizeof(long))
         Printf("%2d: x%8.8lx\n", i, *lp++);
   }
   return(0);
}
```

An alternative approach to traceroute: RFC 1393

Traceroute Using an IP Option

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Traceroute serves as a valuable network debugging tool. The way in which it is currently implemented has the advantage of being automatically supported by all of the routers. It's two problems are the number of packets it generates and the amount of time it takes to run.

This document specifies a new IP option and ICMP message type which duplicates the functionality of the existing traceroute method while generating fewer packets and completing in a shorter time.

1. Traceroute Today

The existing traceroute operates by sending out a packet with a Time To Live (TTL) of 1. The first hop then sends back an ICMP [1] error message indicating that the packet could not be forwarded because the TTL expired. The packet is then resent with a TTL of 2, and the second hop returns the TTL expired. This process continues until the destination is reached. The purpose behind this is to record the source of each ICMP TTL exceeded message to provide a trace of the path the packet took to reach the destination.

The advantage of this algorithm, is that every router already has the ability to send TTL exceeded messages. No special code is required. The disadvantages are the number of packets generated (2n, where n is the number of hops), the time it takes to duplicate all the nearer hops with each successive packet, and the fact that the path may change during this process. Also, this algorithm does not trace the return path, which may differ from the outbound path.

2. Traceroute Tomorrow

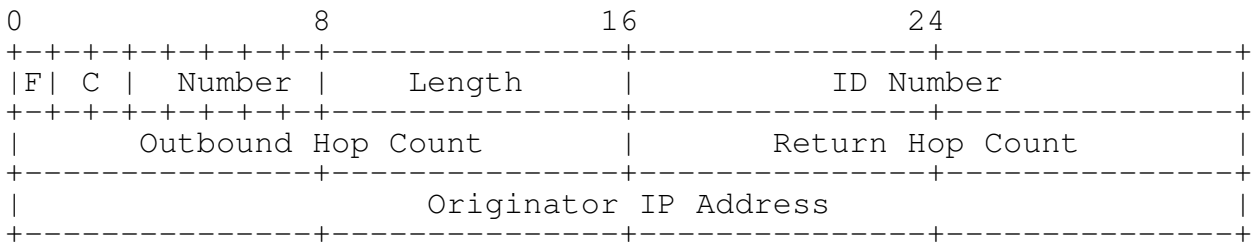
The proposed traceroute would use a different algorithm to achieve the same goal, namely, to trace the path to a host. Because the new traceroute uses an ICMP message designed for the purpose, additional information, unavailable to the original traceroute user, can be made available.

2.1 Basic Algorithm

A new IP Traceroute option will be defined. The presence of this option in an ICMP Echo (or any other) packet, hereinafter referred to as the Outbound Packet, will cause a router to send the newly defined ICMP Traceroute message to the originator of the Outbound Packet. In this way, the path of the Outbound Packet will be logged by the originator with only n+1 (instead of 2n) packets. This algorithm does not suffer from a changing path and allows the response to the Outbound Packet, hereinafter refered to as the Return Packet, to be traced (provided the Outbound Packet's destination preserves the IP Traceroute option in the Return Packet).

The disadvantage of this method is that the traceroute function will have to be put into the routers. To counter this disadvantage, however, is the fact that this mechanism may be easily ported to a new IP version.

2.2 IP Traceroute option format



F (copy to fragments)

0 (do not copy to fragments)

C (class)

2 (Debugging & Measurement)

Number

18 (F+C+Number = 82)

ID Number

An arbitrary number used by the originator of the Outbound Packet to identify the ICMP Traceroute messages. It is NOT related to the ID number in the IP header.

Originator IP Address

The IP address of the originator of the Outbound Packet. This is needed so the routers know where to send the ICMP Traceroute message for Return Packets. It is also needed for Outbound Packets which have a Source Route option.

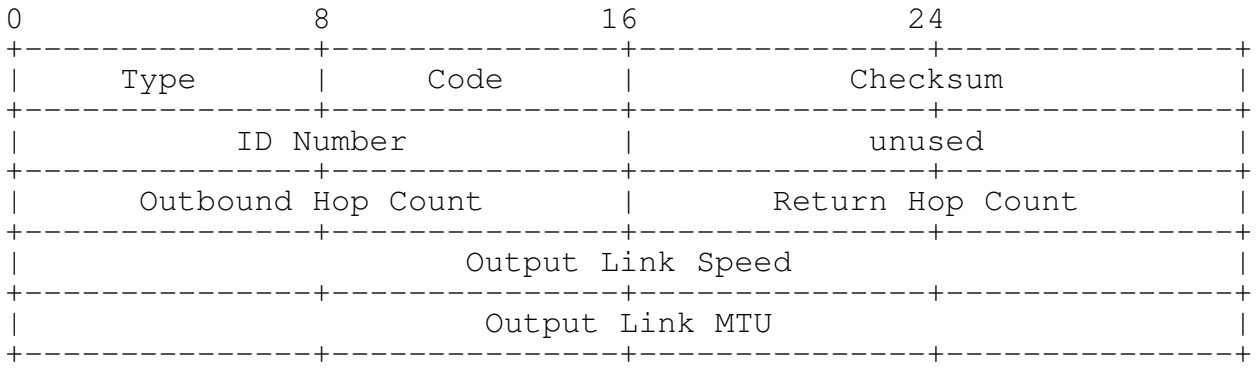
Outbound Hop Count (OHC)

The number of routers through which the Outbound Packet has passed. This field is not incremented by the Outbound Packet's destination.

Return Hop Count (RHC)

The number of routers through which the Return Packet has passed This field is not incremented by the Return Packet's destination.

2.3 ICMP Traceroute message format



Type

30

Code

0 – Outbound Packet successfully forwarded
1 – No route for Outbound Packet; packet discarded

Checksum

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero.

ID Number

The ID Number as copied from the IP Traceroute option of the packet which caused this Traceroute message to be sent. This is NOT related to the ID number in the IP header.

Outbound Hop Count

The Outbound Hop Count as copied from the IP Traceroute option of the packet which caused this Traceroute message to be sent.

Return Hop Count

The Return Hop Count as copied from the IP Traceroute option of the packet which caused this Traceroute message to be sent.

Output Link Speed

The speed, in OCTETS per second, of the link over which the Outbound/Return Packet will be sent. Since it will not be long before network speeds exceed 4.3Gb/s, and since some machines deal poorly with fields longer than 32 bits, octets per second was chosen over bits per second. If this value cannot be determined, the field should be set to zero.

Output Link MTU

The MTU, in bytes, of the link over which the Outbound/Return Packet will be sent. MTU refers to the data portion (includes IP header; excludes datalink header/trailer) of the packet. If this value cannot be determined, the field should be set to zero.

3. Protocol

The Outbound Packet which is used to carry the IP Traceroute option should use no special Type Of Service (TOS) or Precedence, unless the purpose is to trace the path of packets with special TOS or Precedence values.

The TTL of the Outbound Packet should be set to the default value specified in "Assigned Numbers" [2].

3.1 Hop Counts

The hop counts ultimately provide information on the length of the outbound and return paths to the destination. They also provide a means of determining whether or not any ICMP Traceroute messages have been lost. For example, if a Traceroute message with an OHC of 4 is followed by a message with an OHC of 6, then the the message with an OHC of 5 was lost. This is why simply counting Traceroute messages is not sufficient for determining path length.

The originator of the Outbound Packet should set the OHC to zero and the RHC to 0xFFFF. 0xFFFF is a special value which indicates to routers that the packet is an Outbound Packet rather than a Return Packet (which begins with an RHC of zero).

It is important to note that the Traceroute hop counts are NOT related to the IP TTL. A hop count should only be incremented when an ICMP Traceroute message is sent.

3.2 Destination Node Operation

When a node receives an Outbound Packet with an IP Traceroute option, the Return Packet, if such is required (e.g., ICMP Echo Request/Reply), should also carry that option. The values in the ID Number, OHC, and Originator Address fields should be copied into the Return Packet. The value of the RHC field should be set to zero.

The destination should NOT increment any hop counts or send any ICMP Traceroute messages.

3.3 Router Operation

When a router forwards a packet with an IP Traceroute option, it should send an ICMP Traceroute message to the host in the Originator IP Address field of the option. If the value of the RHC field is 0xFFFF then the packet is an Outbound Packet and the OHC should be incremented; otherwise, the RHC field should be incremented. The Traceroute message should reflect the incremented hop count. The Output Link Speed field should be set to the speed, in OCTETS per second, of the link over which the Outbound/Return Packet will be sent (e.g., 1,250,000 for an Ethernet) or zero if the output link speed cannot be determined. The Output Link MTU field should be set to the MTU of the link over which the Outbound/Return Packet will be sent or zero if the MTU cannot be determined.

The Outbound/Return Packet should be forwarded as though the Traceroute option did not exist; that is, it should take the same path to the destination as an optionless packet.

The ICMP Traceroute message should have the same TOS and Precedence values as the Outbound/Return Packet. The TTL should be set to the default defined in "Assigned Numbers". The ICMP Traceroute message should not carry the IP Traceroute option.

If the Outbound Packet cannot be forwarded, the ICMP Traceroute message should have a Code value of 1. If the Return Packet cannot be forwarded because there is no route, then there is no need to send a Traceroute message since it could not be forwarded either.

IP Source Routing

Permits source to specify the route to be taken
Strictly
Each router must appear in list
Loosely
Packet must traverse specified routers in order
but may pass through intermediaries

Source route option
Laid out as follows:

Code = 0x83	Len = N	Ptr = 4	IP addr1	IP addr2	IP addr3

Code = 0x89	Len = N	Ptr = 4	IP addr1	IP addr2	IP addr3

Source host (according to text)
Removes first address from the list and makes it the datagram "destination"
Shifts remaining addresses left leaving the pointer at 4
Adds original destination to the source route list.

In reality, I know of no API that does this...

In practice, both the destination IP address and the operand of *sendto()* must be set to the first intermediate hop.. The remainder of the hops fill the source route list with the ultimate destination being last)

Non–destination router (won't happen with strict source route)
Just forward the datagram

Destination router
Next address in list becomes new "destination"
Pointer is incremented by 4
Outgoing interface becomes the new source address (according to text)

Reversed route can be used by the destination to contact the source.
How so if source host trashed the original host address!
(In practice there is no evidence to support this theory of dynamic sourcing)

Source routing can lead to some nasty security problems
It allows someone to masquerade as a trusted host
Some installations disable its use.