Today

- ip options
- icmp
- raw sockets
- ping, traceroute
- mtu discovery

IP header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

see RFC 791

IP options

- mostly unused
- extends header up to 60 bytes
- IHL greater than 5
- multiple of 4-bytes
- setsockopt() IP_OPTIONS
- dropped by IPv6

```
setsockopt(s, IPPROTO_IP, IP_OPTIONS, srr, optsize);
```


IP options

```
format: type byte   or   type lth data

type
 0      end of options
 1      no-op
 2      security   11 byte length
 3      loose source routing, variable length
 9      strict source routing, variable length
 7      record route, variable length, 9 addressses max
68      timestamp, variable length, 4 addr/time max
```

- security option specifies packet classification
- ping has record-route option (-vR)
- source routing is dangerous, blocked by routers

adding options

from SunOS ping.c

```
char srr[ROUTE_SIZE + 1];
int optsize = sizeof(srr);

bzero(srr, sizeof(srr));
srr[0] = IPOPT_RR;
srr[1] = ROUTE_SIZE;
srr[2] = IPOPT_MINOFF;
if (setsockopt(s, IPPROTO_IP, IP_OPTIONS,
   srr, optsize) < 0) perror("IP Options");
```

```
+--------+--------+--------+---------//--------+
|00000111| length | pointer|     route data    |
+--------+--------+--------+---------//--------+
  Type=7
```


record route

```
on deneb: ping -vR www.utk.edu
  64 bytes from CHEKOV.IT.UTK.EDU (128.169.53.15): icmp_seq=3. time=4. ms
  IP options:  <record route> 192.168.101.40, 128.169.53.1,
   CHEKOV.IT.UTK.EDU (128.169.53.15), 192.168.101.1,
   R5HM01V277.NS.UTK.EDU (128.169.92.1), DENEB.CS.UTK.EDU (128.169.94.74),
   0.0.0.0(Current), 0.0.0.0, 0.0.0.0

on altair: tcpdump -N -x -t icmp
DENEB > CHEKOV: icmp: echo request
          4f00 007c 7aa3 0000 ff01 970a 80a9 5e4a
          80a9 350f 0727 0400 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0800 e373
          457c 0003 37f2 4c3c 0004 5fd7 0809 0a0b
          0c0d
CHEKOV > DENEB: icmp: echo reply (DF)
          4f00 007c 36dd 4000 fd01 ff3c 80a9 350f
          80a9 5e4a 0727 18c0 a865 2880 a935 0180
          a935 0fc0 a865 0180 a95c 0100 0000 0000
          0000 0000 0000 0000 0000 0000 0000 eb73
          457c 0003 37f2 4c3c 0004 5fd7 0809 0a0b
          0c0d
```


IP header

modifiable fields

- with setsockopt()
  - IP_TOS
  - IP_TTL
  - IP_OPTIONS
- with raw sockets (privileged), everything but checksum!

ICMP

Internet Control Message Protocol
RFC 792

- rides on IP, not really transport protocol
- part of IP (proto=1)
- provides feedback (port unreachable)
- error types carry offending IP header
- no ICMP messages sent about ICMP messages
- generated by kernel/routers (and ping)
- ICMP packet includes type and checksum
- unreliable

```
Type    meaning
 0      echo reply
 3      destination unreachable (net,host,port,frag needed)
 4      source quench (shutup)
 5      redirect
 8      echo request
11      time exceeded (TTL went to 0)
12      parameter problem (IP options etc.)
13      timestamp   (14 reply)

see #include <netinet/ip_icmp.h>
```

ICMP packets

```
   16       20      16+                      4
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+-+-+
|ether  |    IP   | ICMP header and data| CRC |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ... -+-+-+-+


  Destination unreachable: type=3
   code: 0=net, 1=host, 2=protocol, 3=port, 4=frag need
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


ICMP tcpdump

- UDP sendto to non-listening port results in ICMP port unreachable
- connected UDP socket will get error on next read/recv

```
   whisper (ttcp) sends UDP to port 5002 on altair

WHISPER.1343 > ALTAIR.5002: udp 1024
          4500 041c 220a 0000 4011 9875 80a9 5dc8
          80a9 5d37 053f 138a 0408 1c98 2021 2223
          2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
          3435 3637 3839 3a3b 3c3d 3e3f 4041 4243
ALTAIR > WHISPER: icmp: ALTAIR udp port 5002 unreachable
          4500 0038 6f2c 0000 ff01 9046 80a9 5d37
          80a9 5dc8 0303 c393 0000 0000 4500 041c
          220a 0000 4011 9875 80a9 5dc8 80a9 5d37
          053f 138a 0408 1c98
```


ICMP packets

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Originate Timestamp                                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Receive Timestamp                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Transmit Timestamp                                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                      ping
```


tcpdump ping

from cetus1a: ping altair

```
 tcpdump -e -x -t -N icmp
CETUS1A > ALTAIR: icmp: echo request (DF)
         4500 0054 92d3 4000 ff01 2c36 80a9 5e15
         80a9 5d37 0800 fa7f 1dd0 0003 37f0 0fdc
         000d acd0 0809 0a0b 0c0d 0e0f 1011 1213
         1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
         2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
         3435
ALTAIR > CETUS1A: icmp: echo reply (DF)
         4500 0054 36d3 4000 ff01 8836 80a9 5d37
         80a9 5e15 0000 0280 1dd0 0003 37f0 0fdc
         000d acd0 0809 0a0b 0c0d 0e0f 1011 1213
         1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
         2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
         3435
```

ping

- Packet InterNet Groper
- reachability test using ICMP ECHO
- some performance feedback (latency, loss)
- requires no server or processes
- host may respond to ping and still be hosed
- ping.c uses raw sockets

raw sockets

- need privileges (root)
- application can read protos icmp, igmp packets (any with IPPROTO_RAW)
- can read/write IPv4 datagrams with a proto field not supported by the kernel
- with IP_HDRINCL socket option, can build IP header! popular with hackers. kernel calculates checksum
- on read, don't get UDP/TCP, but most ICMP, all IGMP, all IP datagrams with unsupported proto fields – watch out, you'll get more packets than you expect

sd=socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);

ping program

- setup raw socket
- build ICMP echo request packet, calculate checksum too
- send it to destination every second
- receive ICMP's, pick out our ICMP echo replies
- need timers and sequence numbers, since unreliable
- example of IP_OPTIONS socket option for record route

Stevens ping.c is simpler

ping: main.c

```
main(argc, argv)
int   argc;
char  **argv;
{
   bzero((char *) &dest, sizeof(dest));
   dest.sin_family = AF_INET;
    .... process args
   if ( (dest.sin_addr.s_addr = inet_addr(argv[0])) != INADDR_NONE) {
      strcpy(hnamebuf, argv[0]);
      hostname = hnamebuf;
      destdotaddr = NULL;
   } else {
      if ( (host = gethostbyname(argv[0])) == NULL) {
         err_quit("host name error: %s %s",
                  argv[0], host_err_str());
      }
      dest.sin_family = host->h_addrtype;
      bcopy(host->h_addr, (caddr_t) &dest.sin_addr, host->h_length);
      hostname = host->h_name;
      destdotaddr = inet_ntoa(*(struct in_addr *)&dest.sin_addr.s_addr);
   }
```

ping: main.c

```
   ident = getpid() & 0xffff;
   if ( (proto = getprotobyname("icmp")) == NULL)
      err_quit("unknown protocol: icmp");
   if ( (sockfd = socket(AF_INET, SOCK_RAW, proto->p_proto)) < 0)
      err_sys("can't create raw socket");
   if (sockoptions & SO_DEBUG)
      if (setsockopt(sockfd, SOL_SOCKET, SO_DEBUG, &on,
                        sizeof(on)) < 0)
         err_sys("setsockopt SO_DEBUG error");
   if (sockoptions & SO_DONTROUTE)
      if (setsockopt(sockfd, SOL_SOCKET, SO_DONTROUTE, &on,
                        sizeof(on)) < 0)
         err_sys("setsockopt SO_DONTROUTE error");
   signal(SIGINT, sig_finish);   /* to let user stop program */
   signal(SIGALRM, sig_alarm);   /* invoked every second */

   sig_alarm();   /* start the output going */

   recv_ping();   /* and start the receive */

   /* NOTREACHED */
}
```


ping: sendping.c

```
sig_alarm()
{
   int   waittime;
   send_ping();      /* first send another packet */
   if (npackets == 0 || ntransmitted < npackets)
     alarm(1);
...
}

send_ping()
{
   icp = (struct icmp *) sendpack;  /* pointer to ICMP header */
   icp->icmp_type  = ICMP_ECHO;
   icp->icmp_code  = 0;
   icp->icmp_cksum = 0;      /* init to 0, then call in_cksum() below */
   icp->icmp_id    = ident;     /* our pid, to identify on return */
   icp->icmp_seq   = ntransmitted++;   /* sequence number */
   if (timing)
      gettimeofday((struct timeval *) &sendpack[SIZE_ICMP_HDR],
              (struct timezone *) 0);
   icp->icmp_cksum = in_cksum(icp, packsize);
   i = sendto(sockfd, sendpack, packsize, 0,
         (struct sockaddr *) &dest, sizeof(dest));
   if (i < 0 || i != packsize)  {
      if (i < 0) err_ret("sendto error");
      else err_ret("wrote %s %d bytes, return=%d",
                  hostname, packsize, i);
   }
}
```


ping: recvping.c

```
recv_ping()
{
   register int      n;
   int         fromlen;
   struct sockaddr_in   from;

   for ( ; ; ) {
      fromlen = sizeof(from);
      if ( (n = recvfrom(sockfd, recvpack, sizeof(recvpack), 0,
            (struct sockaddr *) &from, &fromlen)) < 0) {
         if (errno == EINTR)
            continue;   /* normal */
         err_ret("recvfrom error");
         continue;
      }
      pr_pack(recvpack, n, &from);
      if (npackets && (nreceived >= npackets))
         sig_finish();  /* does not return */
   }
}
```


ping: print.c

```
pr_pack(buf, cc, from)
char                   *buf;   /* ptr to start of IP header */
int                    cc;     /* total size of received packet */
struct sockaddr_in     *from;  /* address of sender */
{
   from->sin_addr = ntohl(from->sin_addr);
   ip = (struct ip *) buf;
   iphdrlen = ip->ip_hl << 2; /* convert # 16-bit words to #bytes */
   if (cc < iphdrlen + ICMP_MINLEN) {
      if (verbose)
         printf("packet too short (%d bytes) from %s\n", cc,
            inet_ntoa(ntohl(from->sin_addr)));
      return;
   }
   cc -= iphdrlen;

   icp = (struct icmp *)(buf + iphdrlen);
   if (icp->icmp_type != ICMP_ECHOREPLY)  return;
   if (icp->icmp_id != ident) return;
   printf("%d bytes from %s: ", cc,
               inet_ntoa(ntohl(from->sin_addr)));
   printf("icmp_seq=%d. ", icp->icmp_seq);
       tvsub(&tv, (struct timeval *) &icp->icmp_data[0]);
      triptime = tv.tv_sec * 1000 + (tv.tv_usec / 1000);
      printf("time=%d. ms", triptime); /* milliseconds */
   putchar('\n');
   nreceived++;   /* only count echo reply packets that we sent */
}
```

traceroute

trouble with ping

- record-route not supported by all routers
- not enough room in header

traceroute

- send a UDP packet (or ICMP-echo) with TTL of 1
- first router, decrements to 0, sends ICMP time-exceeded with routers IP address as source
- send UDP packet with TTL of 2, makes it 2nd router, which sends ICMP time-exceeded back
- and so on
- final destination receives, usually sends back ICMP port-unreachable
- need raw socket and socket option IP_HDRINCL to set TTL or IP_TTL option
- win* version: tracert
- slow printing due to DNS (try -n)
- one-way, return route may differ (traceroute server)

tcpdump of traceroute

```
traceroute to thdsun.epm.ornl.gov (134.167.12.186), 30 hops max, 40 byte packets 1  R5HM01V277.NS.UTK.EDU (128.169.92.1)  3 ms  2 ms  2 ms
 2  192.168.101.3 (192.168.101.3)  3 ms  3 ms  3 ms
 3  mmesgwya32.ctd.ornl.gov (192.31.96.17)  5 ms  6 ms  5 ms
 4  orgwyf1.ctd.ornl.gov (192.31.96.65)  4 ms  4 ms  5 ms
 5  atmgwy.ens.ornl.gov (198.124.42.8)  5 ms  5 ms  4 ms
 6  swge4500n.ens.ornl.gov (160.91.0.2)  7 ms  7 ms  7 ms
 7  swge6010.ens.ornl.gov (160.91.0.10)  6 ms  6 ms  7 ms
 8  thdsun.epm.ornl.gov (134.167.12.186)  5 ms  5 ms  5 ms
```

```
ALTAIR.47368 > thdsun.33435: udp 12 [ttl 1] (id 32461)
R5HM01V277 > ALTAIR: icmp: time exceeded in-transit [tos 0xc0] (ttl 255, id 6475)
...
ALTAIR.47368 > thdsun.33444: udp 12 (ttl 4, id 32474)
orgwyf1 > ALTAIR: icmp: time exceeded in-transit [tos 0xc0] (ttl 252, id 11097)
...
swge6010 > ALTAIR: icmp: time exceeded in-transit (ttl 58, id 27665)
ALTAIR.47368 > thdsun.33456: udp 12 (ttl 8, id 32490)
thdsun > ALTAIR: icmp: thdsun udp port 33456 unreachable (ttl 248, id 59340)
ALTAIR.47368 > thdsun.33457: udp 12 (ttl 8, id 32492)
```



```
ALTAIR.47372 > thdsun.33455: udp 12 (ttl 7, id 32560)
          4500 0028 7f30 0000 0711 c353 80a9 5d37
          86a7 0cba b90c 82af 0014 0000 1507 0000
          37f1 55e6 0006 d893 0000 0000 0000
swge6010 > ALTAIR: icmp: time exceeded in-transit (ttl 58, id 28797)
          4500 0038 707d 0000 3a01 9202 a05b 000a
          80a9 5d37 0b00 b92f 0000 0000 4500 0028
          7f30 0000 0111 c953 80a9 5d37 86a7 0cba
          b90c 82af 0014 0000
```


traceroute program

- get a raw socket for receiving ICMP's
- get a raw socket for sending TTL/UDP
- loop through increasing TTL's, send UDP,
- timed select() for recvfrom of ICMP's
- check type of ICMP, and verify it's our IP header

Steven's version uses setsockopt() of IP_TTL

traceroute.c

SunOS version, also see Steven's Ch 25

```
... process args
 (void) bzero((char *)&whereto, sizeof(struct sockaddr));
 to->sin_family = AF_INET;
 to->sin_addr.s_addr = inet_addr(av[0]);
 if (to->sin_addr.s_addr != -1) {
         (void) strcpy(hnamebuf, av[0]);
         hostname = hnamebuf;
 } else {
         hp = gethostbyname(av[0]);
         if (hp) {
            to->sin_family = hp->h_addrtype;
            bcopy(hp->h_addr, (caddr_t)&to->sin_addr,
               hp->h_length);
            hostname = hp->h_name;
         } else {
                 Printf("%s: unknown host %s\n", argv[0], av[0]);
                 exit(1);
         }
 }
```



```
    ident = (getpid() & 0xffff) | 0x8000;

    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);
    }
    if (sndsock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
           perror("traceroute: raw socket");
           exit(5);
   }
   if (setsockopt(sndsock, IPPROTO_IP, IP_HDRINCL, (char *)&on,
                  sizeof(on)) < 0) {
           perror("traceroute: IP_HDRINCL");
           exit(6);
   }
```



```
 for (ttl = 1; ttl <= max_ttl; ++ttl) {
         for (probe = 0; probe < nprobes; ++probe) {
            (void) gettimeofday(&tv, &tz);
            send_probe(++seq, ttl);
            while (cc = wait_for_reply(s, &from)) {
                    if ((i = packet_ok(packet, cc, &from, seq))) {
                            int dt = deltaT(&tv);
              ... switch on ICMP_type
                         case ICMP_UNREACH_PORT:
                       ...
```



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

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

```
packet_ok(buf, cc, from, seq)
{
...
    for ICMP_TIMXCEED, parse out returned IP header
    and verify it was the UDP packet we sent

  type = icp->icmp_type; code = icp->icmp_code;
  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);
  }


....
}
```


build IP and UDP headers

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

   ip->ip_off = 0;
   ip->ip_p = IPPROTO_UDP;
   ip->ip_len = datalen;
   ip->ip_ttl = ttl;

   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));
```


path MTU discovery

RFC 1191

- discover the minimum MTU on the set of hops between two hosts
- avoid fragmentation while using largest acceptable MTU
- host sends MTU-sized packet with DF bit set (don't fragment)
- intervening routers with smaller MTU sends back ICMP UNREACH-FRAG and its MTU
- source resends with this MTU-sized packet
- on some OS's and routers, trouble if ICMP filters

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |   Code = 4    |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           unused = 0          |         Next-Hop MTU          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Datagram Data      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Stevens has a traceroute.pmtu

next time

UDP

TCP