Today

IPv6

- goals
- architecture
- programming
- deployment

IPv6 goals

next generation IP (IPng)

- larger addresses (128 bits)
- improved performance
  - routing
  - header processing
  - flow control
  - priority (QoS)
- security
- roaming host support
- better multicast
- interoperability with IPv4

IPv6 design

- CIDR was helping routing/addressing ('87)
- OSI – was it the solution?
- IETF call for proposals ('90), SIPP, TUBA
- SIPP became IPv6
- debates on address size, TTL, checksums, mobile hosts, security
- MUST interoperate

IPv6 architecture

- 128-bit addresses
- routing based on prefixes rather than classes
- fixed header, with optional sub-headers
- security (IPsec), authentication/encryption
- QoS fields
- methods to coexist with IPv4

implementation

to make it happen

- routers need to change
- DNS needs to change
- OS kernels need to change
- utilities need to change (ifconfig, netstat, ping, ARP)
- network apps need to change

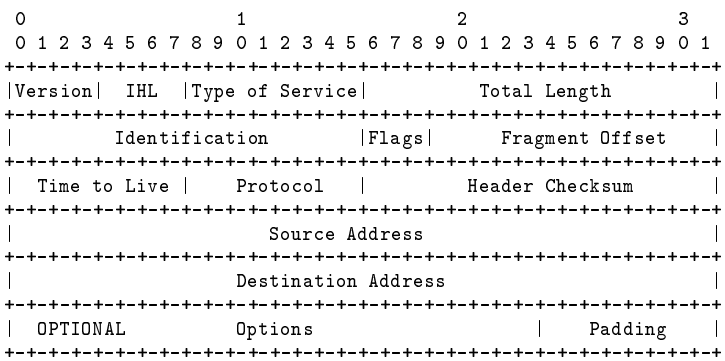
yet must coexist with v4

IPv6 header

- fixed length, 40 bytes (vs 20)
- 7 fields (vs 13)
- no checksum (UDP checksum required)
- no fragmentation (at routers)

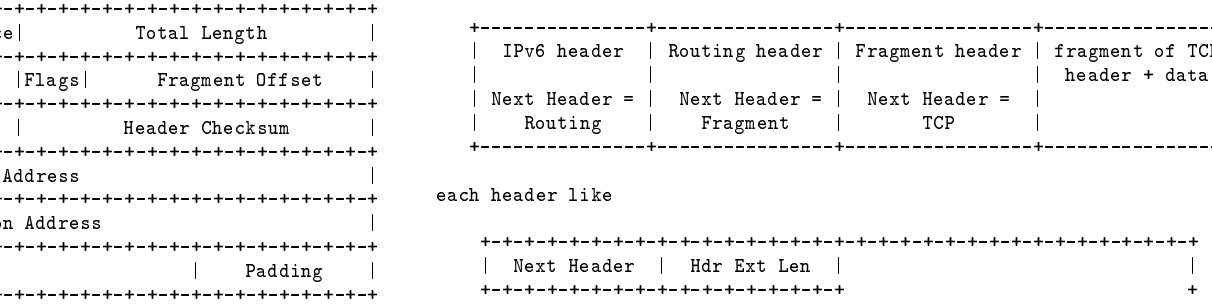
```
vers   6
priority   0-7 can be flow-controlled, 8-15 realtime,constant rate
   routers can handle congestion better
   1 for news, 4 ftp, 6 telnet
flow label experimental, end-to-end pseudoconnection, routers
   can lookup to find characteristics of flow (reserve
   bandwidth, minimize latency etc.
payload lth  doesn't count fixed header (16 bit)
next header protocol type of subheaders
hop limit   ttl
src/dst addresses   128-bit each
```


IPv4 header

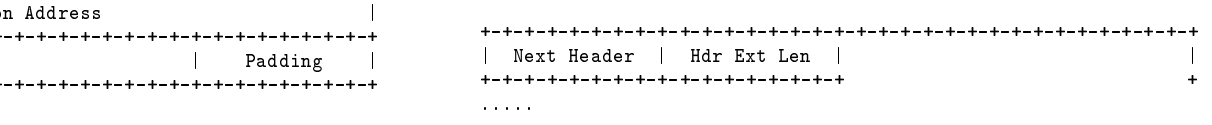


IPv6 headers

fixed header plus optional headers



each header like



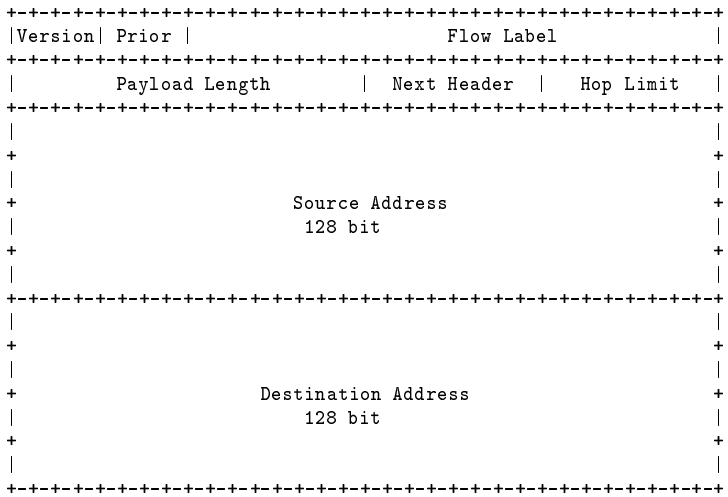
extension headers

preferably in this order

- hop-by-hop options – info for routers, jumbogram
- routing – loose/strict source routing
- fragmentation – end-to-end fragmentation
- authentication (AH)
- encrypted (ESP)
- destination options

IPv6 header

no variable size headers, no frags, no checksum



header debates

- bigger ttl
- payload length, jumbogram option
- checksum
- address length (64 bits, 160/osi)

IPv6 addresses

- 128-bit addresses
- 10**23 address per square meter
- prefixes for efficient routing
  provider based, geographic, link local, site local, multicast, IPv4
- denoted in hex
  8000:0000:0000:0000:0123:4567:89AB:CDEF
- link local uses MAC/ether address
- IPv4 ::128.169.93.44
- NO broadcast

```
netstat -r -n
  Destination      Gateway            Flags     Refs     Use PathMtu
  v6-in-v4 default 128.219.0.1        UCST        0        0    1480
  ::1              ::1                UH          0        0   32768
  ::128.219.8.0    link#1             UC          0        0    1500
  fe80::20:afc0:55a0 fe80::20:afc0:55a0 UH          0        0    15
  ff00::0          link#1             U           0        2    1500
```

ARP

- no broadcast, use multicast
- ARP replaced with multicast neighbor discovery
- also router discovery
- DHCP has to change too

neighbor discovery (multicast ICMP)

```
#define ETHERTYPE_IPV6          0x86DD  /* IPv6 protocol */

from smt1,  ping ::128.219.8.1

08002022fb93 >333380db0801 86dd   86
 6f 00 00 00  00 20 3a ff  fe 80 00 00  00 00 00 01    |o???? :??????????|
 00 00 08 00  20 22 fb 93  ff 02 00 00  00 00 00 00    |???? "??????????|
 00 00 00 01  80 db 08 01  87 00 20 f9  00 00 00 00    |?????????? ?????|
 00 00 00 00  00 00 00 00  00 00 00 00  80 db 08 01    |????????????????|
 01 01 08 00  20 22 fb 93    |???? "??|
```


tunneling

- tunnel ipv6 over ipv4 (proto 41)
- 6BONE, connect islands of ipv6 subnets over ipv4 routers
- or static routes on ipv6 hosts

```
ping ::204.123.39.2
smt1.ctd.ornl.gov > sipper.pa-x.dec.com: ip-proto-41 104 (DF)
          4500 007c 81e9 4000 3b29 b0ca 80db 984c
          cc7b 2702 6000 0000 0040 3aff 0000 0000
          0000 0000 0000 0000 80db 984c 0000 0000
          0000 0000 0000 0000 cc7b 2702 8000 86b3
          01eb 0000 31c9 3f39 000a 8e31 0809 0a0b
          0c0d 0e0f 1011 1213 1415 1617 1819 1a1b
          1c1d 1e1f 2021 2223 2425 26
sipper.pa-x.dec.com > smt1.ctd.ornl.gov: ip-proto-41 104
          4500 007c e5ae 0000 3629 9205 cc7b 2702
          80db 984c 6f00 0000 0040 3a40 0000 0000
          0000 0000 0000 0000 cc7b 2702 0000 0000
          0000 0000 0000 0000 80db 984c 8100 85b3
          01eb 0000 31c9 3f39 000a 8e31 0809 0a0b
          0c0d 0e0f 1011 1213 1415 1617 1819 1a1b
          1c1d 1e1f 2021 2223 2425 26
```


ftp tunnel

```
ftp ::204.123.39.2

smt1.ctd.ornl.gov > sipper.pa-x.dec.com: ip-proto-41 64 (DF)
          4500 0054 a2bd 4000 3b29 901e 80db 984c
          cc7b 2702 6000 0000 0018 06ff 0000 0000
          0000 0000 0000 0000 80db 984c 0000 0000
          0000 0000 0000 0000 cc7b 2702 800a 0015
          024f 0b42 0000 0000 6002 2148 dcb0 0000
          0204 058c d03c 53
sipper.pa-x.dec.com > smt1.ctd.ornl.gov: ip-proto-41 64
          4500 0054 e5b6 0000 3229 9625 cc7b 2702
          80db 984c 6000 0000 0018 063c 0000 0000
          0000 0000 0000 0000 cc7b 2702 0000 0000
          0000 0000 0000 0000 80db 984c 0015 800a
          7ec4 2600 024f 0b43 6012 8100 dbaa 0000
          0204 0204 4960 2c
```


fragmentation

- routers don't fragment, send ICMP size exceeded
- hosts can fragment
- must support min MTU of 1280

```
fragment header    RFC 2460

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |   Reserved    |      Fragment Offset    |Res|M|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Identification                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

class 24 talked about IPsec headers AH/ESP

IPv6 programming

- converting to IPv6
- ipv4/ipv6 client/servers

IPv6 mods

Stevens text

```
1.3         convert to ipv6
3.2         socket_addr for ipv6
3.7-.8      address to/from ascii
7.7-.8      ipv6 socket options
9.2-.6      DNS for ipv6
10          ipv6/v4 client/servers
11          DNS and wrappers
24.4-.7     ipv6 header options
25.3-.6     ipv6 raw sockets ping/traceroute
A.3         ipv6 heaer
A.5-.6      ipv6 address and ICMPv6
B.3         6BONE
```


to IPv6

converting to IPv6

- sockaddr_in6
- socket()
- opaque socket struct and length, connect, accept, recvfrom, sendto
- independent: read/write/close
- addresses: messy
  inet_pton/inet_ntop
  DNS: gethostbyname2/getaddrinfo/getnameinfo
- DNS has AAAA records

IPv6 server

```
main(int argc, char **argv)
{
   int                                     listenfd, connfd;
   socklen_t                       len;
   char                            buff[MAXLINE];
   time_t                          ticks;
   struct sockaddr_in6     servaddr, cliaddr;
   extern struct in6_addr  ipv6addr_any;

   listenfd = Socket(AF_INET6, SOCK_STREAM, 0);
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin6_family = AF_INET6;
   servaddr.sin6_addr   = ipv6addr_any;
   servaddr.sin6_port   = htons(13);       /* daytime server */
   Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
   Listen(listenfd, LISTENQ);
   for ( ; ; ) {
           len = sizeof(cliaddr);
           connfd = Accept(listenfd, (SA *) &cliaddr, &len);
           printf("connection from %s\n",
                      Sock_ntop((SA *) &cliaddr, len));

   ticks = time(NULL);
   snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
   Write(connfd, buff, strlen(buff));
           Close(connfd);
   }
}
```

IPv6 client

```
main(int argc, char **argv)

{
   int                                     sockfd, n;
   struct sockaddr_in6     servaddr;
   char                            recvline[MAXLINE + 1];

   if (argc != 2)
           err_quit("usage: a.out <IPaddress>");
   if ( (sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
           err_sys("socket error");
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin6_family = AF_INET6;
   servaddr.sin6_port   = htons(13);       /* daytime server */
   if (inet_pton(AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
           err_quit("inet_pton error for %s", argv[1]);
   if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
           err_sys("connect error");
   while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
           recvline[n] = 0;        /* null terminate */
           if (fputs(recvline, stdout) == EOF)
                   err_sys("fputs error");
   }
   if (n < 0)
           err_sys("read error");

        exit(0);
}
```


v4/v6 interoperability

```
   fig 10.5
 IPv4 host can communicate with:
     any ipv4 host on local net
     any ipv4 host via ipv4 router
     any ipv6 host with ipv4-mapped address
 IPv6 host (ipv6 only address) can communicate with:
     any ipv6 host on local net
     any ipv6 via ipv6 router on local net (may tunnel
      from router via ipv4)
 IPv6 host (ipv4-mapped address)
     any ipv6 host on local net
     any ipv6 host via ipv6 router on local net
     any ipv4 node
 IPv6/v6 host (ipv4 compatible address)
     any ipv4 or ipv6 on local net
     any ipv4 or ipv6 via ipv4 or ipv6 router
```


v6/v4 client/servers

dual-stacks

- listening ipv4 socket can accept connections only from ipv4 clients
- listening ipv6 socket bound to wildcard can accept v4 or v6 clients, ipv4 address will be mapped to ipv6 format
- ipv6 client to ipv4 only server, client kernel will see ipv4 address and use IPv4 packets
- ipv4 only client can't reach an ipv6 host
- can use getaddrinfo()/getnameinfo() to build v6/v4 socket based on address info
- stevens wrappers (ch 11) handle v4/v6

tcp_connect

```
tcp_connect(const char *host, const char *serv)
{
    int                             sockfd, n;
    struct addrinfo hints, *res, *ressave;

    bzero(&hints, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if ( (n = getaddrinfo(host, serv, &hints, &res)) != 0)
            err_quit("tcp_connect error for %s, %s: %s",
                             host, serv, gai_strerror(n));
    ressave = res;
    do {
      sockfd = socket(res->ai_family, res->ai_socktype, res->ai_proto
      if (sockfd < 0) continue;       /* ignore this one */
      if (connect(sockfd, res->ai_addr, res->ai_addrlen) == 0)
                    break;          /* success */
            Close(sockfd);  /* ignore this one */
    } while ( (res = res->ai_next) != NULL);

    if (res == NULL)        /* errno set from final connect() */
            err_sys("tcp_connect error for %s, %s", host, serv);
    freeaddrinfo(ressave);
    return(sockfd);
}

also see win* ipv6 ttcp.c in ~dunigan/cs494-unp/winsock/
```

deployment

- interoperability helps
- OS support: solaris, aix, *BSD, linux
- many servers can support v4/v6
- DNS has AAAA recs
- tunneling via IPv4 (6BONE) for many years
- 6REN being constructed, ipv6 routing
- IPsec progressing
- need experience with QoS, jumbograms

address exhaustion is the driving force, since many of the IPv6 features are available in v4

next time

XTI

high speed nets (ATM, VIA)