IPv6

Ch. 12

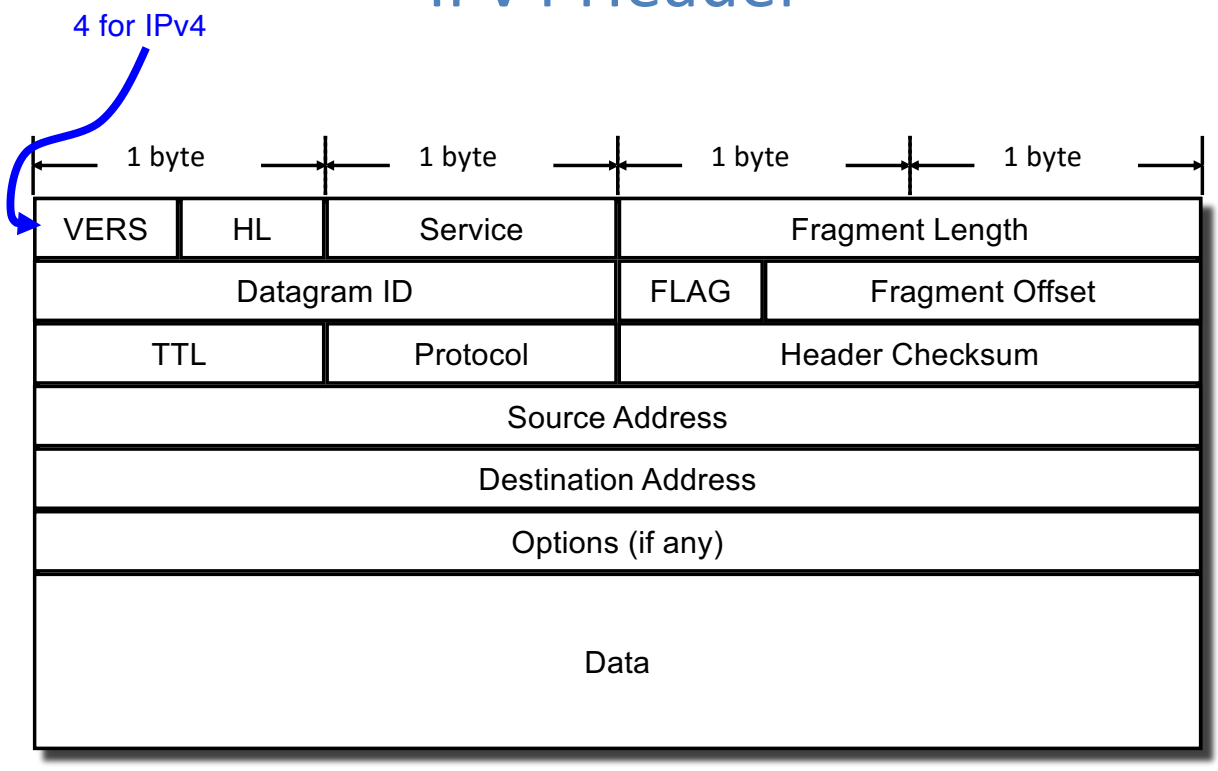
IPv6 Design

- Overcome IPv4 scaling problem with the address space exhaustion
- Flexible transition mechanism
- New routing capabilities
- Quality of service
- Security
- Ability to add features in the future

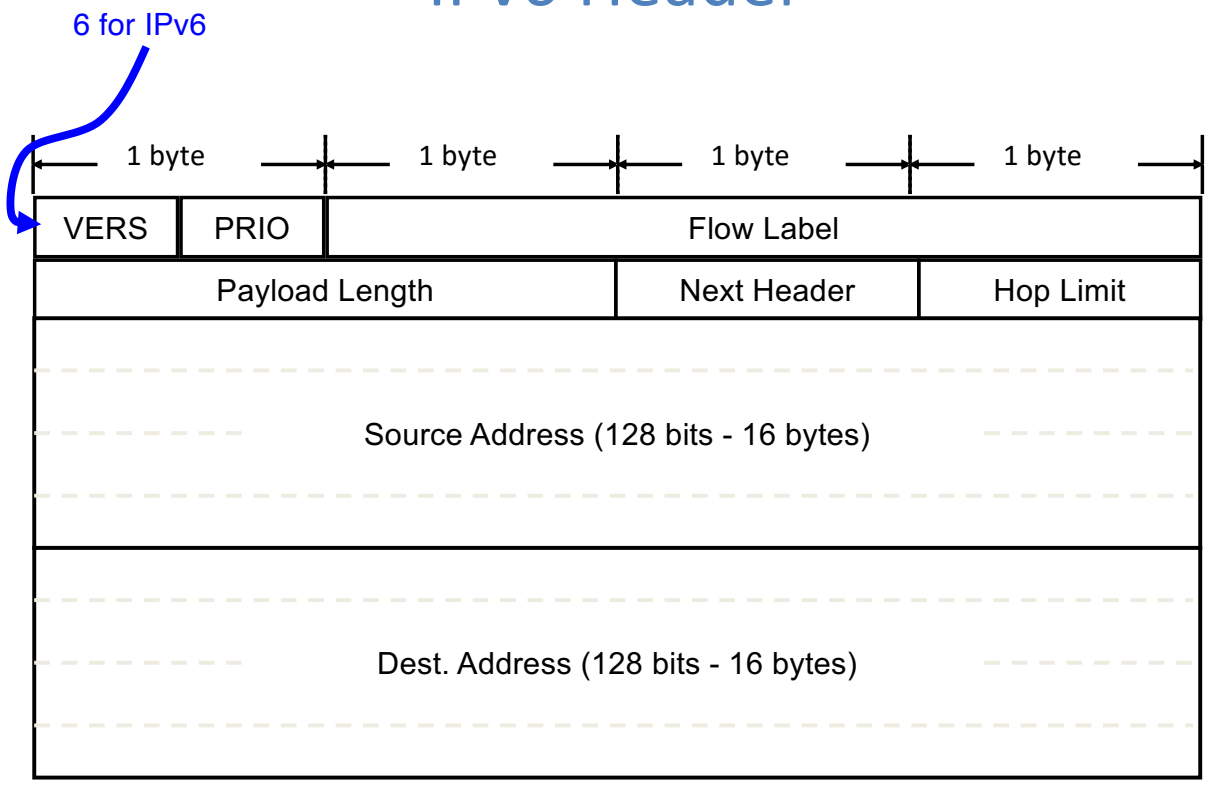
IPv6 Headers

- Simpler header - faster processing by routers
  - No optional fields - fixed size (40 bytes)
  - No fragmentation fields
  - No checksum
- Support for multiple headers
  - More flexible than simple ‘protocol’ field

IPv4 Header



IPv6 Header



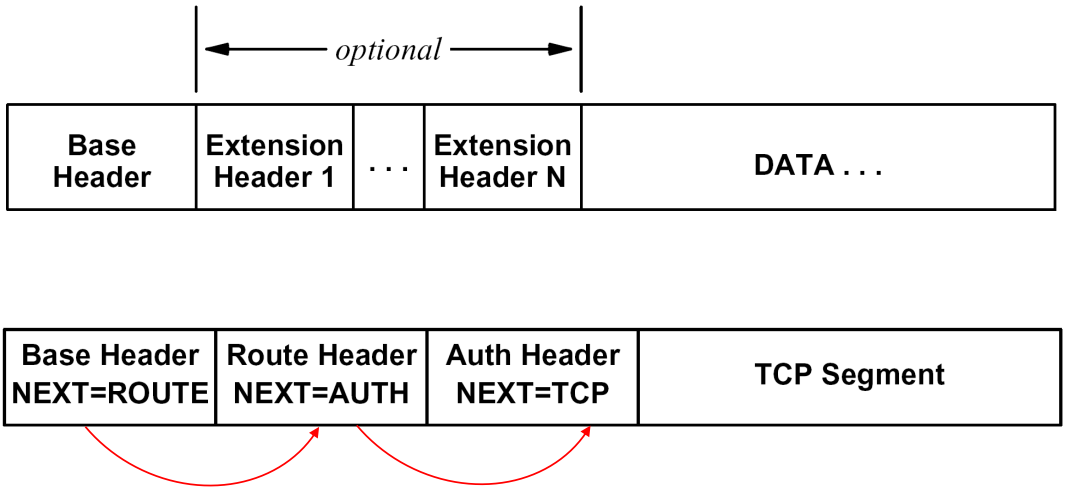
IPv6 Header Fields

- Version: 6 (IP version number)
- Priority: will be used in congestion control
- Flow Label: experimental - sender can label a sequence of packets as being in the same flow
- Payload Length: number of bytes for everything following the 40 byte header, or 0 for a *Jumbogram*

IPv6 Header Fields

- Next Header is similar to the IPv4 “protocol” field - indicates what type of header follows the IPv6 header
- Hop Limit is similar to the IPv4 TTL field (but now it really means hops, not time)

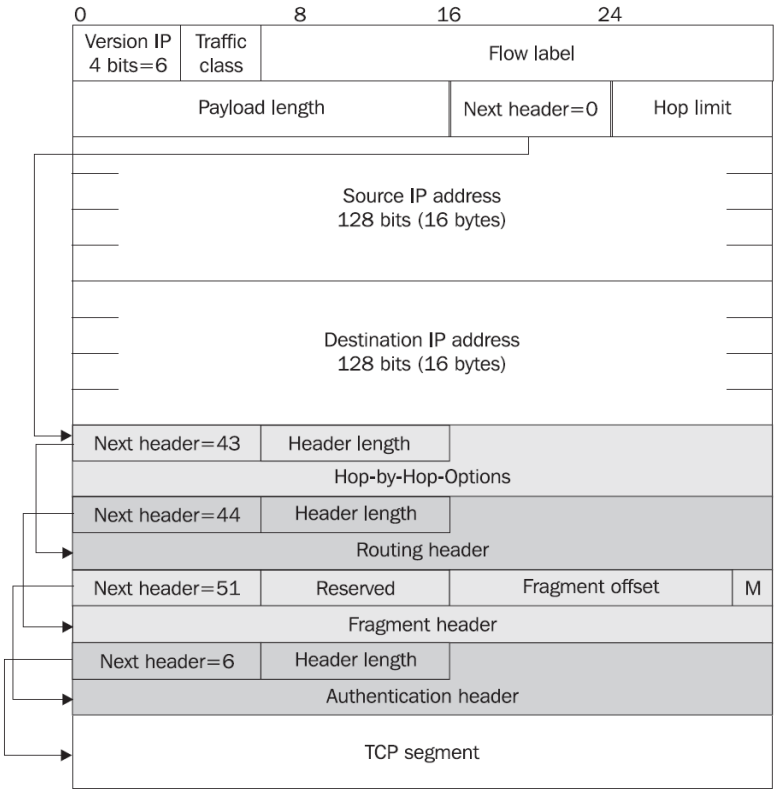
Extension Headers (E.g.)



Extension Headers

- Hop-by-Hop Header
- Routing Header - source routing
- Fragmentation Header - supports fragmentation of IPv6 datagrams
- Authentication Header
- Encapsulating Security Payload Header

Extension Headers (E.g.)

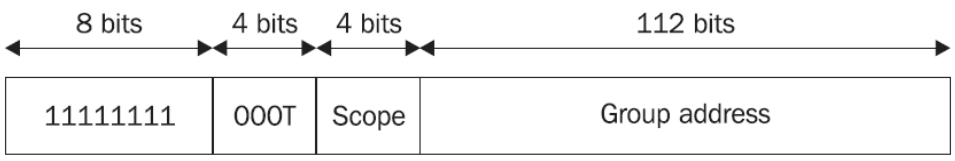


IPv6 Addresses

- Colon hex notation for 128-bit address
  - 2600:df00:ce3e:e250:1020:2801:807:e2e3
  - FF05:0:0:0:0:0:0:B3 = FF05::B3
  - 0:0:0:0:0:0:128.10.2.1
  - 12AB::CD30:0:0:0:0/60
  - ::1, ::0
- High order bits determine the type of address
- Types: unicast, multicast, and anycast
  - Broadcast?
  - FF00::/8 (1111 1111) multicast addresses

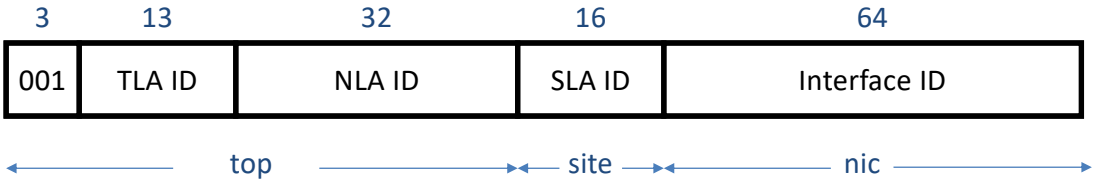
Multicast Addresses

- Well-known multicast (T=0) vs. transient multicast
- Scope:
  - 1: Interface-local scope
  - 2: Link-local scope (LAN)
  - 5: Site-local scope (deprecated)
  - 8: Organization-local scope
  - E: Global scope
- Dedicated: (xx =scope)
  - FFxx::1: Multicasts for all stations (both computers and routers)
  - FFxx::2: Multicasts for all routers
  - FFxx::9: Multicasts for all routers using the RIP protocol



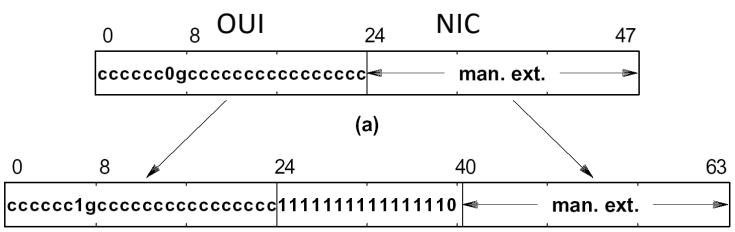
IPv6 Addresses

- Maintained by Internet Registries
  - Distributing IP addresses
- Hierarchy of authority
  - Assigning address classes identified by prefixes
  - 2000::/3 (001) global unicast addresses
    - TLA + NLA IDs used for global routing
    - SLA is similar to subnet ID
  - Interface ID is based on hardware MAC address



IPv6 Addresses (cont'd)

- RFC 4291
- Link-local unicast (1111 1110 10)
  - Neighbor discovery and autoconfiguration
- Interface encoding
  - IEEE's 64-bit Extended Unique Identifier (EUI-64) format
  - 64-bit interface id encoding 48-bit MAC id
  - U/L-bit: ieee (0) is global vs local admin (1)



Autoconfiguration

- Serverless
- Link-local addressing
  - FE80::/10 (1111 1110 10)
- Embedded interface identifier
- Router solicitation (multicast to FF02::2)
  - Default router
  - Get global addresses
  - Managed configuration (i.e., DHCP)
  - Renumbering: valid and preferred lifetimes

IPv6 Neighbor Discovery

- Incorporates the functionality of several IPv4 operations
  - ARP, ICMP Router Discovery, and ICMP Redirect

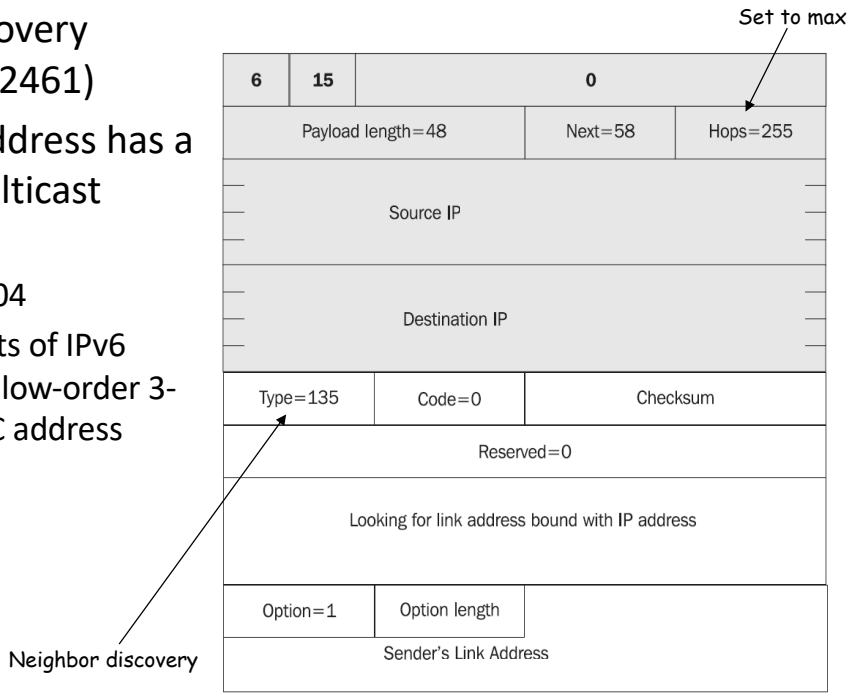
Neighbor Discovery (cont'd)

Two ICMP messages used to resolve the neighboring link address on the local network

- Neighbor Solicitation: Station A solicits the link address of station B via multicast
- Neighbor Advertisement: Station B return its link address to station A
- Functionally similar to ARP use in IPv4

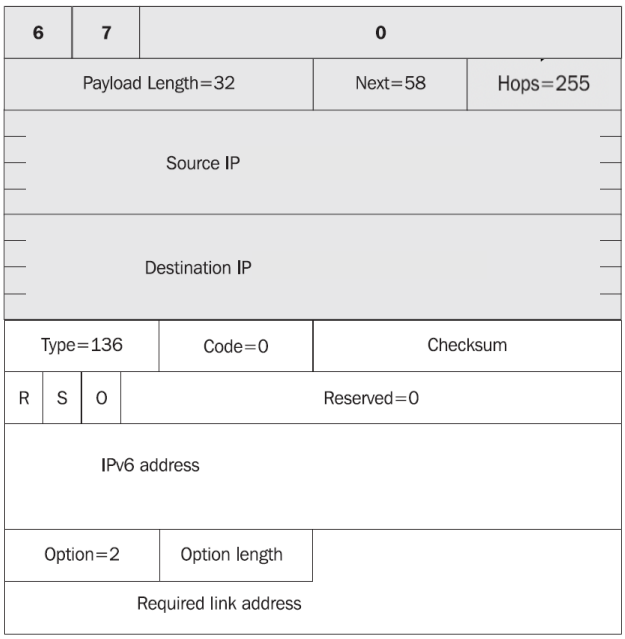
Neighbor Discovery

- Neighbor discovery protocol (RFC 2461)
- Destination address has a composed multicast address from:
  - FF02::1:FF/104
  - Lowest 24-bits of IPv6 address, i.e., low-order 3-bytes of MAC address



Neighbor Advertisement

- R => router is the source
- S => answer to neighbor solicitation
- O => override cache at destination

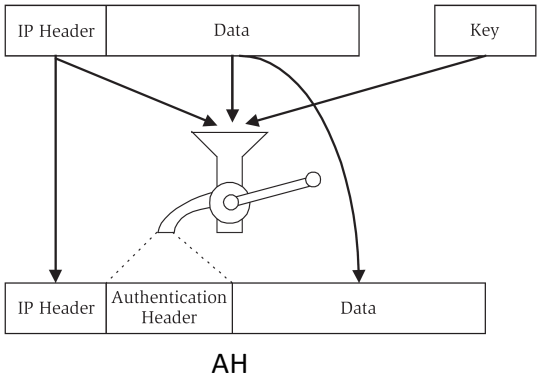


IPv6 versus IPv4

- No more IHL (header length), why?
- No more *protocol* field: *next header* field
- No more fragmentation-related fields
  - All IPv6 hosts and routers must support minimum MTU of 1280 bytes
  - Fragmentation is less likely to occur
  - Router sends error messages back to source when packet is too big
- No more checksum: count on reliable networks and LL/transport checksums

IPv6 vs. IPv4

- RFC 2460 states
  - "Full implementation of IPv6 includes implementation of the authorization header (AH) and encapsulating security payload (ESP)"



Transitioning to IPv6

- Dual-stack
- IPv6 over IPv4 tunnels
  - Challenge: manual configuration of endpoints
  - Many automatic methods exists, e.g., 6to4 tunnels
- Protocol translation mechanisms
  - Can result in feature loss
  - Helps at end-systems with application to network interoperability
  - Basic NAT protocol translation at the network layer

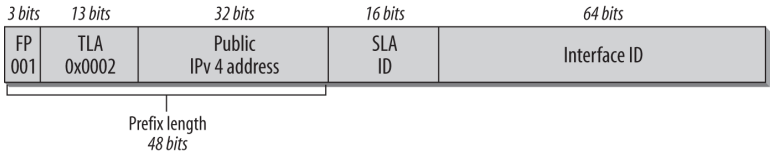
IPv6 Addresses for Transitioning

Embedded IPv4 addresses for dual stacked nodes

- 0000 0000 prefix reserved for IPv4 transition
- "IPv4-compatible IPv6 address" (deprecated):
  - 0000+IPv4 if has a conventional v6 address for tunneling over IPv4 networks
- "IPv4-mapped IPv6 address":
  - FFFF+IPv4 if IPv6 node to IPv4-only peer

6to4 Tunnel

- IPv6 sites communicate without explicit tunnel setup (RFC 3056)
- A transparent mechanism used to provide reachablity between IPv6 nodes
- E.g., the 6to4 tunnel destination is given by the IPv4 address of the router in the IPv6 address that starts with the prefix 2002::/16 with the format is 2002:router-IPv4-address ::/48
- 6to4 addresses (2002:V4ADDR::/48)
- The addresses are used only in configuring static routes at the 6to4 routers for remote IPv6 destinations to go through the tunnel



Application Translator Stacks

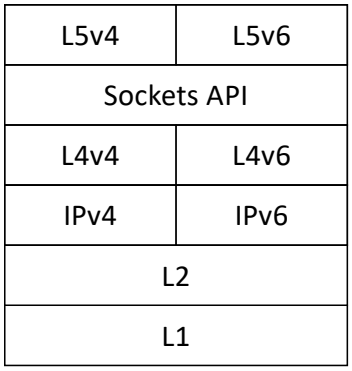
- Allow v4 *applications* to communicate over v6 networks on dual stack hosts without modification
- Implement translator between IPv4 and IPv6
  - Translate packets at IP layer ("Bump-In-Stack")
  - Translate function calls in socket layer ("Bump-In-API")
- Bump-In-API is between the socket API module and the TCP/IP module, comprised of three components:
  - Function mapper, address mapper, name resolver

Application Translator Stacks

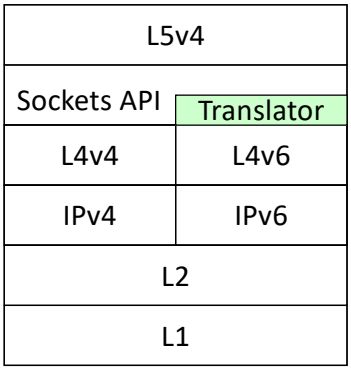
- *Function mapper* translates an IPv4 socket API function into an IPv6 socket API function and vice versa
- *Address mapper* maintains an IPv4 address pool to use with the peer
- At the socket layer API, *name resolver* intercepts the IPv4 calls and instead calls the IPv6 equivalent, i.e., getaddrinfo()

Application Translator Stacks

- SIIT (Stateless IP/ICMP Translation Algorithm): bidirectional translation algorithm defined in RFC 6145
  - Skip most extension headers/IP options (but fragmentation)
- Protocol translator translates IPv4 into IPv6, and vice versa, using the IP conversion mechanism defined by SIIT algorithm

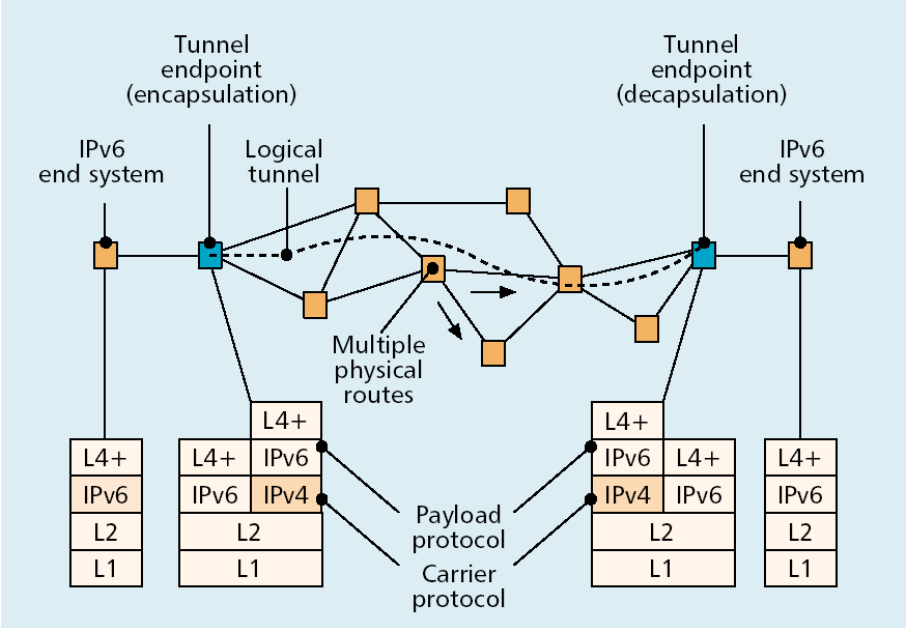


Dual Stack

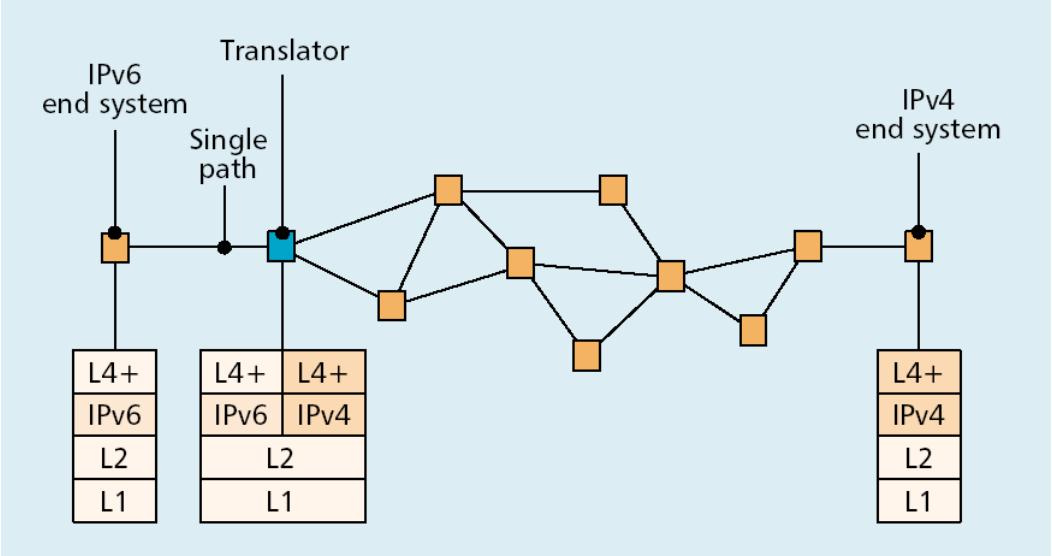


Bump-In-API

Tunnel (E.g.)



Translation (E.g.)

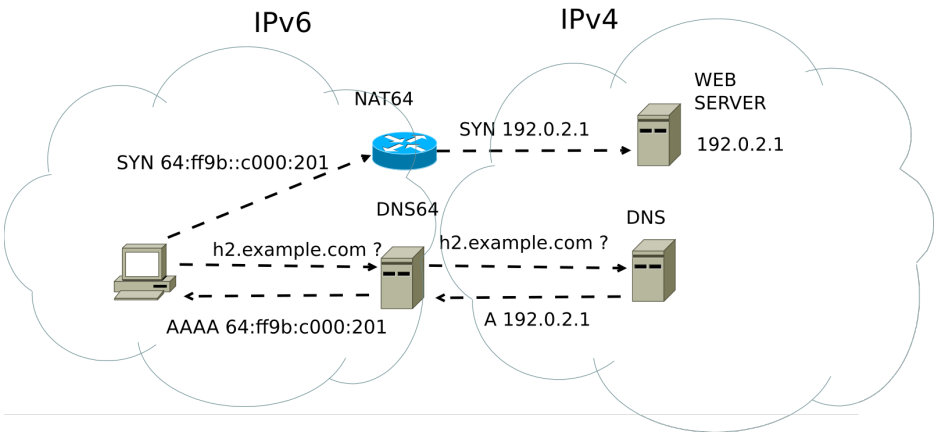


NAT64

- Popular mechanism to enable communication between IPv6 and IPv4 hosts
  - Supports IPv6-only and IPv4-only end points
- Use network address translation (NAT)
- IPv6 client embeds the destination IPv4 address in the IPv6 address
  - Well-known prefix is reserved for the embedding 64:ff9b::/96
- NAT router creates a mapping between the IPv6 and IPv4 addresses
- NAT router also does the IP header translation

NAT64

- NAT router has IPv6 and IPv4 interfaces
- Routing gets traffic to the NAT router
- IP packet header translated



***

IPv6 Sockets Programming

- New address family: `AF_INET6`
- New address data type: `in6_addr`
- New address structure: `sockaddr_in6`

in6_addr

- Network order?

```
struct in6_addr {
    uint8_t s6_addr[16];
};
```

sockaddr_in6

```
struct sockaddr_in6 {
  uint8_t          sin6_len;
  sa_family_t      sin6_family;
  in_port_t        sin6_port;
  uint32_t         sin6_flowinfo;
  struct in6_addr  sin6_addr;
};
```

Dual Stack Server

- In the future it will be important to create servers that handle both IPv4 and IPv6
- Can be handled by the O.S., which has the protocol stacks for both v4 and v6
  - Automatic creation of IPv6 address from an IPv4 client (IPv4-mapped IPv6 address)

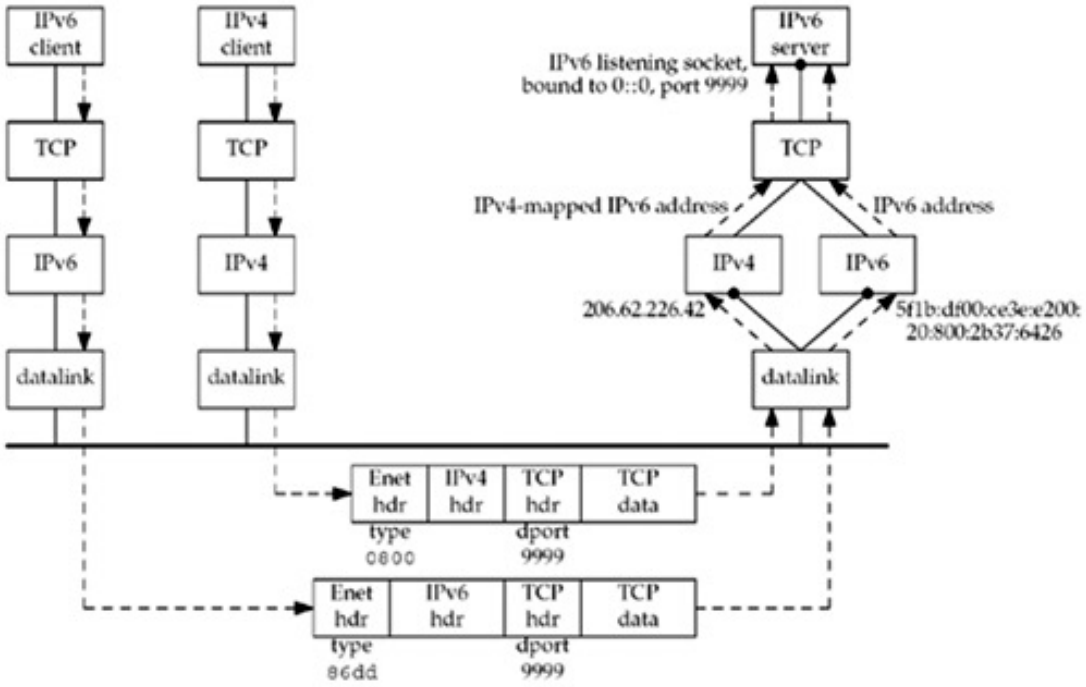
IPv6 Clients

- If the server is IPv4-only
- DNS support for IPv6 addresses can make everything work
  - `getaddrinfo()` returns an IPv4 mapped IPv6 address for hosts that only support IPv4 when AI_V4MAPPED is given
- When the client uses an IPv4-mapped IPv6 address for the server, the kernel detects it and uses IPv4 to the server

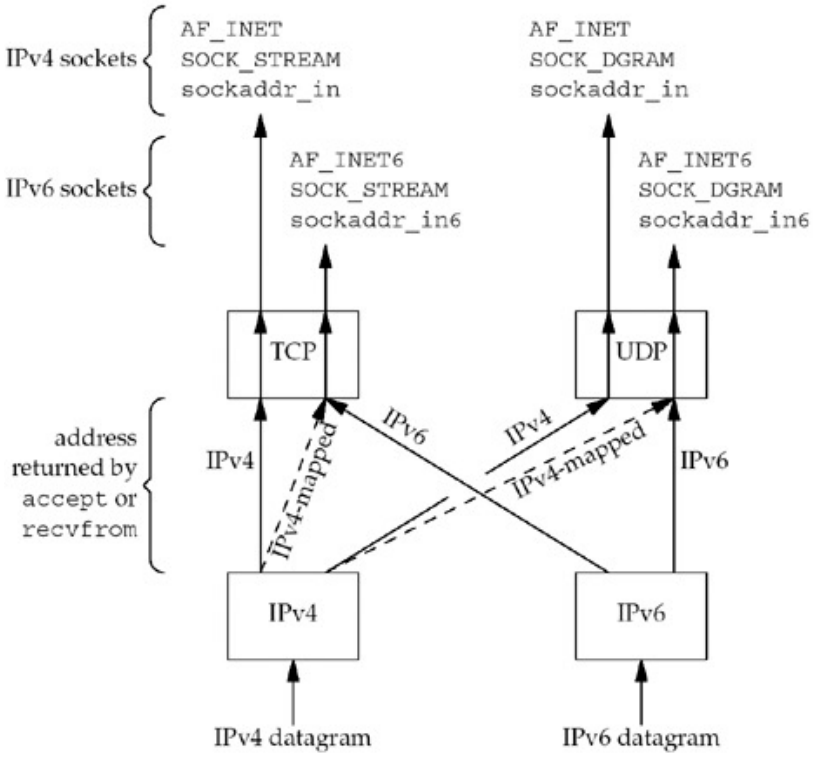
IPv6 - IPv4 Programming

- The kernel does the work for IPv6 server, we can assume we are talking IPv6 to everyone!
- There are macros that determine the type of an IPv6 address
- Can find out if talking to an IPv4 client or server by checking whether the address is an IPv4 mapped address or not

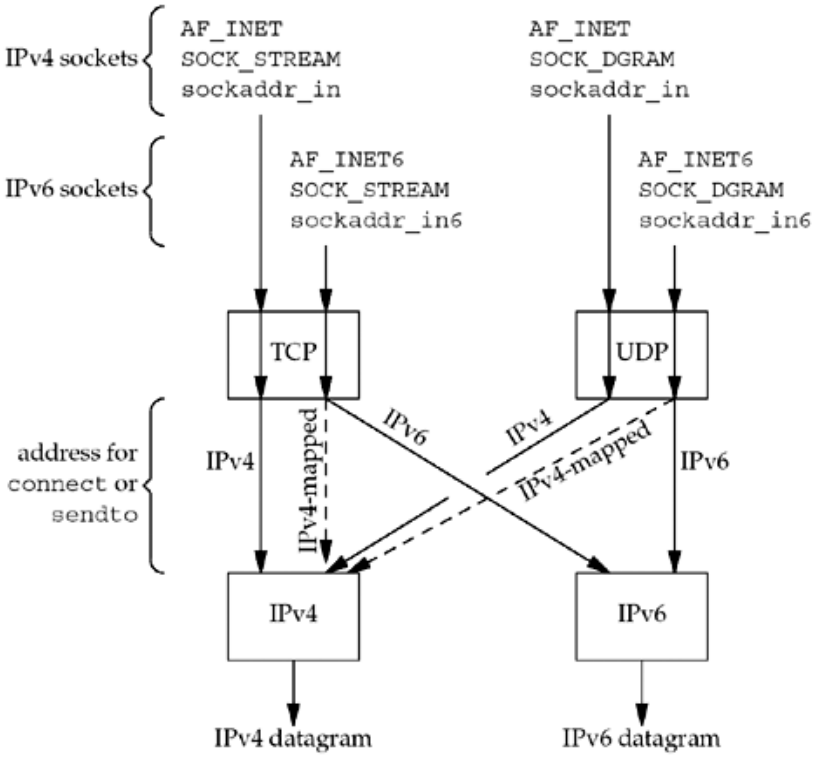
```
int IN6_IS_ADDR_V4MAPPED(const struct in6_addr *aptr);
int IN6_IS_ADDR_LINKLOCAL(const struct in6_addr *aptr);
…
```


IPv6 client
IPv4 client
TCP
TCP
IPv6
IPv4
datalink
datalink
IPv6 server
IPv6 listening socket, bound to 0::0, port 9999
TCP
IPv4-mapped IPv6 address
IPv6 address
IPv4
IPv6
206.62.226.42
5f1b:df00:ce3e:e200:20:800:2b37:6426
datalink
Enet hdr
IPv4 hdr
TCP hdr
TCP data
type 0800
dport 9999
Enet hdr
IPv6 hdr
TCP hdr
TCP data
type 86dd
dport 9999

IPv6 Server


IPv4 sockets
AF_INET
SOCK_STREAM
sockaddr_in
AF_INET
SOCK_DGRAM
sockaddr_in
IPv6 sockets
AF_INET6
SOCK_STREAM
sockaddr_in6
AF_INET6
SOCK_DGRAM
sockaddr_in6
TCP
UDP
address returned by accept or recvfrom
IPv4
IPv4-mapped
IPv6
IPv4
IPv4-mapped
IPv6
IPv4
IPv6
IPv4 datagram
IPv6 datagram

IPv6 Client


IPv4 sockets
AF_INET
SOCK_STREAM
sockaddr_in
AF_INET
SOCK_DGRAM
sockaddr_in
IPv6 sockets
AF_INET6
SOCK_STREAM
sockaddr_in6
AF_INET6
SOCK_DGRAM
sockaddr_in6
TCP
UDP
address for
connect or
sendto
IPv4
IPv4-mapped
IPv6
IPv4
IPv4-mapped
IPv6
IPv4
IPv6
IPv4 datagram
IPv6 datagram

***

IPv6 Statistics

Google IPv6

Statistics

Google collects statistics about IPv6 adoption in the Internet on an ongoing basis. We hope that publishing this information will help Internet providers, website owners, and policy makers as the industry rolls out IPv6.

IPv6 Adoption | Per-Country IPv6 adoption

IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.