What should Spec Writers Know about IPv6 ?

Jordi Palet
(jordi.palet@consulintel.es)
EDU-Team

Why This Class ?

- At the time being, there is no BCP or IETF-wide policy that requires IPv6 support in every new protocol defined for IPv4
- However, omitting IPv6 with no explanation is not generally accepted by the IESG

Actual Situation

- Many documents come to the IESG with IPv6-related issues
  - Mistakes in the usage of IPv6
  - Failure to note differences from IPv4
  - Ignoring IPv6 entirely, without explanation

Is It Difficult ?

- Adding support for IPv6 usually ranges from trivial to simple
  - There may be some more complicated cases
- In general, there is no change in the logic of any upper layer protocol, except for those implied by the socket API differences
  - Typically no need for conceptual redesigns at layer 4 and above
- So … there is no excuse for not doing it !

Contents

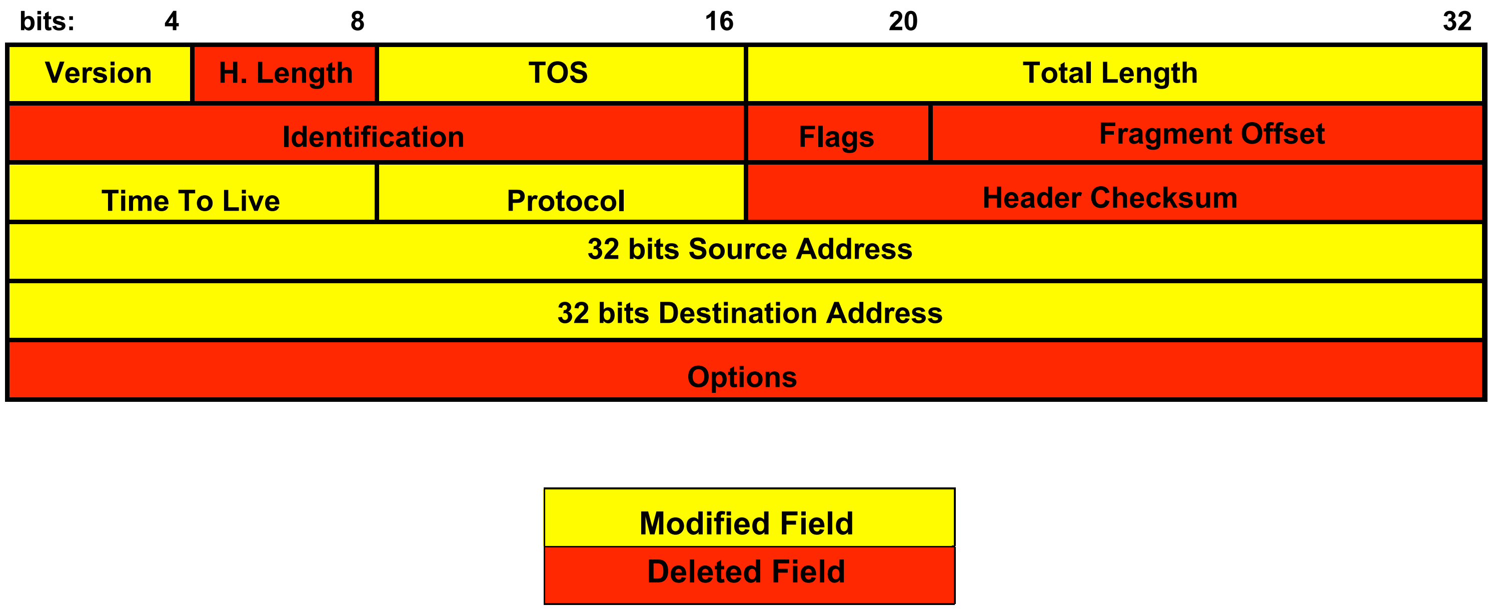
IPv6 Specification (RFC2460)

Changes from IPv4 to IPv6

- Expanded Addressing Capabilities
- Header Format Simplification
- Improved Support for Extensions and Options
- Flow Labeling Capability
- Authentication and Privacy Capabilities

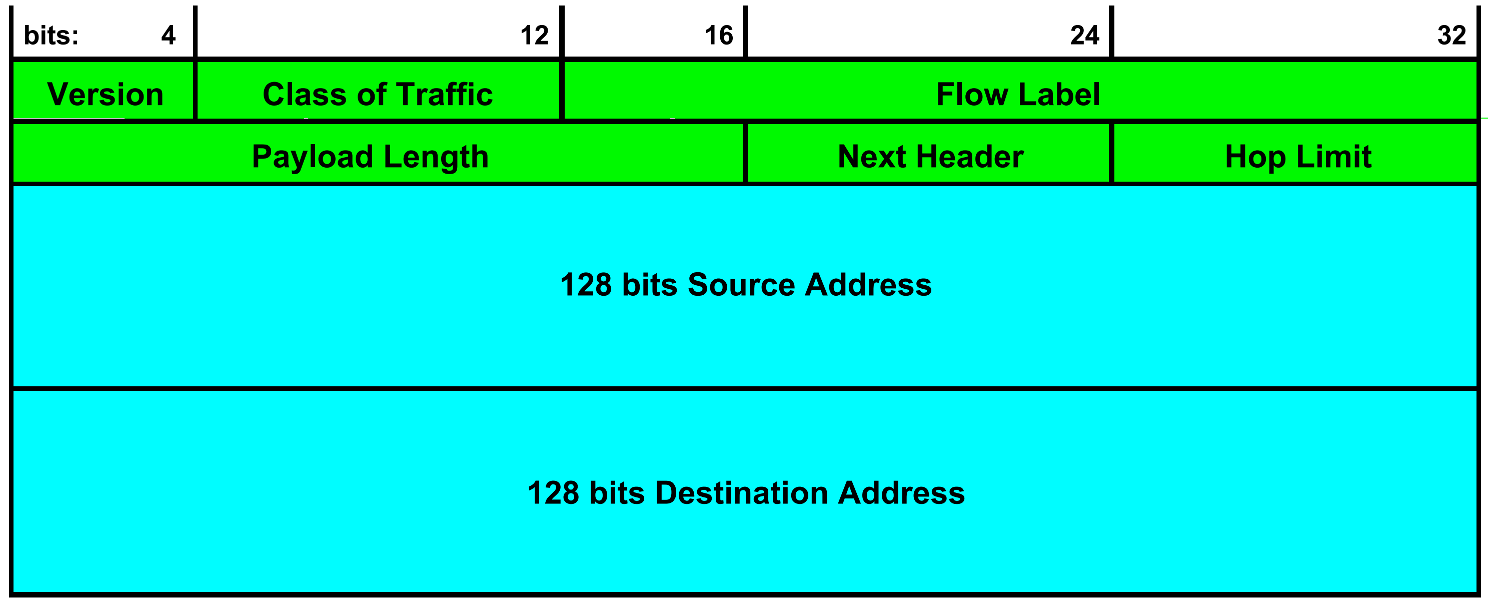
IPv4 Header Format

- 20 Bytes + Options



IPv6 Header Format

- From 12 to 8 Fields (40 bytes)



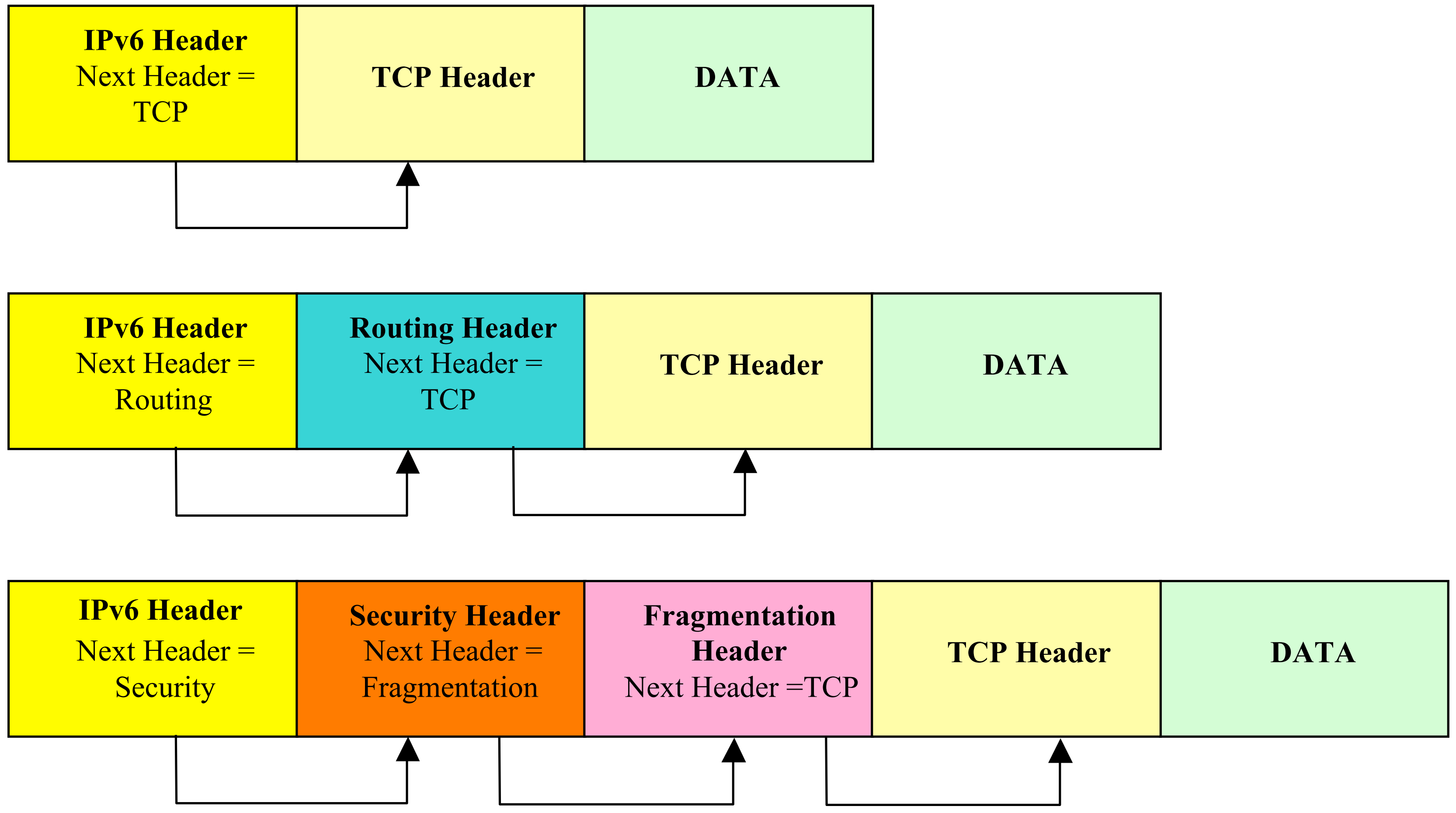
  – Avoid checksum redundancy
  – Fragmentation end to end

Summary of Header Changes

- 40 bytes
- Address increased from 32 to 128 bits
- Fragmentation and options fields removed from base header
- Header checksum removed
- Header length is only payload (because fixed length header)
  - Include length count of present extension headers
- New Flow Label field
- TOS -> Traffic Class
- Protocol -> Next Header (extension headers)
- Time To Live -> Hop Limit
- Alignment changed to 64 bits

Extension Headers

- "Next Header" Field



Extension Headers Goodies

- Processed Only by Destination Node
  - Exception: Hop-by-Hop Options Header
- No more “40 byte limit” on options (IPv4)
- Extension Headers defined currently (to be used in the following order):
  - Hop-by-Hop Options (0)
  - Destination Options (60) / Routing (43)
  - Fragment (44)
  - Authentication (RFC4302, next header = 51)
  - Encapsulating Security Payload (RFC4303, next header = 50)
  - Destination Options (60)
  - Mobility Header (135)
  - No next header (59)
    - TCP (6), UDP (17), ICMPv6 (58)

Unrecognized Headers

- In case of an unrecognized Next Header value in the current header, the node, should discard the packet and send an ICMP Parameter Problem message to the source of the packet

Addressing Architecture

Addressing Architecture

- Not just bigger addresses, a new address architecture …
- Note that IP addresses aren’t 32-bit node identifiers
- See RFC4291 “IP Version 6 Addressing Architecture” for complete info

IPv4 vs. IPv6

IPv4	IPv6
•Address Length: –32 bits	–128 bits
•Literal representation using decimal notation –10.10.10.1	•Hexadecimal notation –2001:0DB8:0003:0004:0005:0006:0007:0008 –2001:DB8:3:4:5:6:7:8 –2001:DB8::1 –FF02::1
•Broadcast	•No IPv6 broadcast address: –Use All-Nodes Multicast address instead
•Loopback: –127.0.0.1	•Loopback: –::1
	•Scoped Unicast Addressing –Link-local vs. Global (site-local is deprecated)
	•All interfaces are required to have at least one Link-Local unicast address and may have multiple IPv6 addresses (any type/scope)

Literal Representation IPv4/IPv6

- Alternative form when dealing with mixed environments
  - IPv6 addresses with embedded IPv4 addresses
    - Use IPv4-Mapped IPv6 Address, IPv4-compatible address is deprecated
    - ::FFFF:10.0.0.1

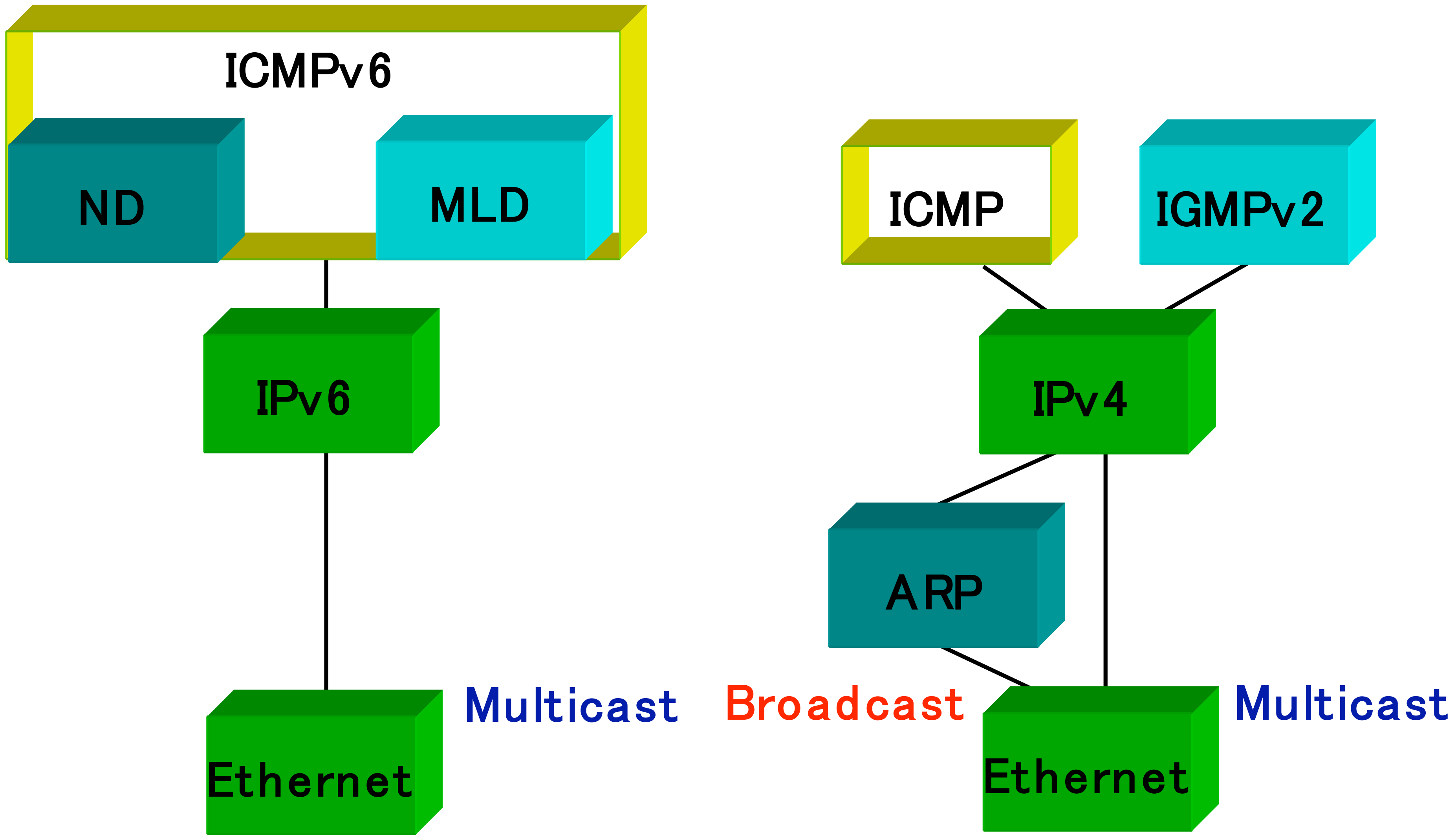
Addressing Model

- IPv6 addresses of all types are assigned to interfaces, not nodes. An IPv6 unicast address refers to a single interface. Since each interface belongs to a single node, any of that node's interfaces' unicast addresses may be used as an identifier for the node.
- All interfaces are required to have at least one Link-Local unicast address.
- A single interface may also have multiple IPv6 addresses of any type (unicast, anycast, and multicast) or scope.
- A unicast address or a set of unicast addresses may be assigned to multiple physical interfaces if the implementation treats the multiple physical interfaces as one interface when presenting it to the internet layer.

IPv4/IPv6 Similarities

- A subnet prefix is associated with one link. Multiple subnet prefixes may be assigned to the same link.
- Literal representation of address prefixes follow CIDR notation
  - ipv6-address/prefix-length (decimal)
    - 2001:DB8::1/60

Control Plane IPv4 vs. IPv6



Address Types

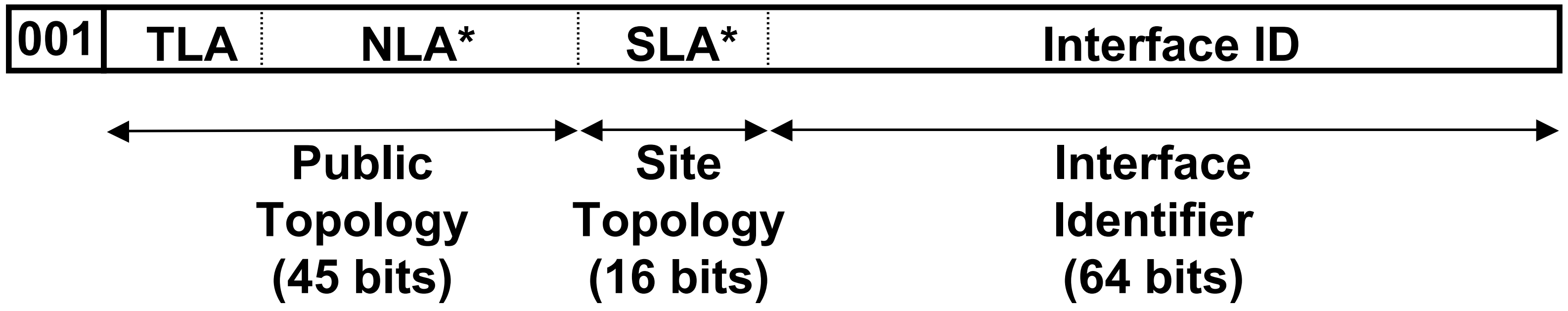
- Unicast (one-to-one)
  - global
  - link-local
  - site-local (deprecated)
  - Unique Local (ULA)
  - IPv4-compatible (deprecated)
- Multicast (one-to-many)
- Anycast (one-to-nearest)
- Reserved

Address Type Identification

Address Type	Binary Prefix	IPv6 Notation
Unspecified	00…0 (128 bits)	::/128
Loopback	00…1 (128 bits)	::1/128
Multicast	1111 1111	FF00::/8
Link-Local Unicast	1111 1110 10	FE80::/10
ULA	1111 110	FC00::/7
Global Unicast	(everything else)	
Site-Local Unicast (deprecated)	1111 1110 11	FEF0::/10
IPv4-compatible (deprecated)	00…0 (96 bits)	::/96

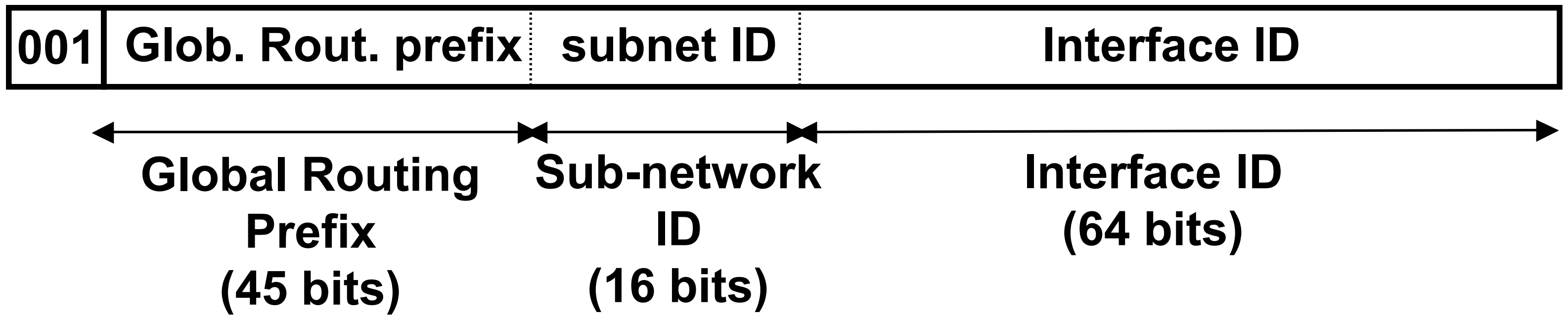
Note: Anycast addresses allocated from unicast prefixes

Aggregatable Global Unicast Addresses (RFC2374) (Deprecated)



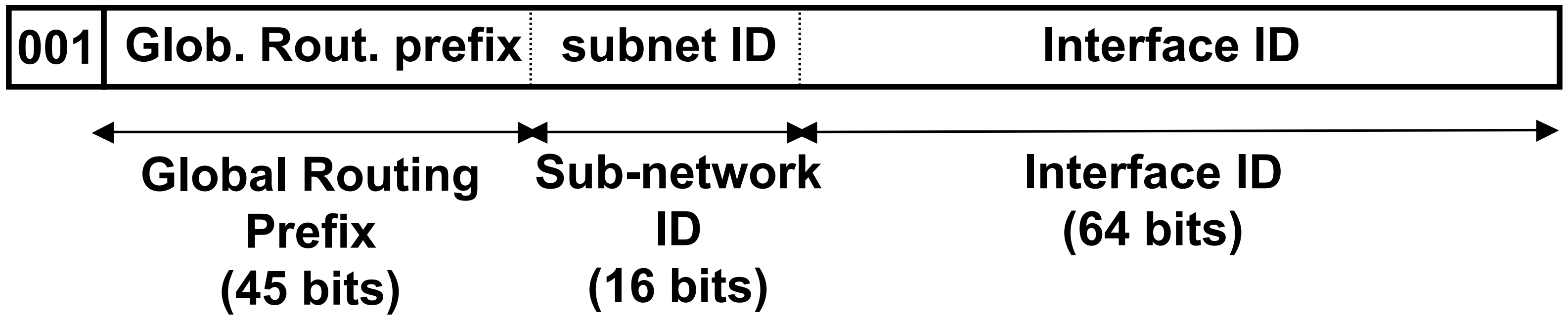
- TLA = Top-Level Aggregator
  NLA* = Next-Level Aggregator(s)
  SLA* = Site-Level Aggregator(s)
- TLAs may be assigned to ISPs and IX

Global Unicast Addresses (RFC3587)



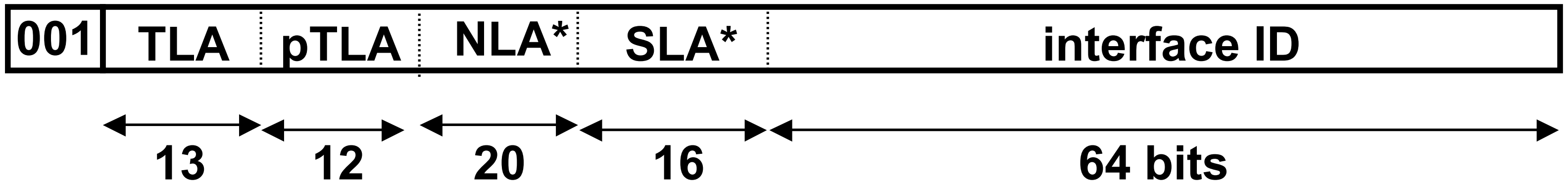
- The global routing prefix is a value assigned to a zone (site, a set of subnetworks/links)
  - It has been designed as an hierarchical structure from the Global Routing perspective
- The subnetwork ID, identifies a subnetwork within a site
  - Has been designed to be an hierarchical structure from the site administrator perspective
- The Interface ID is build following the EUI-64 format

Global Unicast Addresses in Production Networks



- LIRs receive by default /32
  - Production addresses today are from prefixes 2001, 2003, 2400, 2800, etc.
  - Can request for more if justified
- /48 used only within the LIR network, with some exceptions for critical infrastructures
- /48 to /128 is delegated to end users
  - Recommendations following RFC3177 and (some) current policies
    - /48 general case, /47 if justified for bigger networks
    - /64 if only and only one network is required
    - /128 if it is sure that only and only one device is going to be connected

Global Unicast Addresses for the 6Bone Until 06/06/06 !



- 6Bone: experimental IPv6 network used for testing only
- TLA 1FFE (hex) assigned to the 6Bone
  - thus, 6Bone addresses start with 3FFE:
  - (binary 001 + 1 1111 1111 1110)
- Next 12 bits hold a “pseudo-TLA” (pTLA)
  - thus, each 6Bone pseudo-ISP gets a /28 prefix
- Not to be used for production IPv6 service

Link-Local & Site-Local Unicast Addresses

Link-local addresses for use during auto-configuration and when no routers are present:

1111111010	**0**	**interface ID**

Site-local addresses for independence from changes of TLA / NLA* (deprecated !):

1111111011	**0**	**SLA***	**interface ID**

Interface IDs

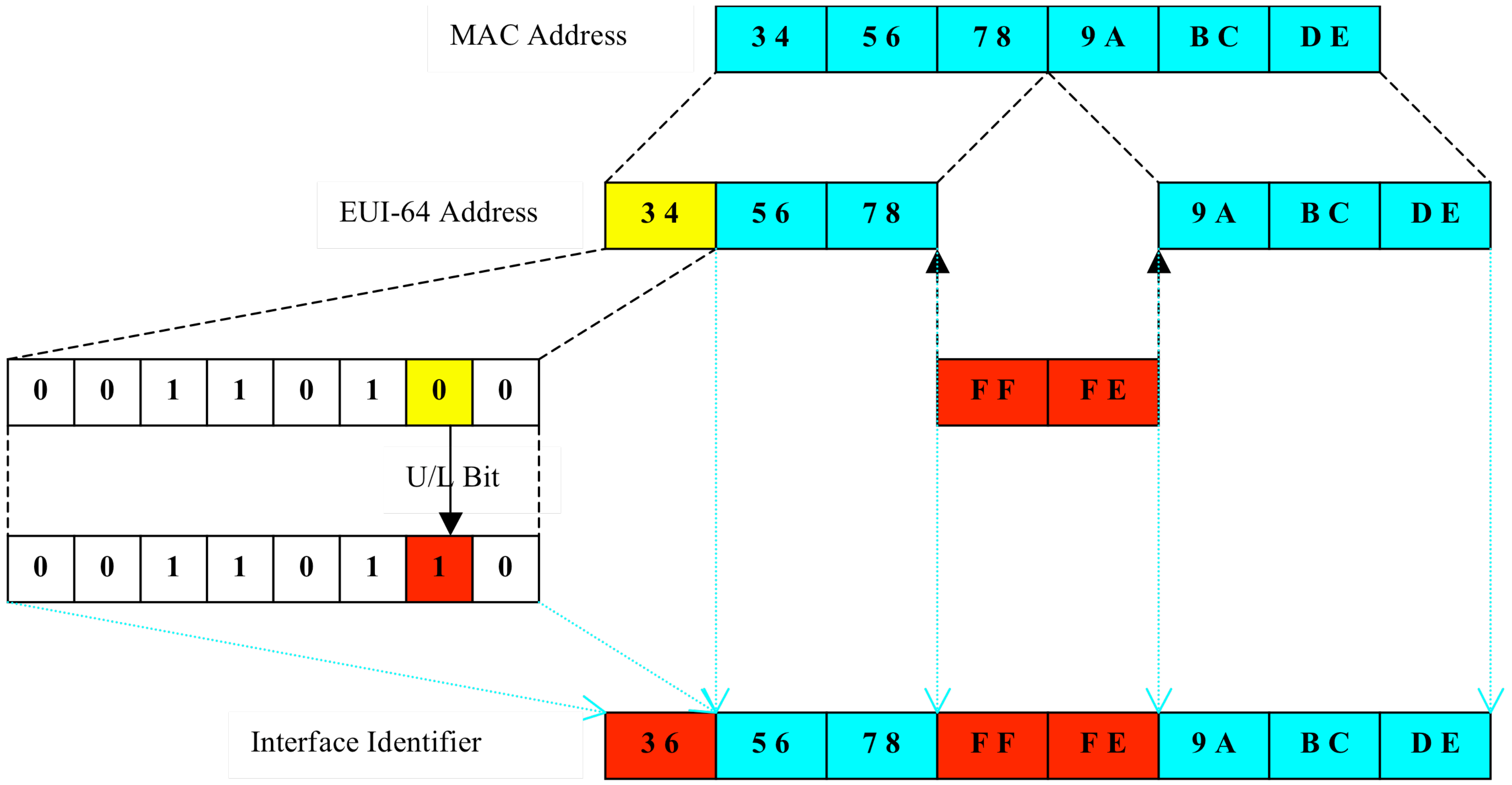
The lowest-order 64-bit field of unicast addresses may be assigned in several different ways:

- auto-configured from a 48-bit MAC address (e.g., Ethernet address), expanded into a 64-bit EUI-64
- assigned via DHCP
- manually configured
- auto-generated pseudo-random number (to counter some privacy concerns)
- possibly other methods in the future

IPv6 in Ethernet

48 bits	48 bits	16 bits	
Ethernet Destination Address	Ethernet Source Address	1000011011011101 (86DD)	IPv6 Header and Data

EUI-64


MAC Address
3 4
5 6
7 8
9 A
B C
D E
EUI-64 Address
3 4
5 6
7 8
9 A
B C
D E
0 0 1 1 0 1 0 0
F F
F E
U/L Bit
0 0 1 1 0 1 1 0
Interface Identifier
3 6
5 6
7 8
F F
F E
9 A
B C
D E

Some Special-Purpose Unicast Addresses

- The unspecified address, used as a placeholder when no address is available:

  0:0:0:0:0:0:0:0

- The loopback address, for sending packets to self:

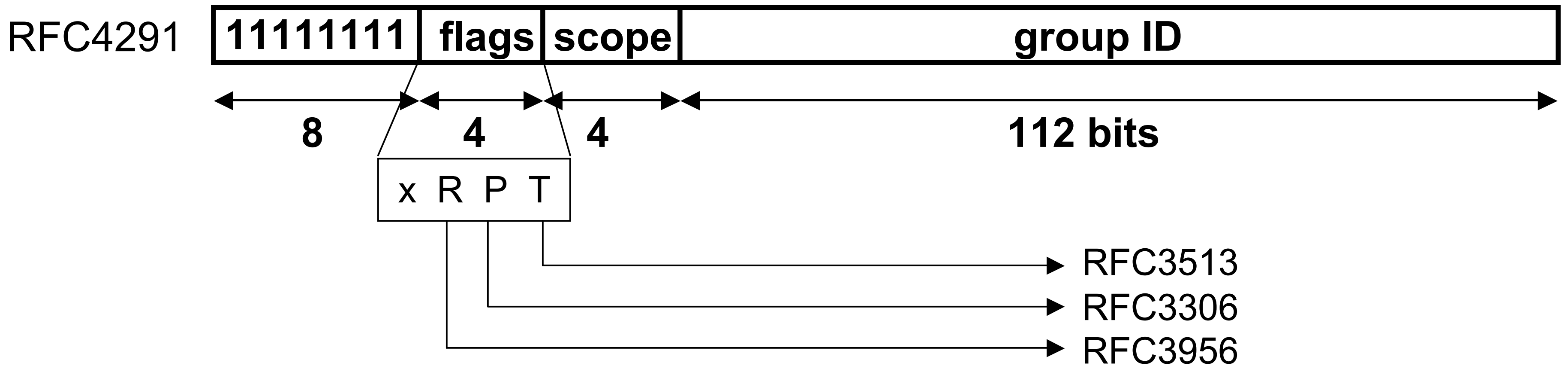
  0:0:0:0:0:0:0:1

Required Anycast Address

- The Subnet-Router anycast address is predefined.
- Format:

  Subnet prefix (n bits) + 0…0 (128-n)
- The "subnet prefix" in an anycast address is the prefix that identifies a specific link. This anycast address is syntactically the same as a unicast address for an interface on the link with the interface identifier set to zero.

Multicast Addresses



- Low-order flag indicates permanent/transient group
- Scope field:

1	node local
2	link-local
5	site-local
8	organization-local
B	community-local
E	global

(all other values reserved)

Reserved Multicast Addresses (I)

- Node-Local Scope

– FF01::1	All Nodes Address
– FF01::2	All Routers Address

- Link-Local Scope

– FF02::1	All Nodes Address
– FF02::2	All Routers Address
– FF02::4	DVMRP Routers
– FF02::5	OSPFIGP
– FF02::6	OSPFIGP Designated Routers
– FF02::9	RIP Routers
– FF02::B	Mobile-Agents
– FF02::D	All PIM Routers
– FF02::1:2	All-DHCP-agents
– FF02::1:FFXX:XXXX	Solicited-Node Address

Reserved Multicast Addresses (II)

- Site-Local Scope

– FF05::2	All Routers Address
– FF05::1:3	All-DHCP-servers
– FF05::1:4	All-DHCP-relays

- Variable Scope Multicast Addresses

– FF0X::101	Network Time Protocol (NTP)
– FF0X::129	Gatekeeper
– FF0X::2:0000-FF0X::2:7FFD	Multimedia Conference Calls
– FF0X::2:7FFE	SAPv1 Announcements
– FF0X::2:8000-FF0X::2:FFFFSAP	Dynamic Assignments

Important Multicast Addresses

• FF01::1, FF02::1	All-nodes
• FF01::2, FF02::2, FF05::2	All routers

- Solicited Node (SN) address from a unicast one

– For the address that finish with	“XY:ZTUV”
– the SN is	FF02::1:FFXY:ZTUV

- Every IPv6 node must join SN for all its unicast and anycast addresses, and to “all-nodes”

Multicast Listener Discovery

- MLD (RFC2710) enables each IPv6 router to learn which multicast addresses have listeners on each of its directly attached links
- This is a mandatory function in IPv6 nodes
- Is used instead of IGMP (as in IPv4)
- Current version MLDv2: RFC3810 which interoperates with MLDv1
- It supports source-filtering but it requires PIM-SSM

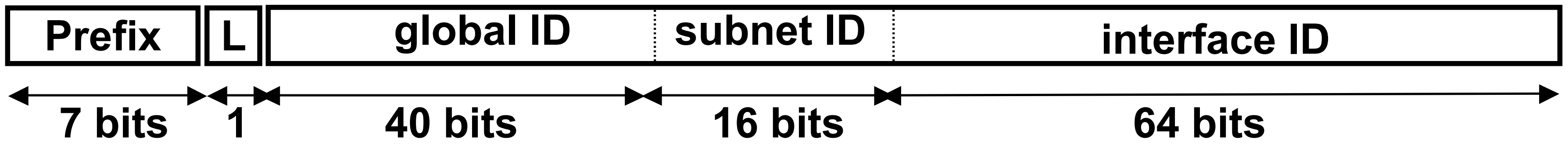
Node Required Addresses

- A host is required to recognize the following addresses as identifying itself:
  - Its required Link-Local address for each interface
  - Any additional Unicast and Anycast addresses that have been configured for the node's interfaces (manually or automatically)
  - The loopback address
  - The All-Nodes multicast address
  - The Solicited-Node multicast address for each of its unicast and anycast addresses
  - Multicast addresses of all other groups to which the node belongs
- A router, in addition, is required to recognize:
  - The Subnet-Router Anycast addresses for all the interfaces for which it is configured to act as a router
  - All other Anycast addresses with which the router has been configured
  - The All-Routers multicast addresses

Unique Local Addresses

- IPv6 adds support for unique local addresses (ULAs) in RFC4193.
  - Globally, well-known, unique prefix
    - Easy filtering at site boundaries
  - Probabilistically unique, so a node can be on more than one local network
  - Not a direct map to IPv4 net 10 addresses
  - Useful for local or private network addressing
    - They are not expected to be routable on the Global Internet
    - They are routable inside of a more limited area such as a site
    - They may also be routed between a limited set of sites
    - If accidentally leaked (routing or DNS), there is no conflict with any other addresses
  - In practice, applications may treat these addresses like global scoped ones
  - Locally-Assigned Local addresses
    - vs Centrally-Assigned Local addresses

IPv6 ULA Format



- FC00::/7 Prefix identifies the Local IPv6 unicast addresses
- L = 1 if the prefix is locally assigned
- L = 0 may be defined in the future (ULA-central)
- ULA are created using a pseudo-randomly allocated global ID
  - This ensures that there is not any relationship between allocations and clarifies that these prefixes are not intended to be routed globally

Privacy Extensions (RFC3041)

- Extension to IPv6 stateless address autoconfiguration, to allow nodes to generate global-scope (temporary) addresses from interface identifiers that change over time providing some privacy degree
- Changing addresses have implications for transport protocols which may rely globally unique interface identifiers
- Makes more complex debugging of problems
- Not associated to a DNS name, so invalidate the usage of DNS PTR queries to grant access to some services
- For implementations, it is difficult to keep track which addresses are being used by upper layers:
  - For TCP connections, info available in control blocks
  - For UDP-based applications, it may be the case that only the applications know it
- Decision about using public or temporary addresses, is typically only made by applications

IP Layer Issues

Packet Size Issues

- IPv6 requires lower layers to support an MTU of 1280 bytes (as opposed to 536 in IPv4)
- It is recommended that those links with a configurable MTU (such as PPP) use 1500 or greater (to accommodate possible encapsulations)
- Path MTU discovery (RFC1981) it is strongly recommended
- Support for IPv6 Jumbograms (RFC2675)

Fragmentation Issues

- IPv6 routers do not fragment packets
  - Path MTU discovery is mandatory
  - All fragmentation happens at the source
- Application fragmentation preferred versus IPv6 Fragment header usage

ND vs. ARP

- IPv6 L2 address resolution is performed using Neighbor Discovery (ND, RFC2461) instead of ARP
  - ND is ICMP-based
  - Uses Solicited-Node Multicast addresses
  - Requires link-layer multicast
    - Some link types (NBMA) need alternatives
  - Combines address resolution with host autoconfiguration and reachability detection

ND Features

- Router Discovery is part of the base protocols set
- Router advertisements:
  - carry link-layer addresses
  - carry prefixes for a link
  - enable Address Autoconfiguration
- Routers can advertise and MTU for hosts to use on the link
- Redirects contain the link-layer address of the new first hop
- Multiple prefixes can be associated with the same link
- Recipient of an IPv6 redirect assumes that the next-hop is on-link
- Self-contained Neighbor Unreachability Detection
- Detects half-link failures
- Facilitates renumbering
- Immune to off-link ND messages
- More media-independent than ARP (allows using IPsec)

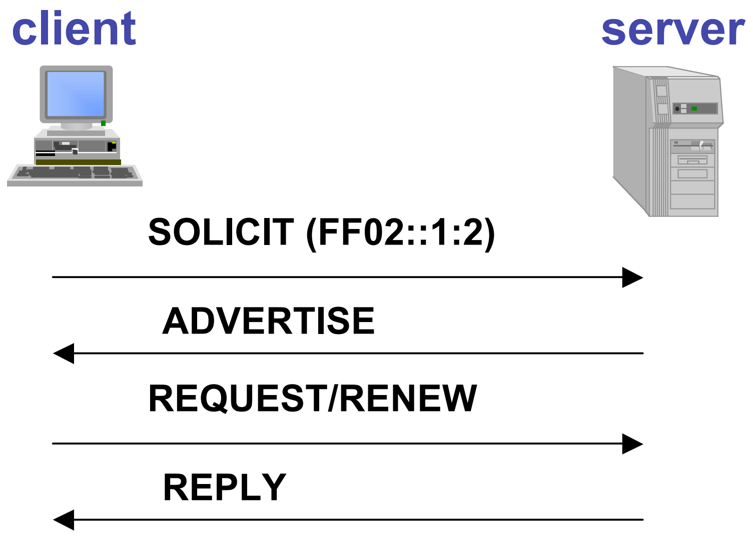
DHCPv6 (RFC3315/RFC4361)

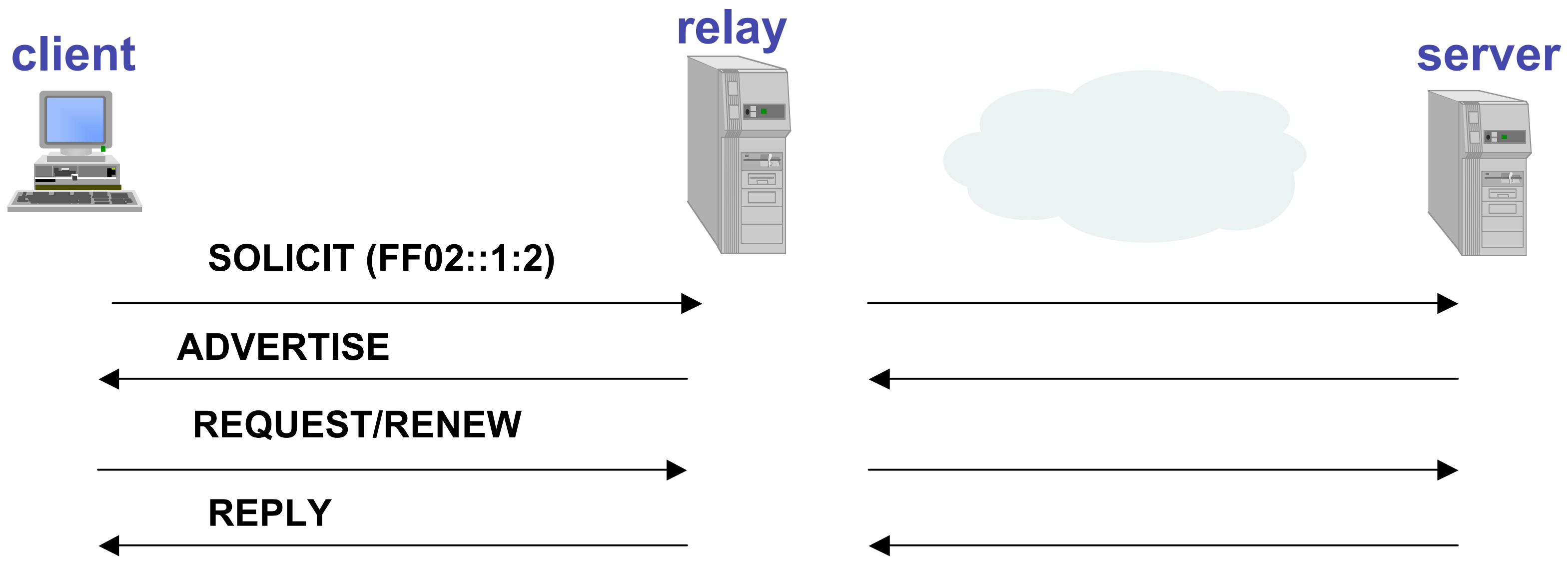
- DHCPv6 is a client-server-based UDP protocol designed to reduce the IPv6 nodes management cost in those environments whereby control of IPv6 address allocation is required and/or more control than the one provided by the stateless mechanism about the provision of network parameters is needed
- DHCP reduces the cost of ownership by centralizing the management of network resources such as IP addresses, routing information, OS installation information, directory service information, and other such information on a few DHCP servers, rather than distributing such information in local configuration files among each network node
- DHCPv6 provides a superset of features, and benefits from the additional features of IPv6 and freedom from BOOTP -backward compatibility constraints

DHCPv6 Details

- UDP ports are
  - Clients listens to 546
  - Server and relays listen to 547
- Address for DHCPv6 relay agent and servers
  - FF02::1:2 (link local scope)
  - FF05::1:3 (site scope only for servers)
- DHCP messages
  - SOLICIT
  - ADVERTISE
  - REQUES
  - CONFIRM
  - RENEW
  - REBIND
  - REPLY
  - RELEASE
  - DECLINE
  - RECONFIGURE
  - INFORMATION-REQUEST
  - RELAY-FORW
  - RELAY-REPL
- Each message can carry one or more DHCP options
  - Domain-list
  - DNS-server
  - IA-NA, etc.
- DHCP Unique Identifier (DUID)
  - servers use DUIDs to identify clients for the selection of configuration parameters and in the association of IAs with clients
  - clients use DUIDs to identify a server in messages where a server needs to be identified

Basic DHCPv6 Example





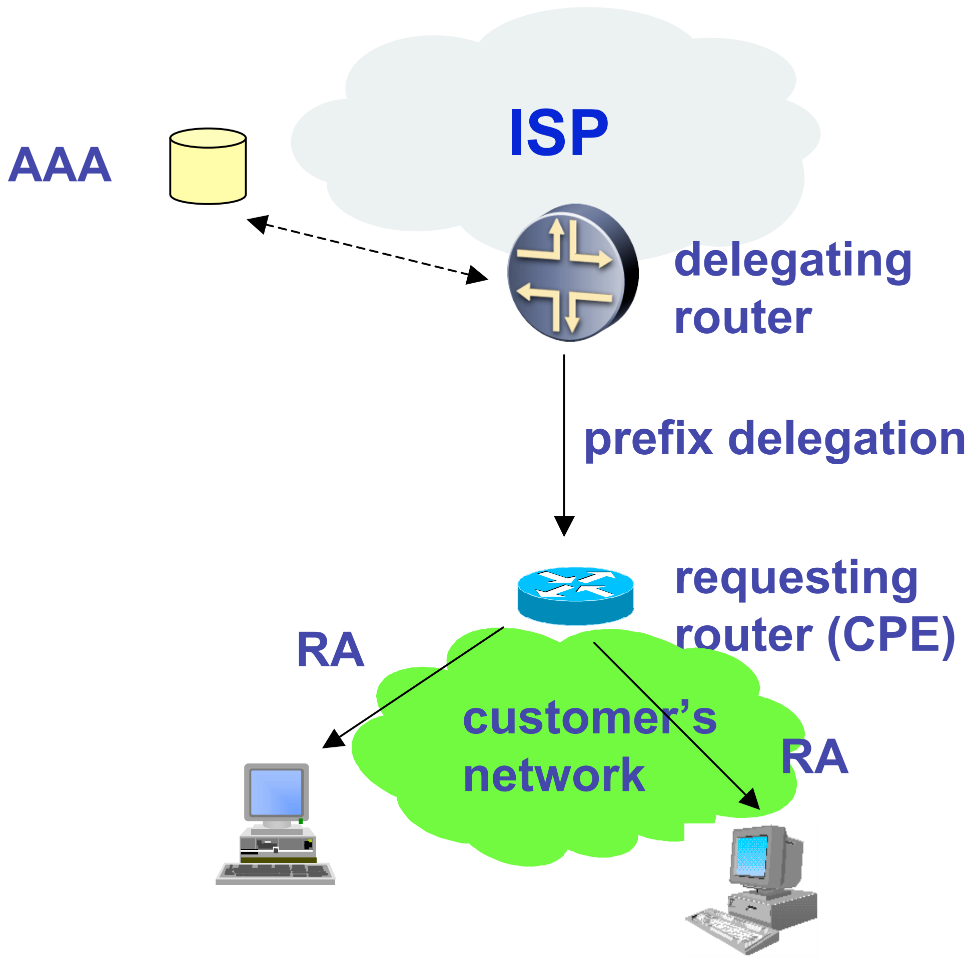
DHCPv6-PD (RFC3633)

- It provides an automated mechanism for the delegation of IPv6 prefixes to authorized requesting routers
- Delegating router does not require knowledge about the topology of the networks to which the requesting router is attached
- Delegating router does not require other information aside from the identity of the requesting router to choose a prefix for delegation
  - for example a ISP to assign a prefix to a CPE device acting as a router

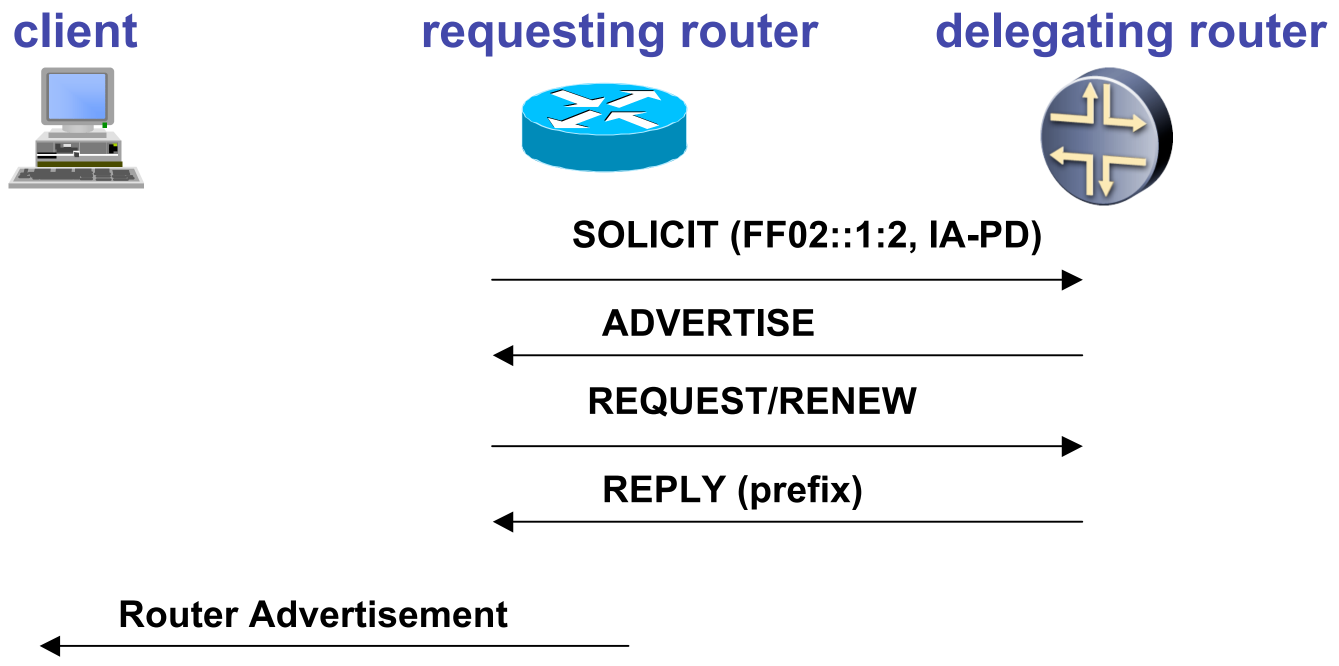
DHCPv6 Details

- Requesting router (RR) authentication is needed
- Profile for a RR could be stored in AAA server
- Delegated prefix could be gotten from either:
  - the customer’s profile stored in the AAA server
  - prefix pool
- The delegated prefixes have lifetime as IPv6 address in DHCPv6
- DHCPv6-PD doesn’t provide a way to propagate the delegated prefix through the customer’s network
  - ::/64 prefixes form the delegated prefix are assigned in the RR according to the configured policy
- DHCPv6 relay agents could also be used as in DHCPv6

Network architecture for DHCPv6-PD



Basic DHCPv6-PD Example



Transport Issues

IPv6 Pseudo-Header

- Any transport or other upper-layer protocol that includes the addresses from the IP header in its checksum computation must be modified for use over IPv6, to include the 128-bit IPv6 addresses instead of 32-bit IPv4 addresses.
- A TCP and UDP pseudo-header is consequently defined for IPv6.
- UDP/IPv6 packets require UDP checksum computed by the originating node.
- ICMPv6 includes the pseudo-header in its checksum computation.

Maximum Packet Lifetime

- IPv6 nodes are not required to enforce maximum packet lifetime.
- Any upper-layer protocol that relies on the IP layer to limit packet lifetime need to be upgraded to provide its own mechanism for detecting and discarding obsolete packets.

Maximum Upper-Layer Payload Size

- When computing the maximum payload size available for upper-layer data, an upper-layer protocol must take into account the larger size of the IPv6 header versus the IPv4 one.

Routing Header Issues

- When an upper-layer protocol responds to a received packet that include a Routing header, only can use one of the following:
  - Response packet without Routing header
  - Response packet with a Routing header NOT derived by reversing the received Routing header
  - Response packet with a Routing header derived by reversing the received Routing header IF AND ONLY IF the integrity and authenticity of the Source Address and Routing header from the received packet have been verified by the responder

Flow Label

- The 20-bit Flow Label field may be used by a source to label sequences of packets for which it request special handling by the IPv6 routers, such as non-default QoS or “real-time” service
- The Flow Label value set by the source must be delivered unchanged to the destination(s)
- For more info see RFC3697
- More work to be done in this area

Security Issues

Security

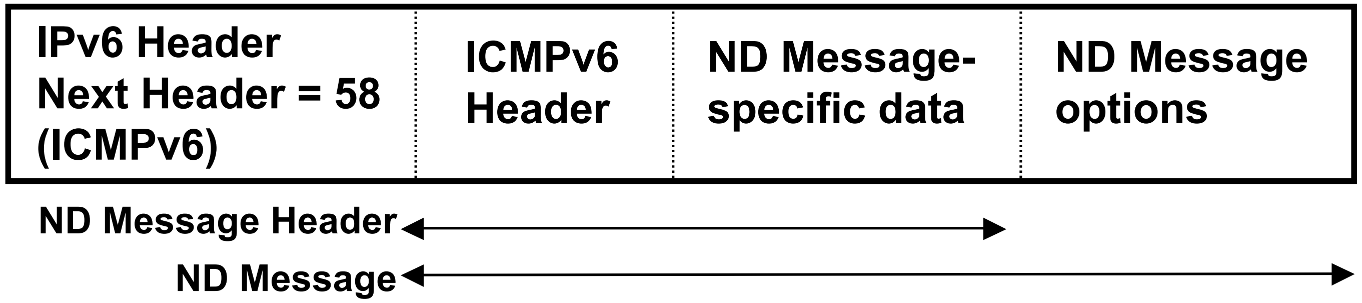
- IPsec (AH and ESP) required in “complete” IPv6 stacks
- Security features of IPv6 as described in RFC2401
- IPv6 addressing architecture do not have additional security implications

Secure Neighbor Discovery (SEND) - RFC3971

- IPv6 nodes use the Neighbor Discovery Protocol (NDP) to:
  - Discover other nodes on the link
  - Determine their link-layer addresses to find routers
  - Maintain reachability information about the paths to active neighbors
- NDP is vulnerable to various attacks if it is not secured
- RFC3971 specifies security mechanisms for NDP
  - Unlike those in the original NDP specifications, these mechanisms do not use IPsec
  - SEND is applicable in environments where physical security on the link is not assured (such as over wireless) and attacks on NDP are a concern

SEND Elements

- The NDP messages follow the ICMPv6 message format
- An actual NDP message includes
  - an NDP message header
    - ICMPv6 header
    - ND message-specific data
  - and zero or more NDP options, which are formatted in the Type-Length-Value format



- To secure the NDP, a set of new Neighbor Discovery options is introduced and used to protect NDP messages
- RFC3971 introduces
  - SEND's Neighbor Discovery options
  - An authorization delegation discovery process
  - An address ownership proof mechanism
  - And requirements for the use of these components in NDP

SEND’s Neighbor Discovery Options

- CGA (Cryptographically Generated Addresses, RFC3972) option, is used to carry the public key and associated parameters
  - CGA are used to make sure that the sender of a Neighbor Discovery message is the "owner" of the claimed address
  - A public-private key pair is generated by all nodes before they can claim an address
  - RFC3971 also allows a node to use non-CGAs with certificates that authorize their use. The details of such use are beyond the scope of this specification and are left for future work
- RSA (RSA Encryption Standard) Signature option, is used to protect all messages relating to Neighbor and Router discovery
  - Public key signatures protect the integrity of the messages and authenticate the identity of their sender
  - The RSA Signature option allows public key-based signatures to be attached to NDP messages
- Timestamp and Nonce Options are introduced in order to prevent replay attacks
  - The Timestamp option offers replay protection without any previously established state or sequence numbers. For example, It can be used when Neighbor and Router Discovery messages are sent in some cases to multicast addresses
  - The Nonce option protects the messages used in solicitation-advertisement pairs
    - Nonce is an unpredictable random or pseudo-random number generated by a node and used exactly once. In SEND, nonces are used to assure that a particular advertisement is linked to the solicitation that triggered it

Network Management

MIB Textual Conventions

- Internet addresses on devices that connect multiple zones are not necessarily unique, so an additional zone index is needed to select an interface:
  - InetAddressIPv6z
- If no zone index is needed then:
  - InetAddressIPv6
- To support arbitrary combinations of scoped Internet addresses, MIB authors should use a separate InetAddressType object for each InetAddress object
- RFC4001

Textual Convention for IPv6 Flow Label

- Two definitions:
  - IPv6FlowLabel
  - IPv6FlowLabelOrAny (-1 used as a wildcard, any value)
- RFC3595

DNS Issues

DNS Extensions to Support IPv6

- RFC3596 defines:
  - A RR type (AAAA) to map a domain name to an IPv6 address
  - A domain to support lookups based on addresses
  - Modification of existing queries that perform additional section processing to locate IPv4 addresses (now do both, IPv4 and IPv6)
- IP protocol version used for the query is independent of the protocol version of the RRs (transport vs. contents)

IP6.ARPA

- Special domain defined to look up a record given an IPv6 address, in order to provide a way of mapping an IPv6 address to a host name
- An IPv6 address is represented as a name in the IP6.ARPA domain by a sequence of nibbles (encoded in reverse order, low-order nibble first), represented in hexadecimal, separated by dots with the suffix ".IP6.ARPA"
- Example:
  - 2001:db8:1:2:3:4:567:89ab, would be
  - b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.

Modifications to Existing Queries

- All existing query types that perform type A additional section processing (such as NS, SRV and MX), must be redefined to perform both type A and AAAA additional section processing
- Consequently, a name server must add any relevant IPv4 and IPv6 addresses available locally to the additional section of a response when processing any one of the above queries

Application Issues

The Porting Issue

- Network layer change is not transparent
  - IPv4 applications need to be modified for IPv6
- Best practice is to turn IPv4 apps into protocol-independent apps
- Usually not difficult
  - Simple apps (e.g. telnet) take only hours to port
- Care should be given to how IPv4 or IPv6 protocols are preferred and selected when both available
  - Apps may need to iterate connection attempts due to multiple IPv6 addresses available (or both, IPv4+IPv6)

Main Changes From IPv4

- Address Size
  - 32 bits (IPv4) to 128 bits (IPv6)
- Implications on “display” of addresses
  - Up to 39 characters, even 45 for IPv4-mapped
- API changes
  - Address size issues
  - Protocol independence
- Dependencies on IP header size
- Dependencies on particular addresses
- **STRONG** recommendation to use FQDNs rather than IP addresses
  - Also store FQDNs preferred vs. storage of addresses

Not All Applications Need to be Changed

- Many applications don’t talk to the network directly, but rather use library functions to carry out those tasks. In some cases, only the underlining library needs to be changed
- Examples:
  - RPC
  - DirectPlay

IPv6 APIs

- Basic Socket Interface Extensions for IPv6 (RFC3493)
- Advanced Sockets Application Program Interface (API) for IPv6 (RFC3542)

Address Storage Issues

- Problem: you can’t store a 128 bit value in a 32 bit space
- Most applications today store and reference IP addresses as either:
  - sockaddrs (good)
  - in_addrs (okay)
  - ints (bad)
- Storage versus reference

Anatomy of a sockaddr

```
struct sockaddr {
    u_short sa_family; // Address family
    char sa_data[14];  // Address data
};
```

- The sa_family field contains a value which indicates which type of address this is (IPv4, IPv6, etc)

sockaddr_in

```
struct sockaddr_in {
    short sin_family;
    u_short sin_port;
    struct in_addr
  sin_addr;
    char sin_zero[8];
};
```

sockaddr_in6

```
struct sockaddr_in6 {
    short sin6_family; //
  AF_INET6
    u_short sin6_port;
    u_long sin6_flowinfo;
    struct in_addr6 sin6_addr;
    u_long sin6_scope_id;
};
```

API Changes

- Most of the socket APIs don’t need to change – they were originally designed to be protocol independent, and thus take pointers to sockaddrs as input or output
  - bind, connect, getsockname, getpeername, etc.
- The name resolution APIs are the big offenders that need to be changed
  - gethostbyname, gethostbyaddr

New Name Resolution APIs

- getaddrinfo – for finding the addresses and/or port numbers that corresponds to a given host name and service
- getnameinfo – for finding the host name and/or service name that corresponds to a given address or port number
- Both of these APIs are protocol-independent – they work for both IPv4 and IPv6

Getaddrinfo

```
int
getaddrinfo(
    IN const char FAR * nodename,
    IN const char FAR * servicename,
    IN const struct addrinfo FAR * hints,
    OUT struct addrinfo FAR * FAR * res
);
```

Anatomy of Addrinfo

```
typedef struct addrinfo {
    int ai_flags;
    int ai_family;   // PF_xxx.
    int ai_socktype; // SOCK_xxx.
    int ai_protocol; // IPPROTO_xxx.
    size_t ai_addrlen;
    char *ai_canonname;
    struct sockaddr *ai_addr;
    struct addrinfo *ai_next;
} ADDRINFO, FAR * LPADDRINFO;
```

Getnameinfo

```
int
getnameinfo(
    IN  const struct sockaddr FAR * sa,
    IN  socklen_t salen,
    OUT char FAR * host,
    IN  DWORD hostlen,
    OUT char FAR * service,
    IN  DWORD servlen,
    IN  int flags
);
```

Header Size Dependencies

- Problem: The IPv6 header is 20 bytes larger than (the minimal) IPv4 header
- Programs that calculate their datagram payload size by computing MTU – (UDP header size + IP header size) need to know that the IP header size has changed

IPv4 Address Dependencies

- Some programs “know” certain addresses (e.g. loopback = IPv4 address 127.0.0.1)
- Programs whose purpose is to manipulate addresses (e.g. Network Address Translators, or NATs) obviously have innate knowledge of IPv4 addresses
- Only an issue for those sorts of programs

IPv6 in Java

- A good example (JDK >1.4/1.5)
- Java hides the dual stack almost completely

Application Aspects of IPv6 Transition (RFC4038)

- The document introduces how to enable IPv6 support in applications running on IPv6 hosts, and the best strategy to develop IP protocol support in applications
- Specifies scenarios and aspects of application transition
- It also proposes guidelines on how to develop IP version-independent applications during the transition period

Transition Issues

Transition / Co-Existence

- A wide range of techniques have been identified and implemented, basically falling into three categories:
  - Dual-stack techniques, to allow IPv4 and IPv6 to co-exist in the same devices and networks
  - Tunneling techniques, to avoid order dependencies when upgrading hosts, routers, or regions
  - Translation techniques, to allow IPv6-only devices to communicate with IPv4-only devices

Transition Considerations

- Design for an indefinite period of time of IPv4-IPv6 co-existence
- RFC4001 is a good example about doing that

Default Address Selection (RFC3484)

- Algorithms for source and destination address selection
- Specify default behavior for all Internet Protocol version 6 (IPv6) implementations. They do not override choices made by applications or upper-layer protocols, nor do they preclude the development of more advanced mechanisms for address selection
- The two algorithms share a common context, including an optional mechanism for allowing administrators to provide policy that can override the default behavior.
- In dual stack implementations, the destination address selection algorithm can consider both IPv4 and IPv6 addresses -depending on the available source addresses, the algorithm might prefer IPv6 addresses over IPv4 addresses, or vice-versa

Other Issues

IPv6 Documentation Addresses

- Global unicast address prefix reserved for documentation purposes (see RFC 3849)
  - 2001:DB8::/32

Literal IPv6 Addresses in URLs

- RFC3986 defines a syntax for IPv6 addresses to be used in browsers, avoiding confusion with the port separator (:)
- The literal address should be enclosed in "[" and "]" characters
- Example:
  – http://[2001:DB8::367]:80/index.html
- Fully Qualified Domain Names (FQDNs) should be used in preference to IP addresses whenever possible
- Use of *localhost* by name abstracts the difference between IPv4 and IPv6

Literal IPv6 Addresses in SMTP

- RFC2821 defines how to use literal addresses in SMTP, enclosed in square brackets:
  - IPv6-address-literal = “IPv6:” IPv6-addr
- Example:
  - [IPv6:2001:db8::1]

Mobile IPv6 ?

- Route optimization

Useful Web Sites

- http://www.ipv6-to-standard.org
- http://www.ipv6tf.org

Acknowledgments

- Edu-Team members who contributed to produce this class
  - Brian Carpenter
  - Margaret Wasserman