

# What are the technical benefits of implementing IPv6

Jordi Palet (jordi.palet@consulintel.es)

CEO / CTO

*Consulintel*

# Agenda

1. Header Formats & Packet Size Issues
2. Quality of Service
3. Mobility
4. Multihoming
5. Porting Applications to IPv6



# 1. Header Formats & Packet Size Issues

# RFC2460

- Internet Protocol, Version 6: Specification
- Changes from IPv4 to IPv6:
  - Expanded Addressing Capabilities
  - Header Format Simplification
  - Improved Support for Extensions and Options
  - Flow Labeling Capability
  - Authentication and Privacy Capabilities



# Agenda

**2.1. Terminology**

**2.2. IPv6 Header Format**

**2.3. Packet Size Issues**

**2.4. Upper-Layer Protocol Issues**



## 2.1. Terminology

# Terminology

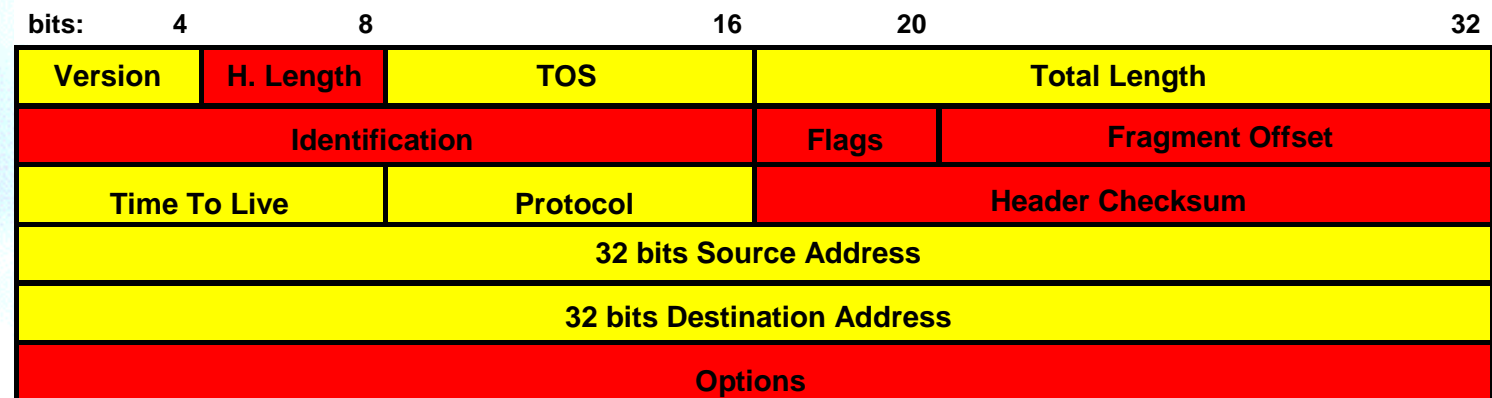
- **Node:** Device that implements IPv6
- **Router:** Node that forwards IPv6 packets
- **Host:** Any node that isn't a router
- **Upper Layer:** Protocol layer immediately above IPv6
- **Link:** Communication Facility or Medium over which nodes can communicate at the link layer
- **Neighbors:** Nodes attached to the same link
- **Interface:** A node's attachment to a link
- **Address:** An IPv6-layer identification for an interface or a set of interfaces
- **Packet:** An IPv6 header plus payload
- **Link MTU:** Maximum Transmission Unit
- **Path MTU:** Minimum link MTU of all the links in a path between source and destination node's

## 2.2. IPv6 Header Format



# IPv4 Header Format

- 20 Bytes + Options



# IPv6 Header Format

- From 12 to 8 Fields (40 bytes)

bits:	4	12	16	24	32
Version	Class of Traffic	Flow Label			
Payload Length			Next Header	Hop Limit	
128 bits Source Address					
128 bits Destination Address					

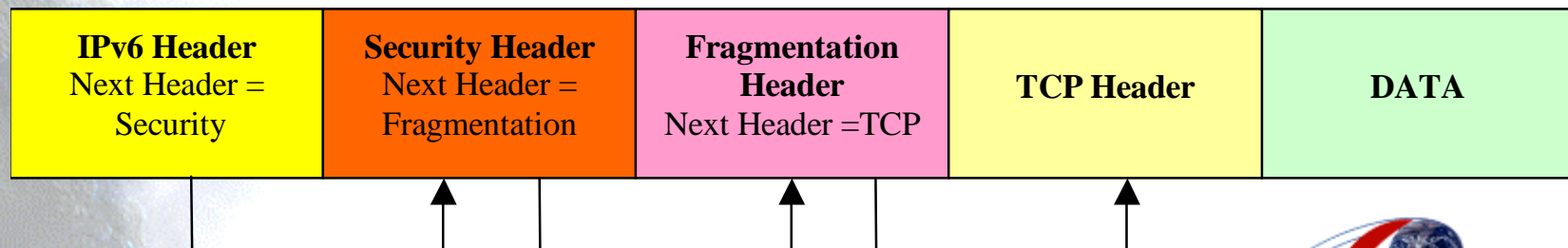
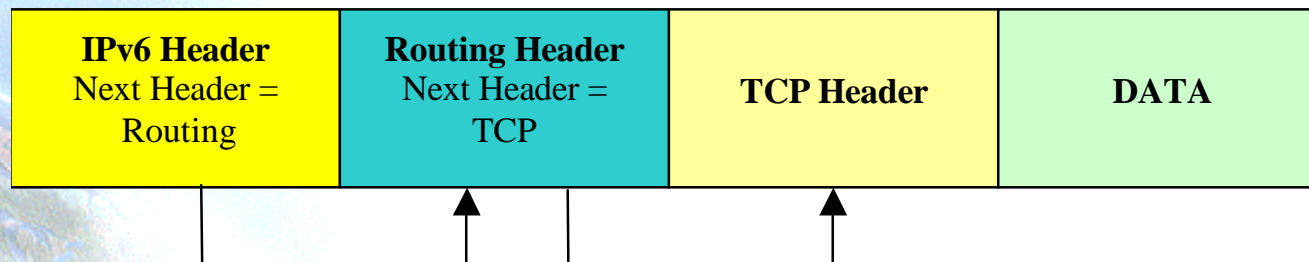
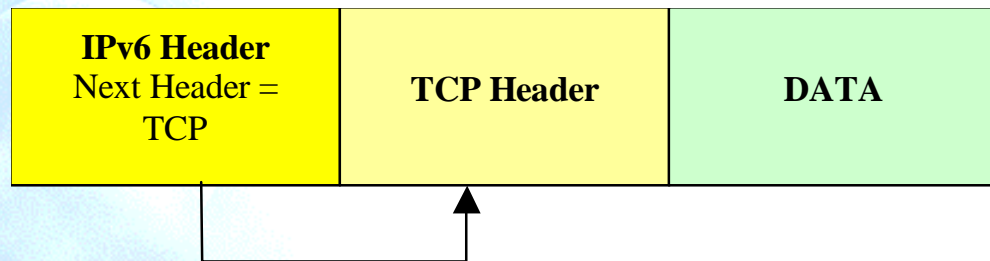
- Avoid checksum redundancy
- Fragmentation end to end

# 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)
- 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





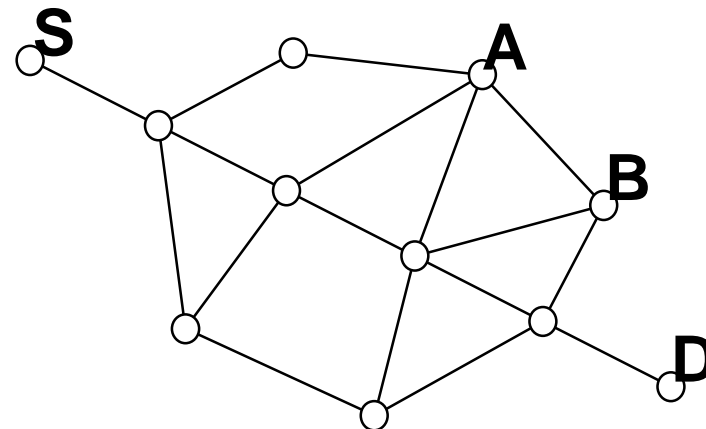
# Extension Headers Goodies

- Processed Only by Destination Node
  - Exception: Hop-by-Hop Options Header
- No more “40 byte limit” on options (IPv4)
- Extension Headers defined currently:
  - Hop-by-Hop Options
  - Routing
  - Fragment
  - Authentication (RFC 2402, next header = 51)
  - Encapsulating Security Payload (RFC 2406, next header = 50)
  - Destination Options

# Example: Using the Routing Header

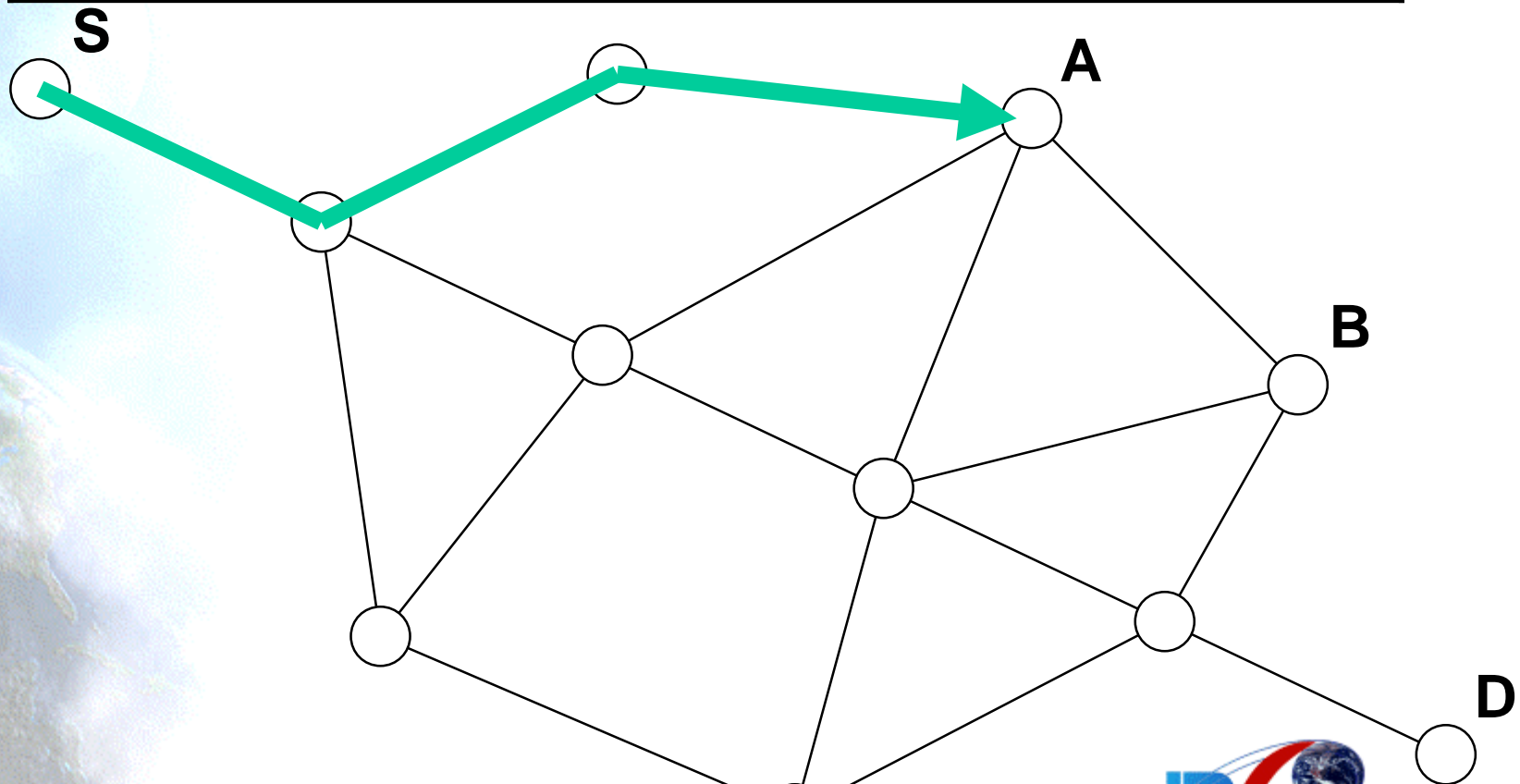
8 bits	8 bits unsigned	8 bits	8 bits unsigned
Next Header	H. Ext. Length	Routing Type = 0	Segments Left
Reserved = 0			
Address 1			
Address 2			
...			
Address n			

- Next Header value = 43
- A type 0 routing header, where:
  - Source Node: S
  - Destination: D
  - Intermediate Nodes: A & B



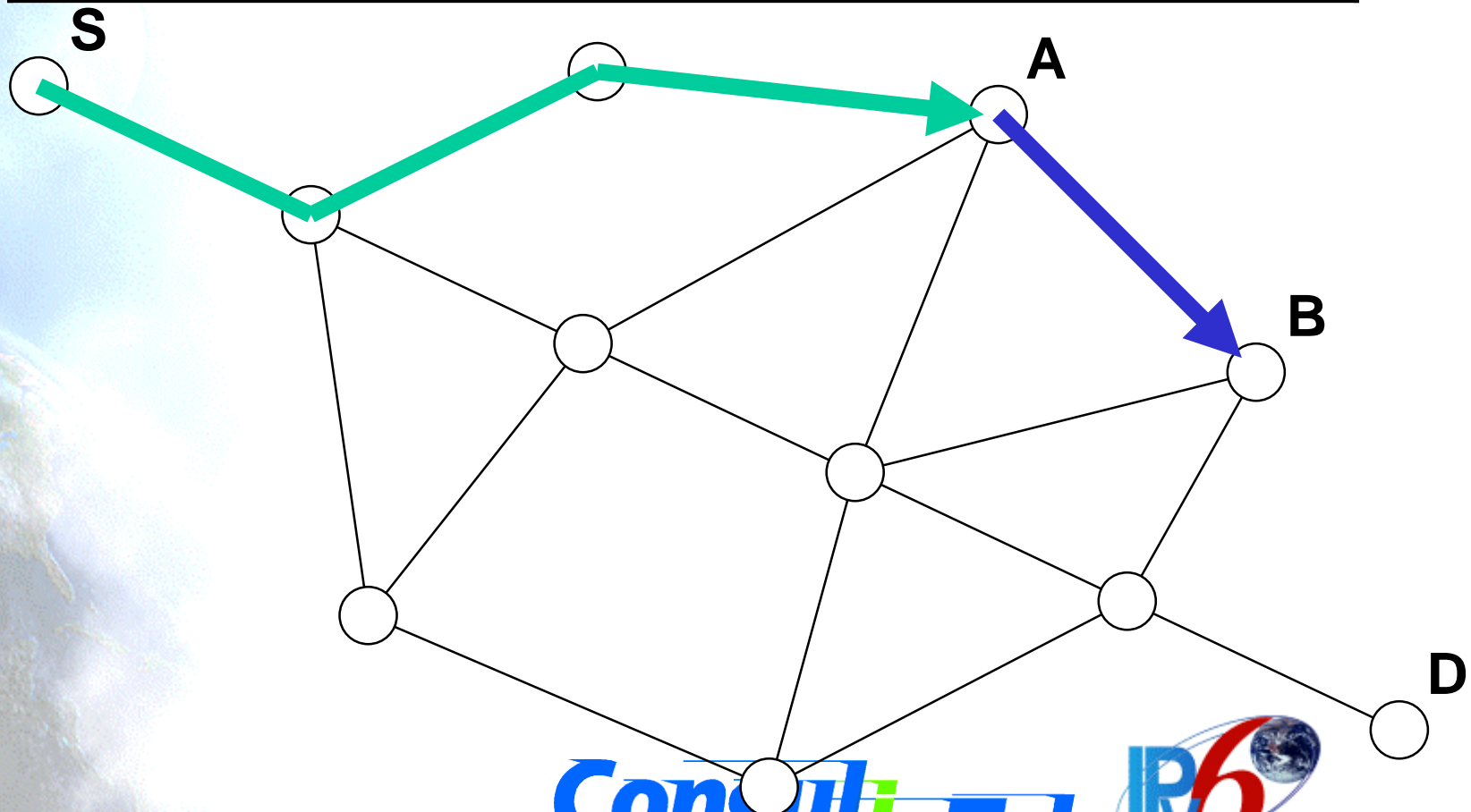
# Example: Headers when S to A

IPv6 Base Header	Routing Header
Source Address = S Destination Address = A	H. Ext. Length = 4 Segments Left = 2 Address 1 = B Address 2 = D



# Example: Headers when A to B

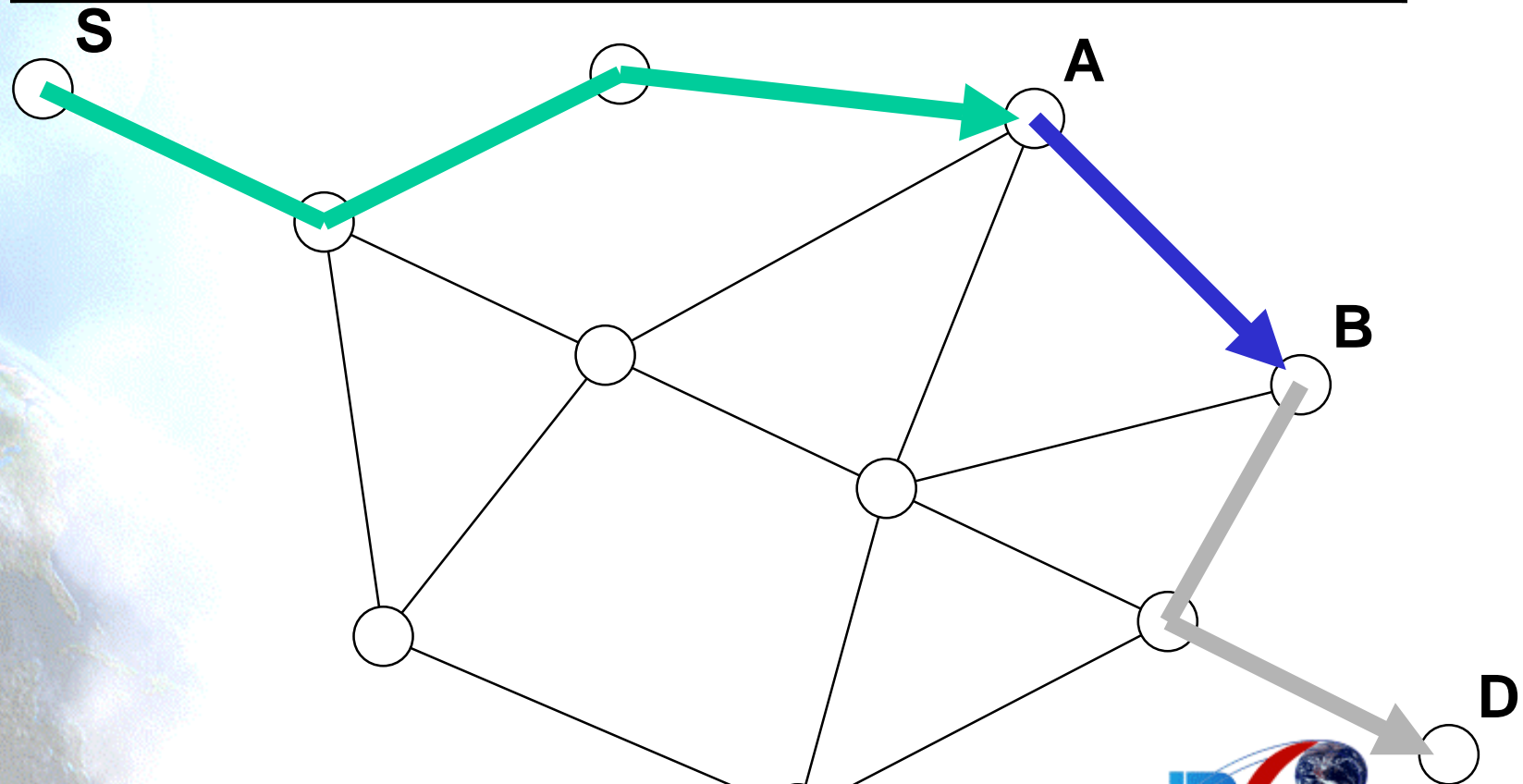
IPv6 Base Header	Routing Header
Source Address = S Destination Address = B	H. Ext. Length = 4 Segments Left = 1 Address 1 = A Address 2 = D





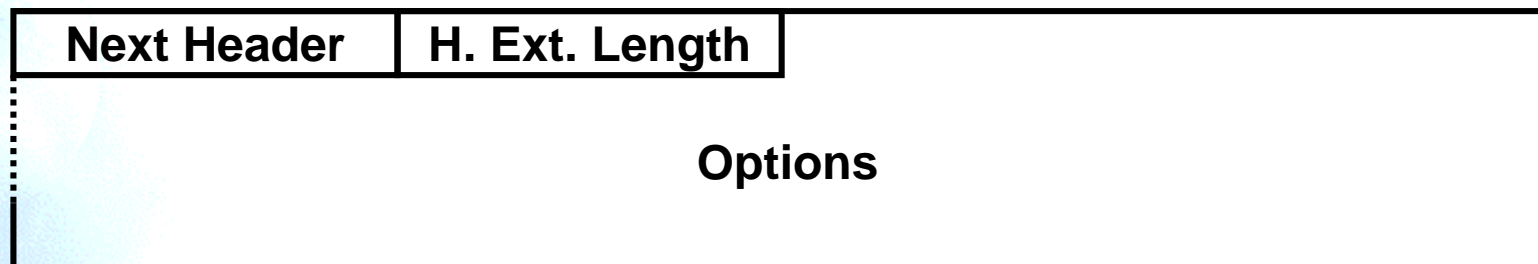
# Example: Headers when B to D

IPv6 Base Header	Routing Header
Source Address = S Destination Address = D	H. Ext. Length = 4 Segments Left = 0 Address 1 = A Address 2 = B



# Hop-by-Hop & Destination Options Headers

- “Containers” for variable-length options:

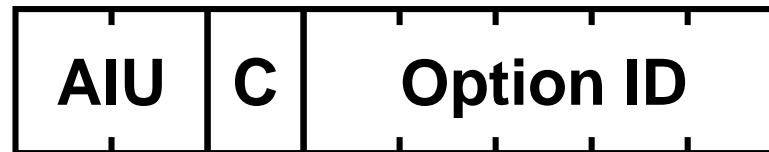


- Where Options =



- Next Header values:
  - 0 for Hop-by-Hop Options Header
  - 60 for Destination Options Header

# Option Type Encoding



AIU — action if unrecognized:

- 00 — skip over option
- 01 — discard packet
- 10 — discard packet &  
send ICMP Unrecognized Type to source
- 11 — discard packet &  
send ICMP Unrecognized Type to source  
only if destination was not multicast
- C — set (1) if Option Data changes en-route  
(Hop-by-Hop Options only)

# Option Alignment and Padding

Two Padding Options:

Pad1 

0
---

 ← special case: no Length or Data fields

PadN 

1	N - 2
---	-------

 ..... N-2 zero octets...

- Used to align options so multi-byte data fields fall on natural boundaries
- Used to pad out containing header to an integer multiple of 8 bytes



# Fragment Header

- Used by an IPv6 Source to send a packet larger than would fit in the path MTU to its Destination.
- In IPv6 the Fragmentation is only performed by source nodes, not routers.
- Next Header value = 44

8 bits	8 bits	13 bits unsigned	2 bits	1 bit
Next Header	Reserved = 0	Fragment Offset	Res. = 0	M
Identification				

- Original Packet (unfragmented):

Unfragmentable Part	Fragmentable Part
---------------------	-------------------

# Fragmentation Process

- The Fragmentable Part of the original packet is divided into fragments, each, except possibly the last ("rightmost") one, being an integer multiple of 8 octets long. The fragments are transmitted in separate "fragment packets"

Unfragmentable Part	1 <sup>st</sup> Fragment	2 <sup>nd</sup> Fragment	...	Last Fragment
---------------------	--------------------------	--------------------------	-----	---------------

- Fragment Packets:

Unfragmentable Part	Fragment Header	1 <sup>st</sup> Fragment
Unfragmentable Part	Fragment Header	2 <sup>nd</sup> Fragment
...		
Unfragmentable Part	Fragment Header	Last Fragment

## 2.3. Packet Size Issues

# Minimum MTU

- Link MTU:
  - A link's maximum transmission unit, i.e., the max IP packet size that can be transmitted over the link
- Path MTU:
  - The minimum MTU of all the links in a path between a source and a destination
- Minimum link MTU for IPv6 is 1280 octets vs. 68 octets for v4
- On links with MTU < 1280, link-specific fragmentation and reassembly must be used
- On links that have a configurable MTU, it's recommended a MTU of 1500 bytes



# Path MTU Discovery (RFC1981)

- Implementations are expected to perform path MTU discovery to send packets bigger than 1280 octets:
  - for each destination, start by assuming MTU of first-hop link
  - if a packet reaches a link in which it can't fit, will invoke ICMP “packet too big” message to source, reporting the link's MTU; MTU is cached by source for specific destination
  - occasionally discard cached MTU to detect possible increase
- Minimal implementation can omit path MTU discovery as long as all packets kept  $\leq 1280$  octets
  - e.g., in a boot ROM implementation

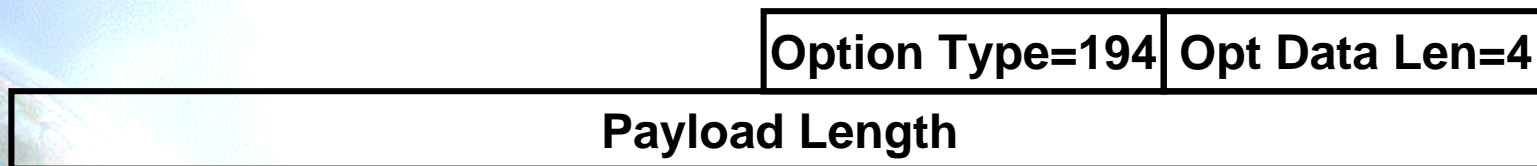
# Fragment Header

Next Header	Reserved	Fragment Offset	0 0 M
Original Packet Identifier			

- Though discouraged, can use IPv6 Fragment header to support upper layers that do not (yet) do path MTU discovery
- IPv6 fragmentation & reassembly is an end-to-end function; routers do not fragment packets en-route if too big, instead, they send ICMP “packet too big”.

# Maximum Packet Size

- Base IPv6 header supports payloads of up to 65,535 bytes (not including 40 byte IPv6 header)
- Bigger payloads can be carried by setting IPv6 Payload Length field to zero, and adding the “jumbogram” hop-by-hop option:



- Can't use Fragment header with jumbograms (RFC2675)

## 2.4. Upper-Layer Protocol Issues



# Upper-Layer Checksums

- 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.
- TCP/UDP “pseudo-header” for IPv6:

Source Address	
Destination Address	
Upper-Layer Packet Length	
zero	Next Header

- ICMPv6 includes the above pseudo-header in its checksum computation (change from ICMPv4). Reason: Protect ICMP from misdelivery or corruption of those fields of the IPv6 header on which it depends, which, unlike IPv4, are not covered by an internet-layer checksum. The Next Header field in the pseudo-header for ICMP contains the value 58, which identifies the IPv6 version of ICMP.

# Maximum Packet Lifetime

- IPv6 nodes are not required to enforce maximum packet lifetime.
- That is the reason the IPv4 "Time to Live" field was renamed "Hop Limit" in IPv6.
- In practice, very few, if any, IPv4 implementations conform to the requirement that they limit packet lifetime, so this is not a "real" change.
- Any upper-layer protocol that relies on the internet layer (whether IPv4 or IPv6) to limit packet lifetime ought to be upgraded to provide its own mechanisms 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 relative to the IPv4 header.
- Example: in IPv4, TCP's MSS option is computed as the maximum packet size (a default value or a value learned through Path MTU Discovery) minus 40 octets (20 octets for the minimum-length IPv4 header and 20 octets for the minimum-length TCP header). When using TCP over IPv6, the MSS must be computed as the maximum packet size minus 60 octets, because the minimum-length IPv6 header (i.e., an IPv6 header with no extension headers) is 20 octets longer than a minimum-length IPv4 header.



# Responding to Packets Carrying Routing Headers

- When an upper-layer protocol sends one or more packets in response to a received packet that included a Routing header, the response packet(s) must not include a Routing header that was automatically derived by "reversing" the received Routing header UNLESS the integrity and authenticity of the received Source Address and Routing header have been verified (e.g., via the use of an Authentication header in the received packet).



## 2. Quality of Service

# Concept of QoS

- Quality: Reliable delivery of data (“better than normal”)
  - Data loss
  - Latency
  - Jittering
  - Bandwidth
- Service: Anything offered to the user
  - Communication
  - Transport
  - Application

# Abstract

- “Quality of Service is a measurement of the network behavior with respect to certain characteristics of defined services” !!!!!
- Common concepts to all definitions of QoS:
  - Traffic and type of service differentiation
  - Users may be able to treat one or more traffic classes differently

# IP Quality of Service Approaches

Two basic approaches developed by IETF:

- “Integrated Service” (int-serv)
  - fine-grain (per-flow), quantitative promises (e.g., x bits per second), uses RSVP signalling
- “Differentiated Service” (diff-serv)
  - coarse-grain (per-class), qualitative promises (e.g., higher priority), no explicit signalling



# IPv6 Support for Int-Serv

20-bit Flow Label field to identify specific flows needing special QoS

- each source chooses its own Flow Label values; routers use Source Addr + Flow Label to identify distinct flows
- Flow Label value of 0 used when no special QoS requested (the common case today)
- this part of IPv6 is not standardized yet, and may well change semantics in the future

# IPv6 Support for Diff-Serv

8-bit Traffic Class field to identify specific classes of packets needing special QoS

- same as new definition of IPv4 Type-of-Service byte
- may be initialized by source or by router enroute; may be rewritten by routers enroute
- traffic Class value of 0 used when no special QoS requested (the common case today)



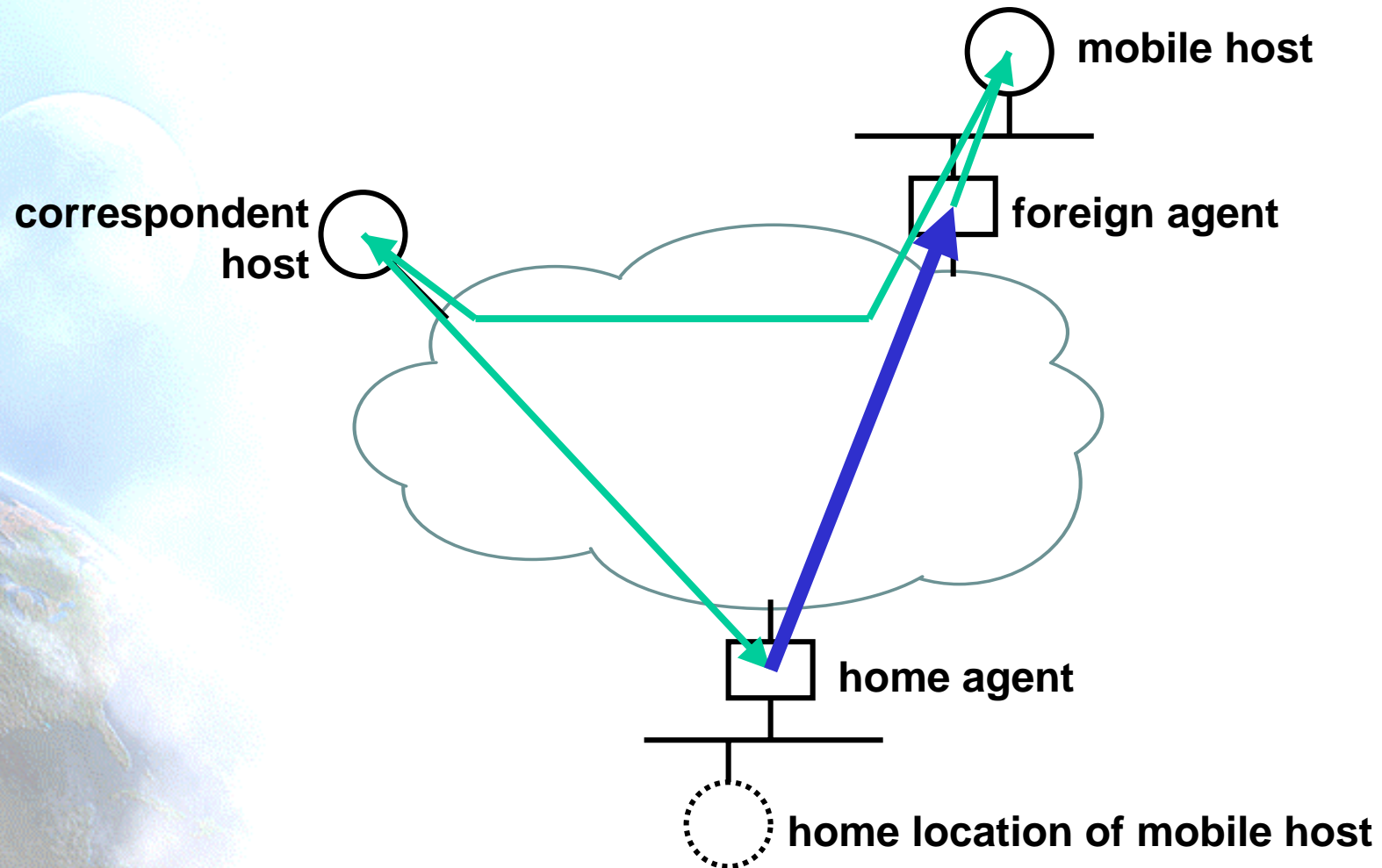
## 3. Mobility

# IPv6 Mobility

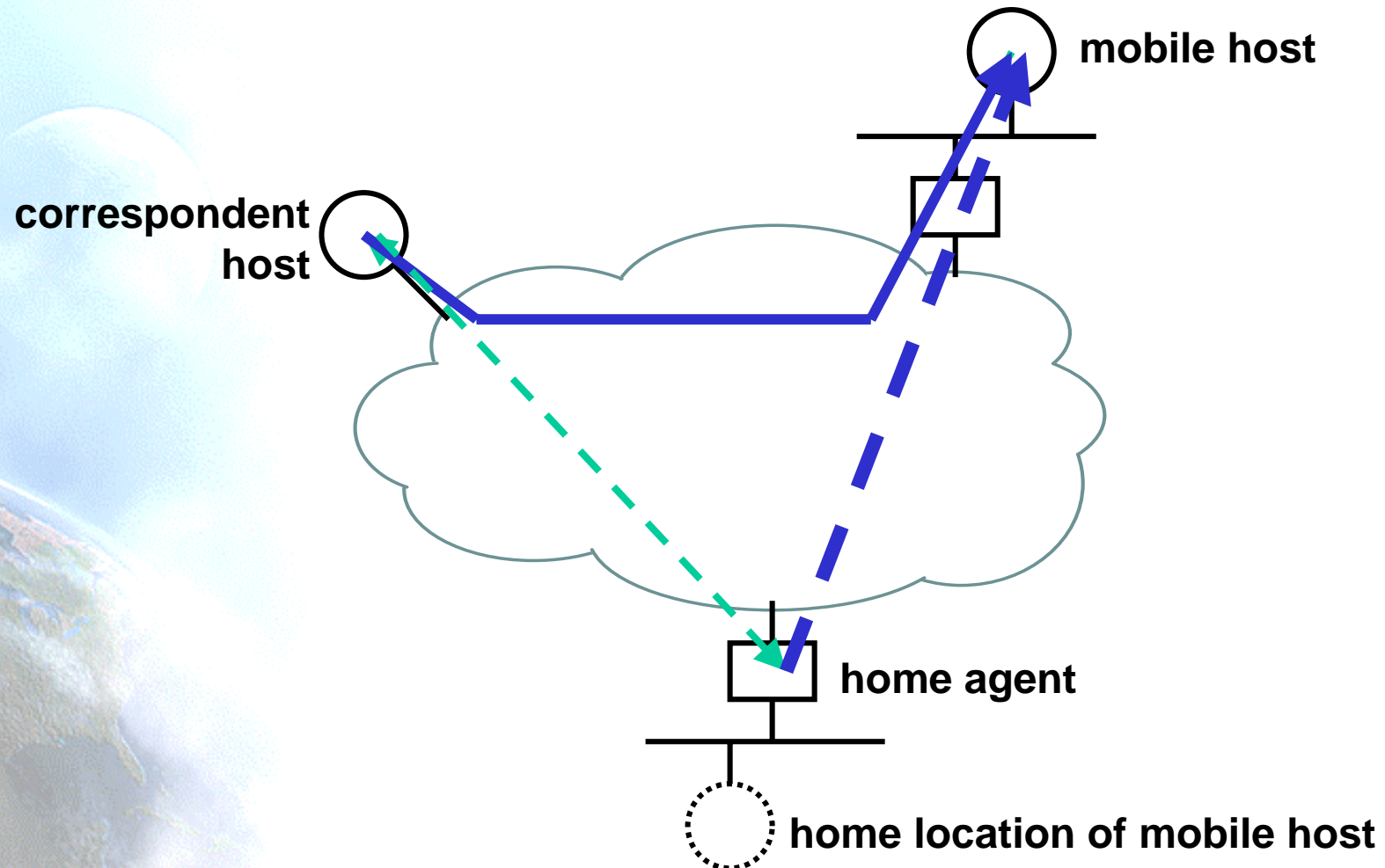
- A mobile host has one or more home address(es)
  - relatively stable; associated with host name in DNS
- When it discovers it is in a foreign subnet (i.e., not its home subnet), it acquires a foreign address
  - uses auto-configuration to get the address
  - registers the foreign address with a home agent, i.e, a router on its home subnet
- Packets sent to the mobile's home address(es) are intercepted by home agent and forwarded to the foreign address, using encapsulation

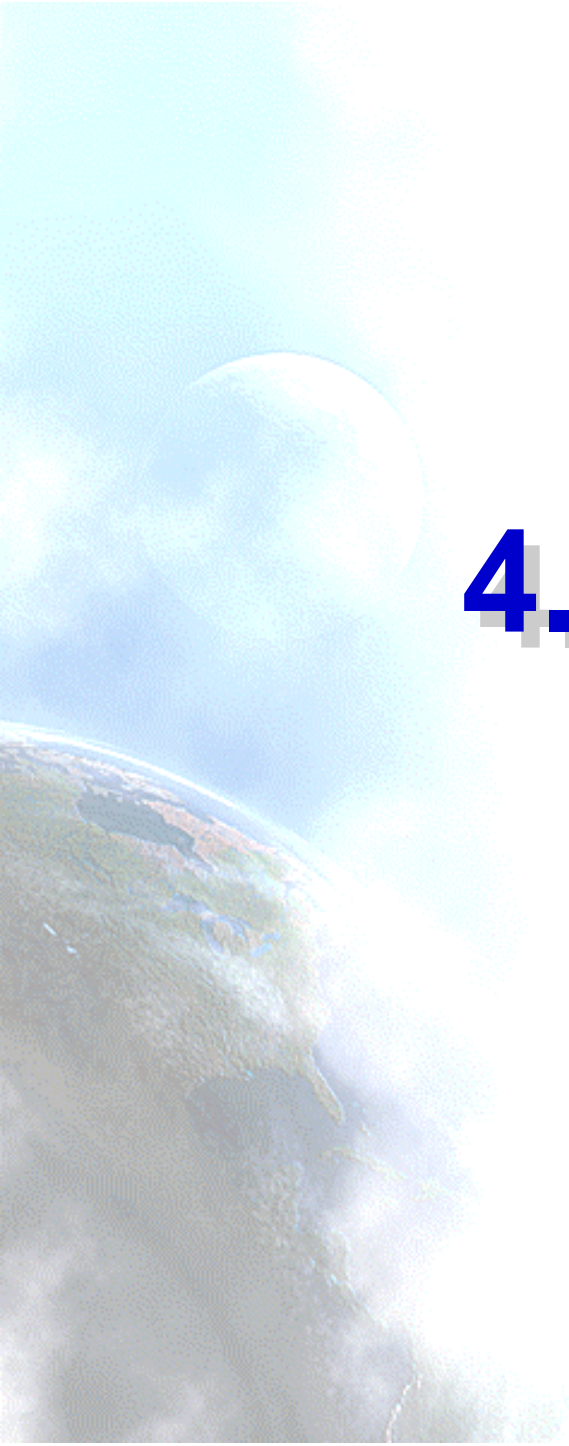


# Mobile IP (v4 version)



# Mobile IP (v6 version)





## 4. Multi-Homing

# Motivations

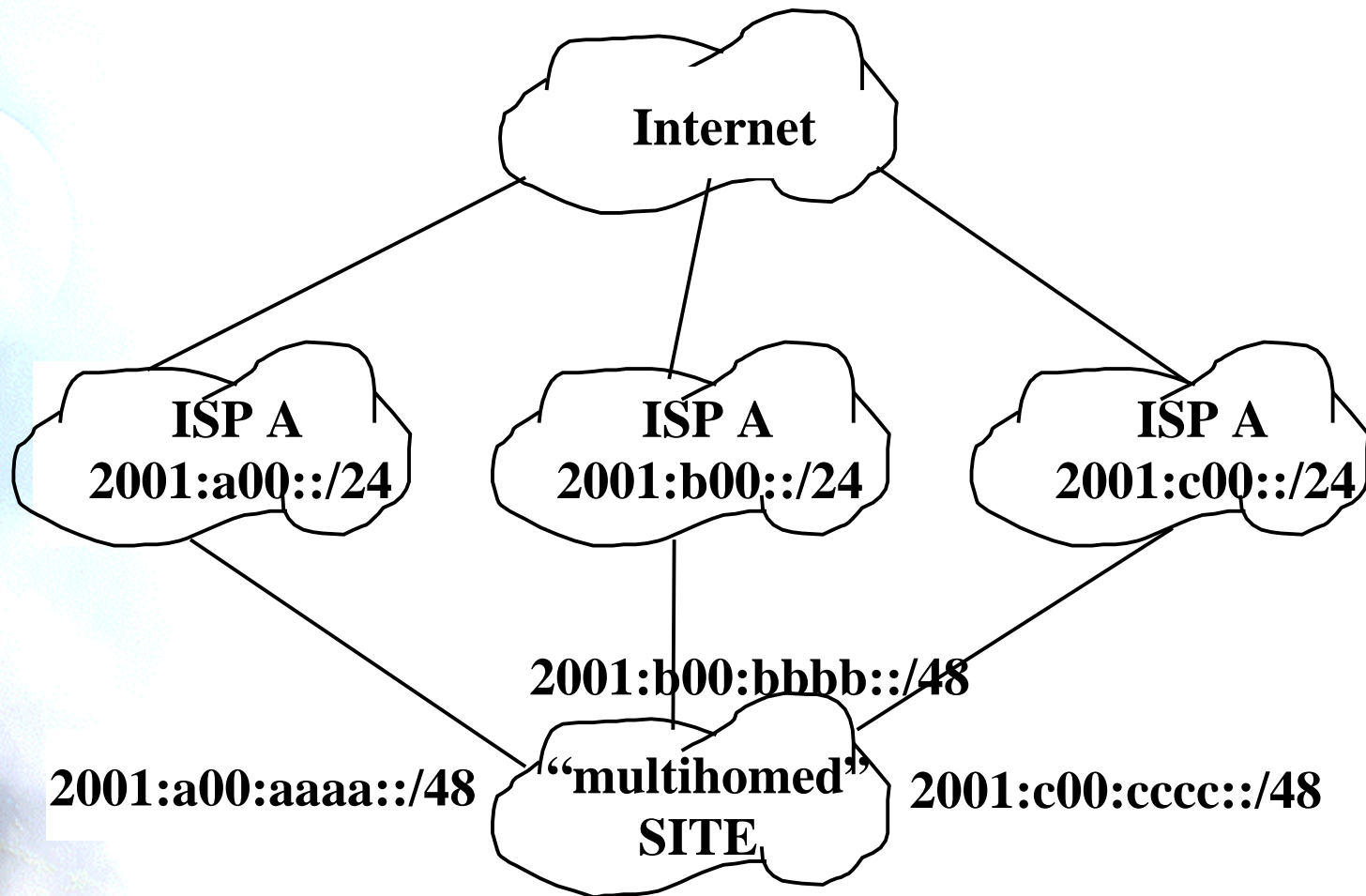
- Connectivity to more than one Internet Service Providers (ISPs) is becoming relevant, for the purpose of redundancy and traffic load distribution.
- Large numbers of Multi-homed sites impose a direct challenge on routing aggregation and consequently on global Internet routing scalability.
- IPv6 multihoming also:
  - Provides redundancy and load sharing for the multi-homed sites
  - Facilitate the scalability of the global IPv6 Internet routing table
  - Is simple and operationally manageable.
- Uses existing routing protocol and implementation thus no new protocol or changes are needed.



# Multihoming Mechanism

- Two types of multihoming connections:
  1. Site multi-homed to a single ISP, commonly at different geography locations
  2. Site multi-homed to more than one ISPs.
- The specific routes associated with the multihomed site 1 will not be visible outside of the particular ISP network and thus there is no real impact on the global routing: No special mechanism is needed for multihoming in this scenario.
- To obtain IP addresses, a multi-homed site will designate one of its ISPs as its primary ISP and receive IP address assignment from the primary ISP's IPv6 aggregation block.

# Multi-Homing Example



# Current Status

- Available IPv6 multi-homing solution:
  - RFC2260 (same as IPv4): Tunnels, but limited fault tolerance, not provides load sharing, performance and policying.
- IETF multi6 Working Group:
  - Work in Progress (IPv6 site multihoming requirements).

# 5. Porting Applications to IPv6



# 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

# Main Changes From IPv4

- Address Size
  - 32 bits (IPv4) to 128 bits (IPv6)
- API changes
  - Address size issues
  - Protocol independence
- Dependencies on IP header size
- Dependencies on particular 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

# 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\_addr (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 * servcname,  
    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  $MTU - (UDP \text{ header size} + IP \text{ header size})$  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.
- NATs are evil anyway, so who cares.