Computer Networks

Ch.3 - IPv6

(bruno.quoitin@umons.ac.be)

Important note: These slides are partly based on a course by O. Bonaventure (UCLouvain) and the book "IPv6 Essentials" by S. Hagen.

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

Issues with IPv4 (1/3)

- Late 1980s
  - Exponential growth of the Internet

- 1990
  - Other network protocols exist (*AppleTalk*, *DECnet*, *IPX*, ...)
  - Governments (esp. US) push for *ConnectionLess Network Protocol* (CLNP, ISO standard)

- 1992
  - Most class B networks have been assigned
  - Class based routing failure
  - Networking experts warn that IPv4 address space could become exhausted

Issues with IPv4 (2/3)

- How to solve the exhaustion of class B addresses ?
- Short term solution
  - Define *Classless Interdomain Routing* (CIDR) and introduce the related changes in routers
  - Deployment started in 1994
- Long term solution
  - Develop Internet Protocol – next generation (IPng)
    - call for proposals RFC1550 (dec 1993)
    - criteria for choice, RFC1719/1726 (dec 1994)
    - several proposed solutions
      - TUBA, PIP, CATNIP, SIP, NIMROO, ENCAPS...

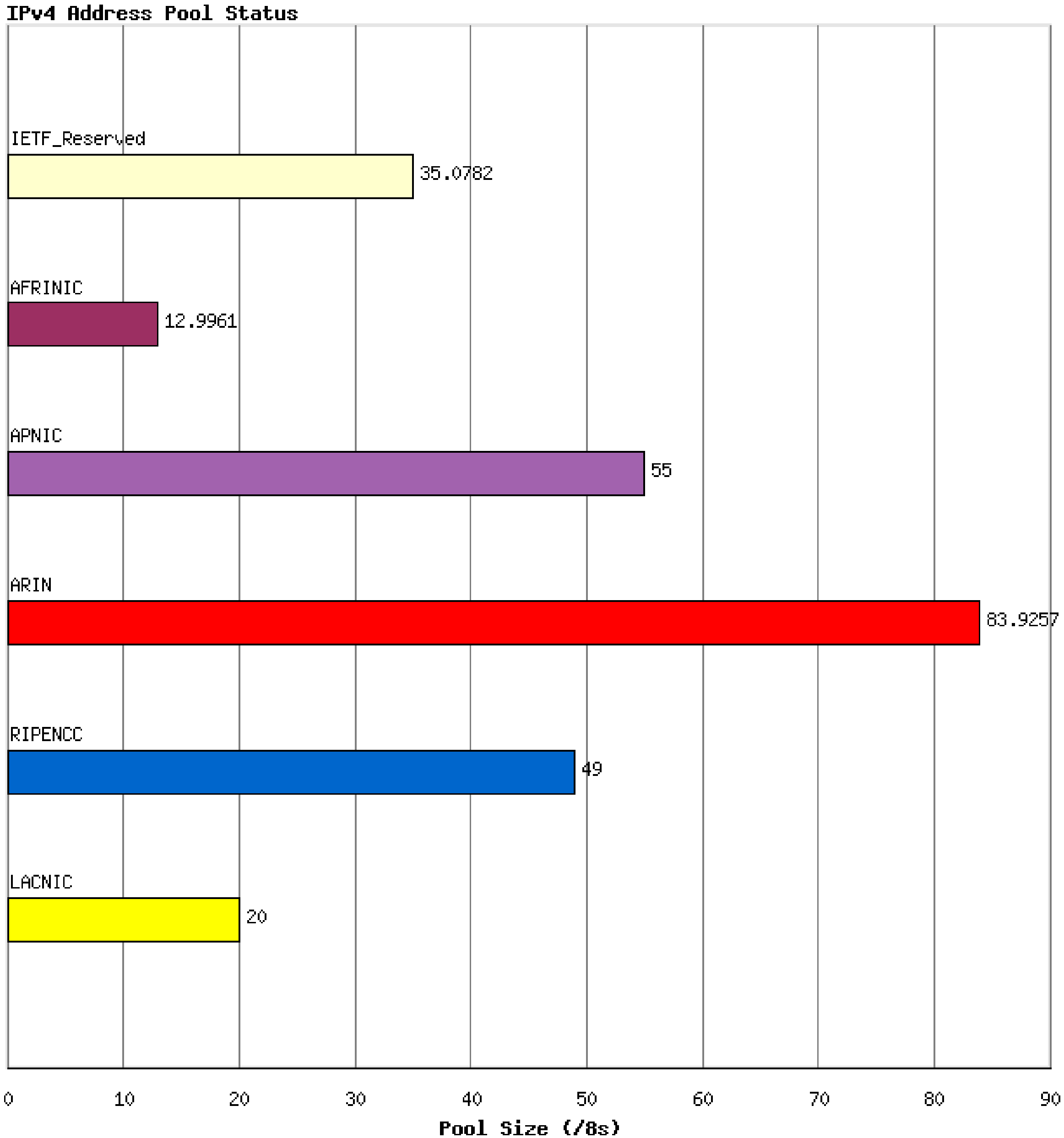
Issues with IPv4 (3/3)

- Implementation issues – 1990s
  - IPv4 packet format is complex
  - IP forwarding difficult to implement in hardware

- Missing functions – 1990s
  - IPv4 requires lots of **manual configuration**
    - competing protocols (CLNP, AppleTalk, IPX, ...) already supported auto-configuration in 1990s
  - How to support ***Quality of Service*** (QoS) in IP ?
    - IntServ and DiffServ did not exist then
  - How to better support **security** in IP ?
    - security problems started to appear (but less important than today)
  - How to better support **mobility** in IP ?
    - GSM started to appear and some were dreaming of mobile devices attached to the Internet...

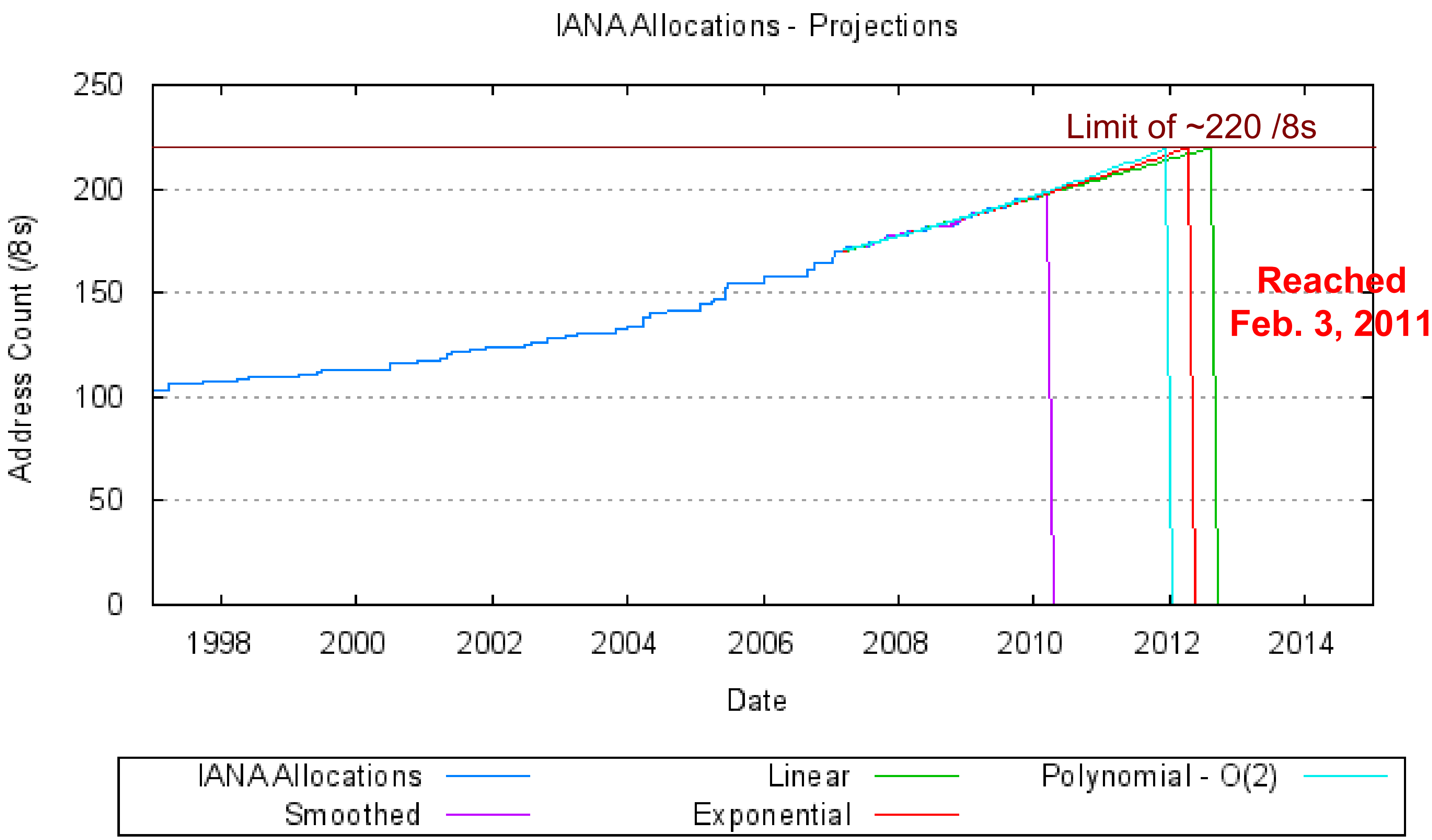
Source: http://xkcd.com/195/

IPv4 address allocation



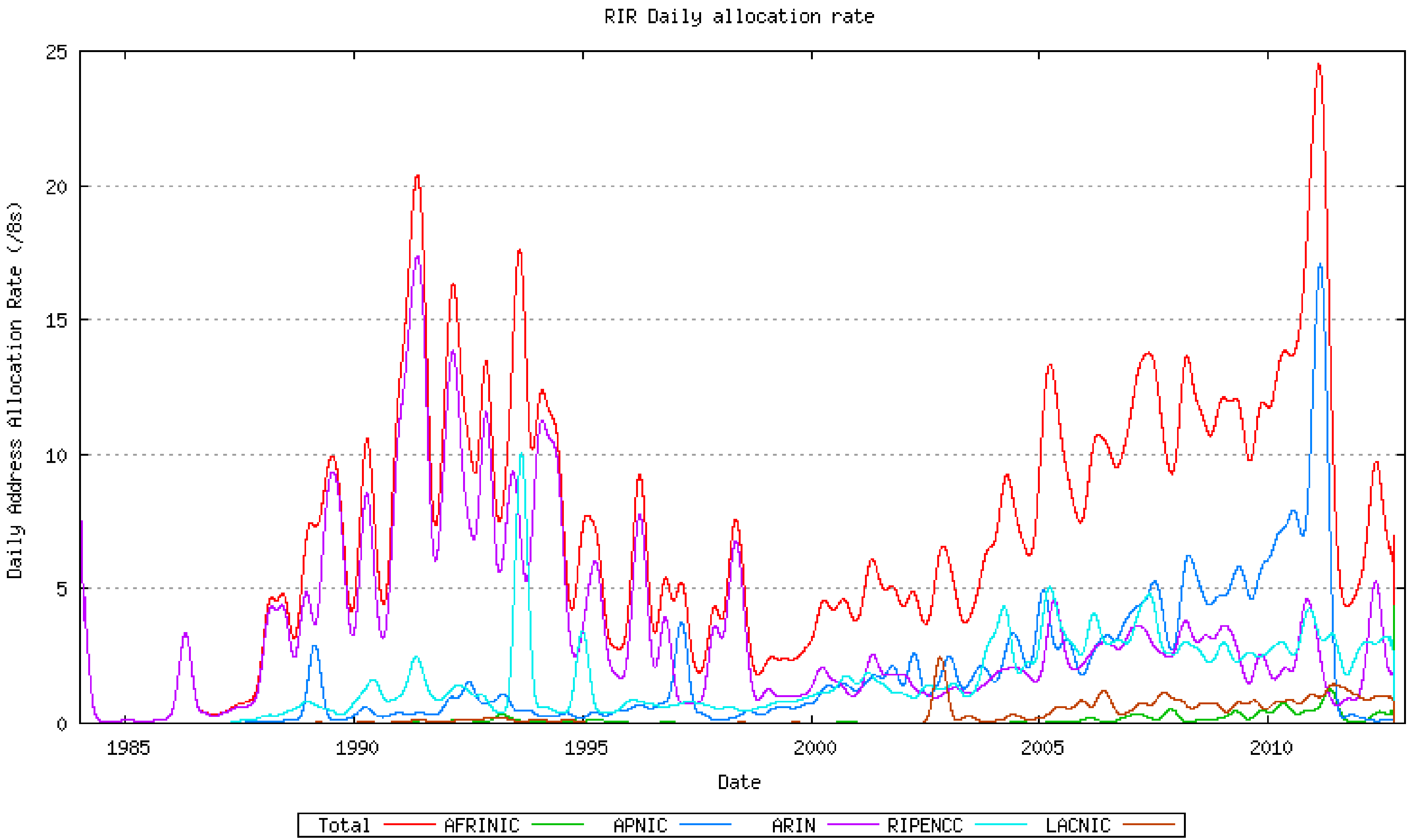
Source: http://www.potaroo.net/tools/ipv4/index.html
(retrieved on 24/10/2011)

IPv4 addr. exhaustion - prediction



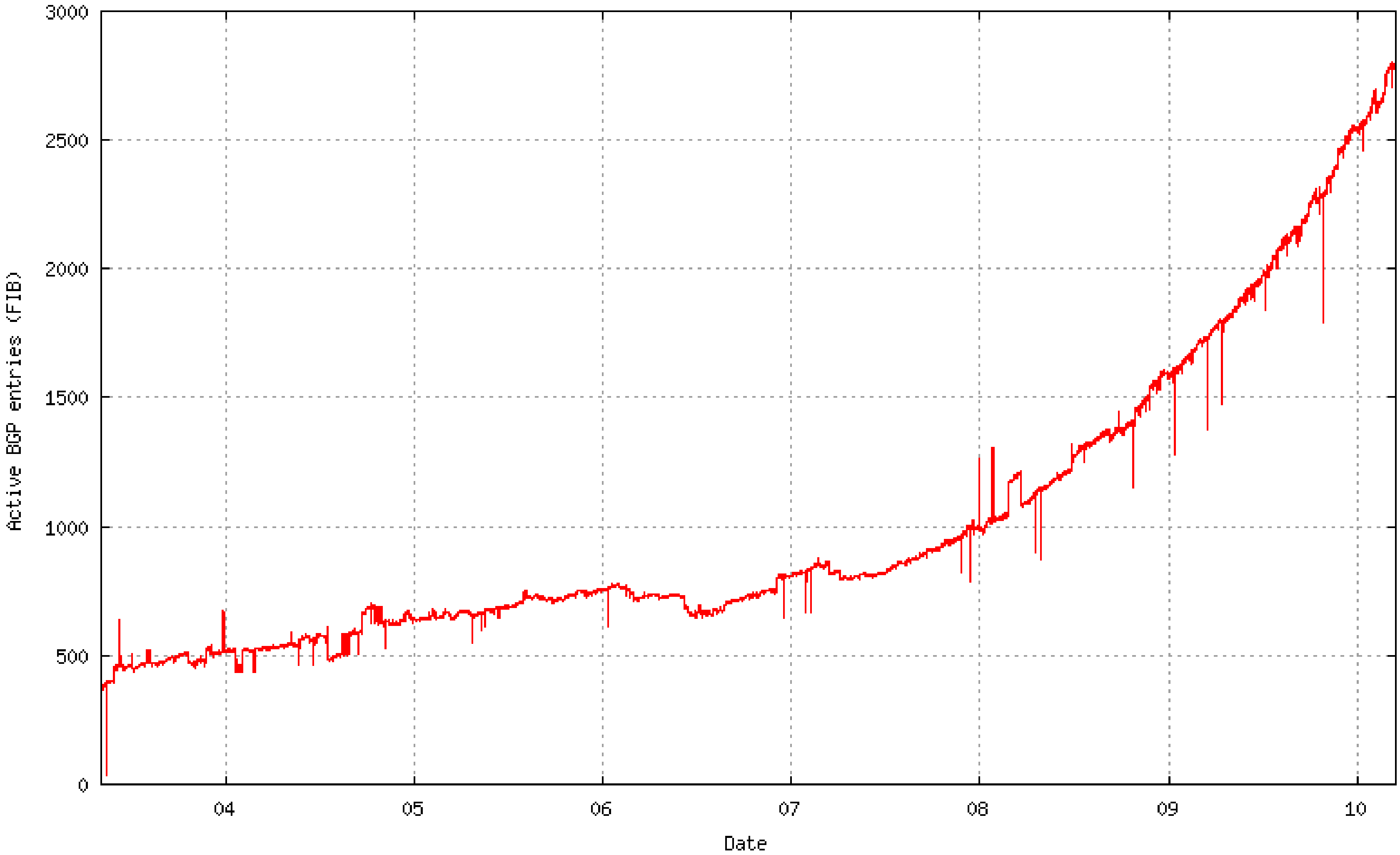
Source: http://www.potaroo.net/tools/ipv4/index.html
(retrieved on 17/03/2010)

IPv4 growth rate per region



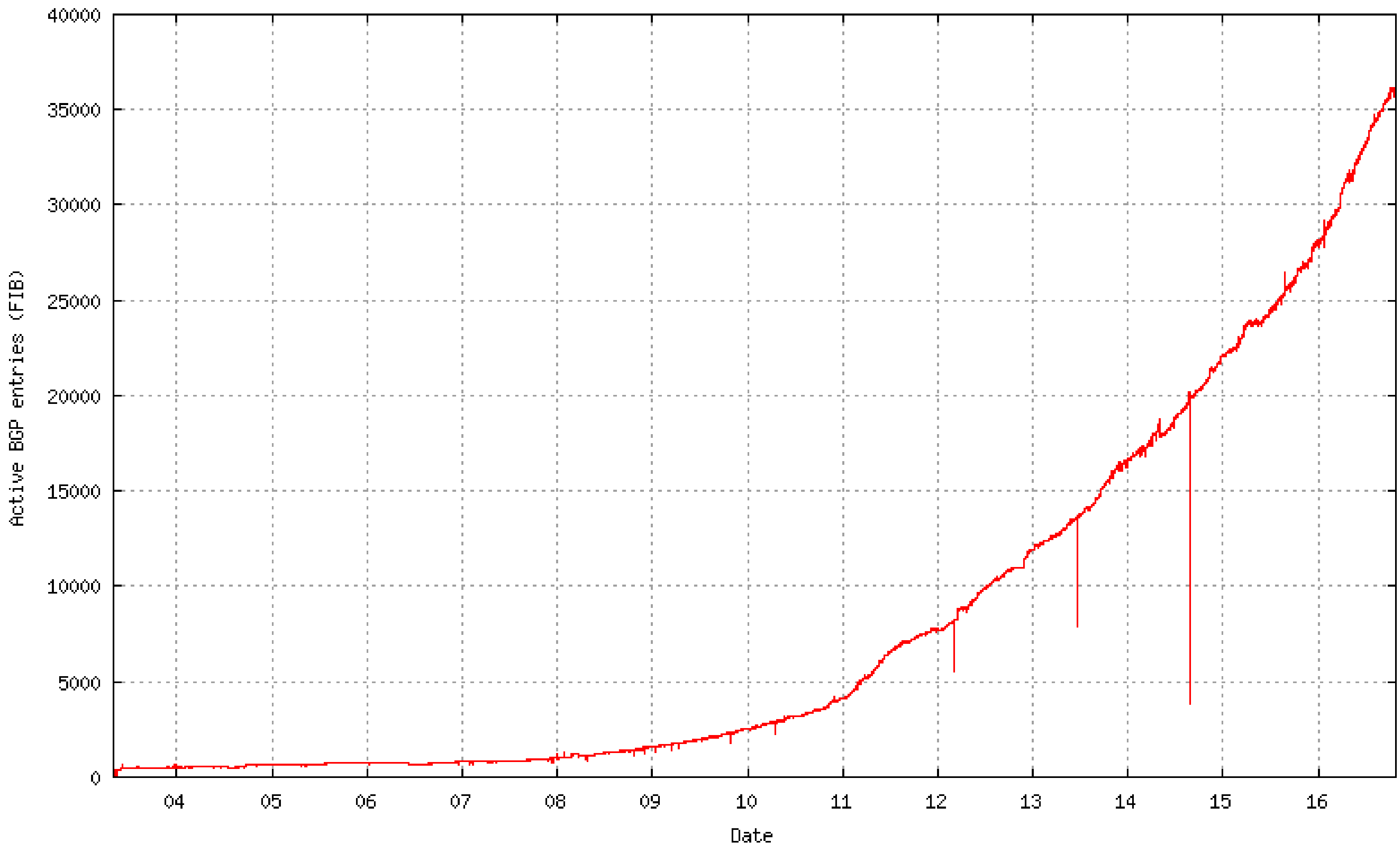
Source: http://www.potaroo.net/tools/ipv4/index.html
(retrieved on 11/10/2012)

IPv6 BGP advertisements (years ago)



Source: http://bgp.potaroo.net/v6/as6447/
(retrieved on 17/03/2010)

IPv6 BGP advertisements **today**



Source: http://bgp.potaroo.net/v6/as6447/
(retrieved on 25/10/2016)

NAT a solution ?

- **Benefits**
  - Reduces consumption of public IPv4 addresses
  - Hides internal IPv4 addresses inside residential and corporate networks

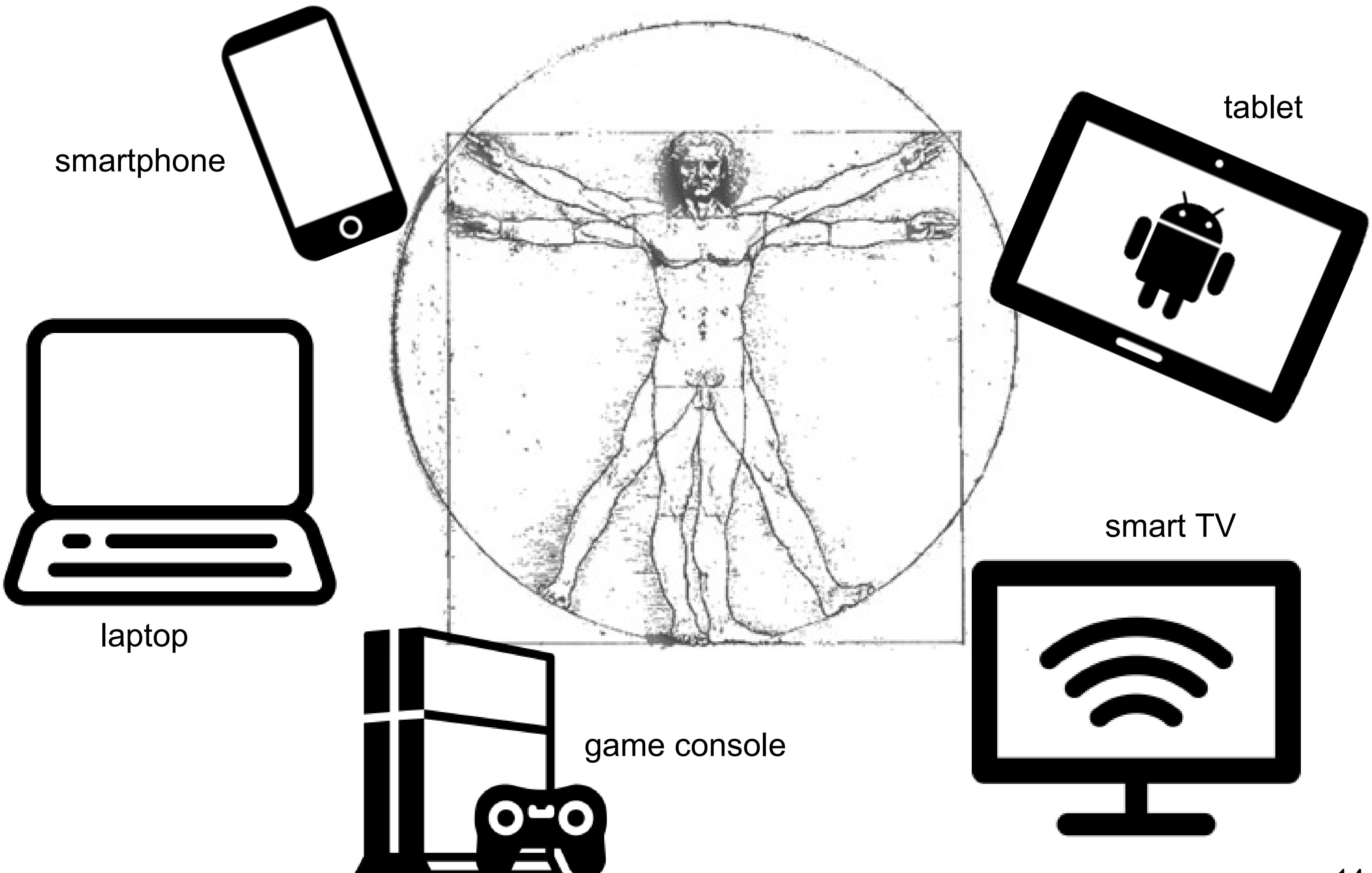
- **Drawbacks**
  - NAT traversal can be tricky
  - Goes against end-to-end principle
  - Gives a false feeling of security
  - Intermediate nodes may modify packet content
    - IP addresses
    - TCP/UDP port information
    - Some protocols encode IP addresses inside payload

  - Carrier-Grade NAT (CGN) : NAT444, LSN, ...

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
  - 4.2.1 Unicast Addresses
  - 4.2.2 Multicast Addresses
  - 4.2.3 Anycast Addresses
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

How many addresses do you need ?



IPv6 Addresses

- Each IPv6 address is encoded in 128 bits

IPv4

IPv6

  - about $3.4 \cdot 10^{38}$ possible addresses
    - 340,282,366,920,938,463,463,374,607,431,768,211,456
  - about $4.86 \cdot 10^{28}$ addresses per person on the earth
  - $6.67 \cdot 10^{23}$ addresses per square meter
  - Looks unlimited... today

- Why 128 bits ?
  - Some wanted variable size addresses
  - Some wanted 64 bits (~ to exp. CPU data path width)
  - Hardware implementers prefer fixed size addresses

IPv6 Addresses - Notation

- How can we write a 128 bits IPv6 address ?
- **Hexadecimal format**
  - `FEDC:BA98:7654:3210:FEDC:BA98:7654:3210`
  - `2001:0DB8:0000:0000:0202:B3FF:FE1E:8329`
    - leading zeros can be omitted →
      `2001:DB8:0:0:202:B3FF:FE1E:8329`
- **Compact hexadecimal format**
  - Consecutive zeros can be replaced by a double colon. The double colon can appear only once in an address !
  - `2001:DB8::202:B3FF:FE1E:8329`
  - `FF01:0:0:0:0:0:0:101` → `FF01::101`
  - `0:0:0:0:0:0:0:1` → `::1`

IPv6 Addressing Architecture

- Three types of IPv6 addresses

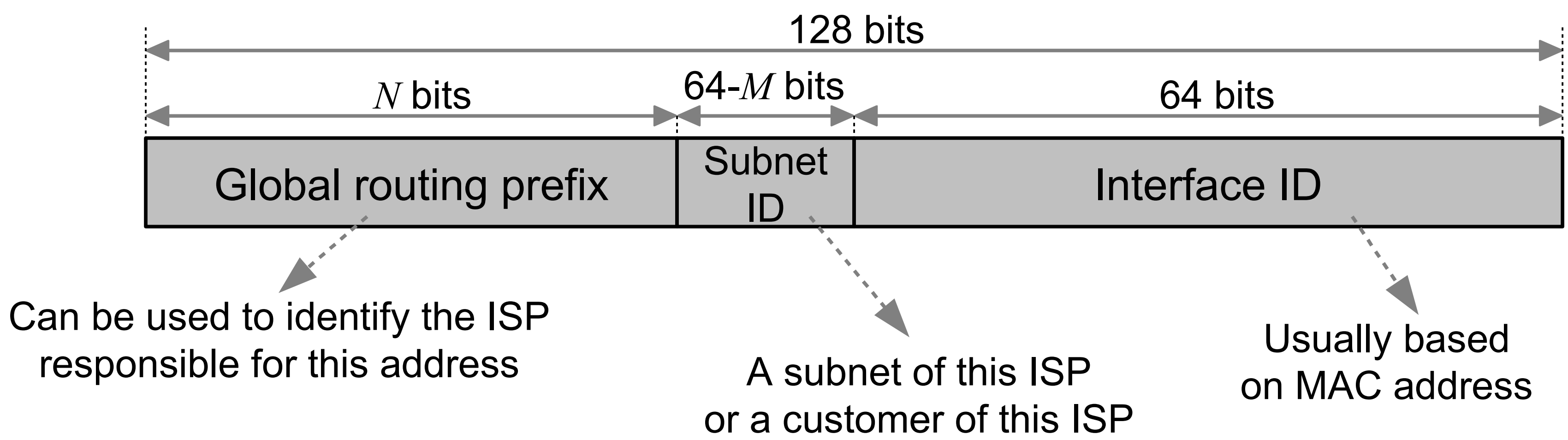
- **Unicast addresses**
  - Identifier for a single interface.
  - Deliver to the single interface identified by that address.
- **Anycast addresses**
  - Identifier for a set of interfaces (usually on multiple nodes).
  - Deliver to the single “nearest” interface identified by that address.
- **Multicast addresses**
  - Identifier for a set of interfaces (usually on multiple nodes).
  - Deliver to all interfaces identified by that address.

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
  - 4.2.1 Unicast Addresses
  - 4.2.2 Multicast Addresses
  - 4.2.3 Anycast Addresses
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

Unicast Addresses

- **Special addresses**
  - Unspecified address: `0:0:0:0:0:0:0:0` (`::`)
  - Loopback address: `0:0:0:0:0:0:0:1` (`::1`)
- **Global Unicast Addresses** (RFC 4291)
  - Reserved prefix: `2000::/3`
  - Addresses allocated hierarchically (see later)
  - Size N of prefix usually between 32 and 48

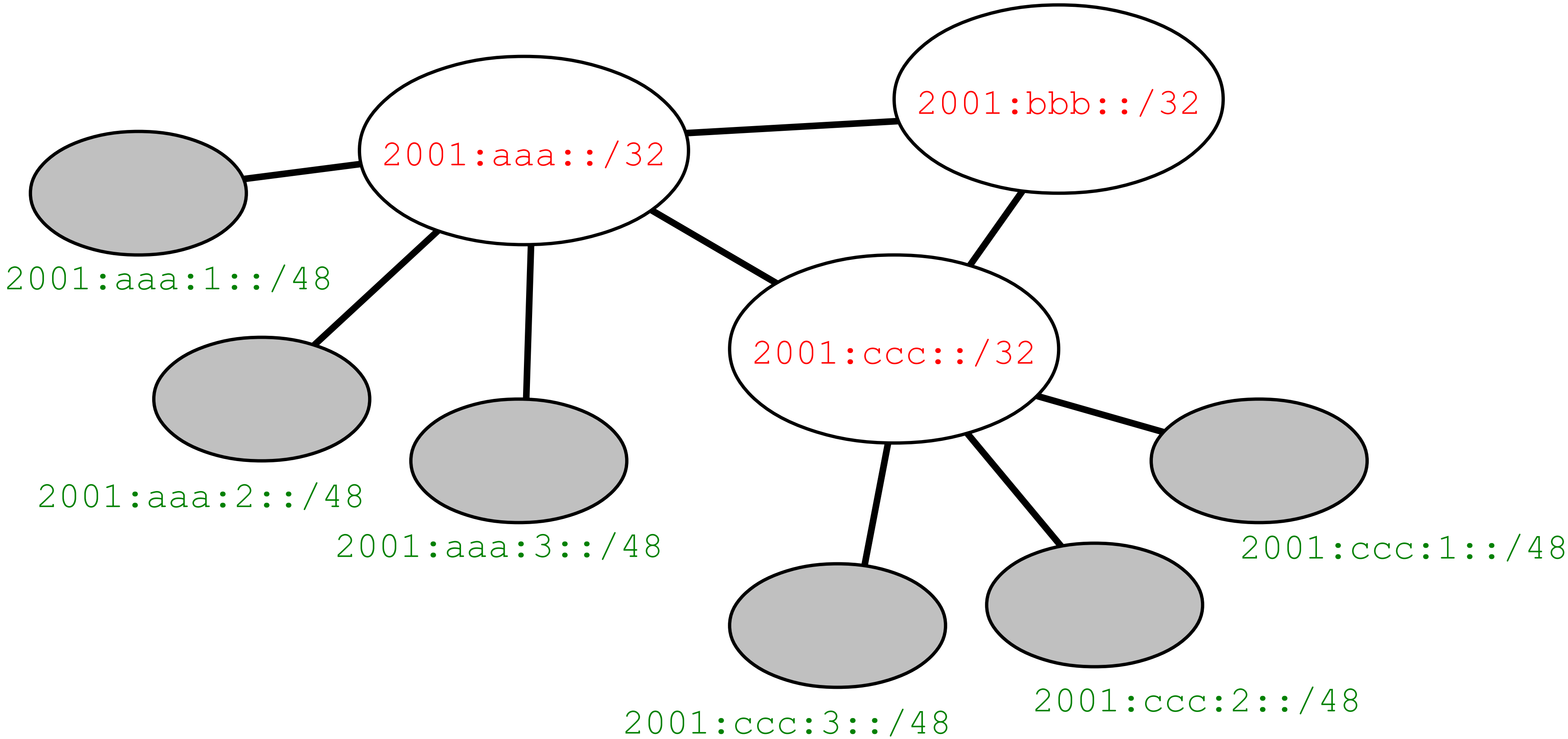


Unicast Addresses - Allocation

- IANA controls all IP addresses
  - delegates assignments of blocks to *Regional Internet Registries* (RIR): RIPE, ARIN, APNIC, AFRINIC, ...

- Allocation: 2 types of IPv6 address blocks
  - **Provider Independent (PI)** addresses
    - Usually allocated to ISPs or very large enterprises directly by RIRs
    - Default size is /32
  - **Provider Aggregatable (PA)** addresses
    - Smaller prefixes, assigned by ISPs from their PI block
    - Size
      - /48 in the general case, except for very large subscribers
      - /64 when a single subnet is required by design
      - /128 when it is absolutely known that one and only one device is connecting

Unicast Addresses - Allocation

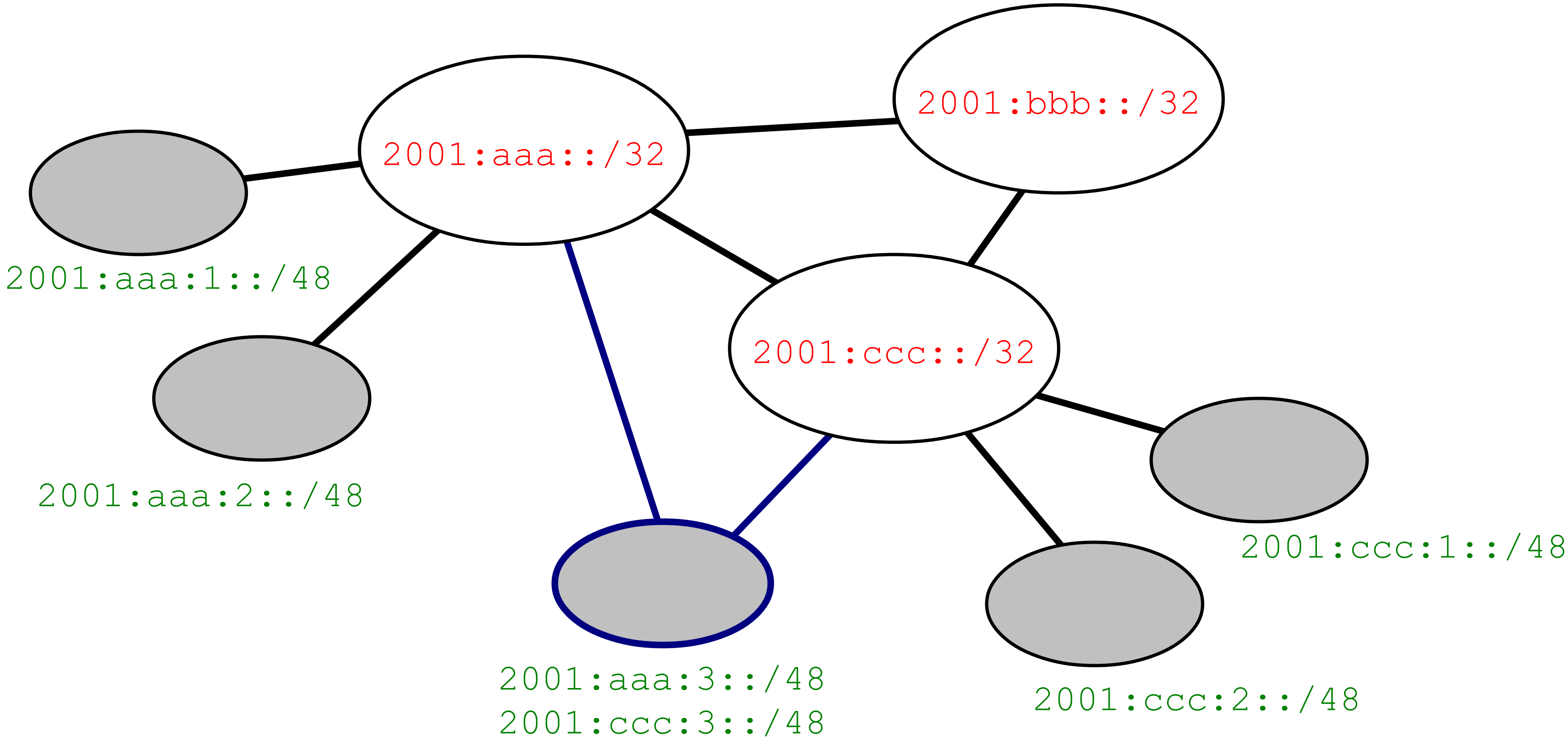


Provider Independent (PI)
Provider Aggregatable (PA)

Unicast Addresses - Allocation

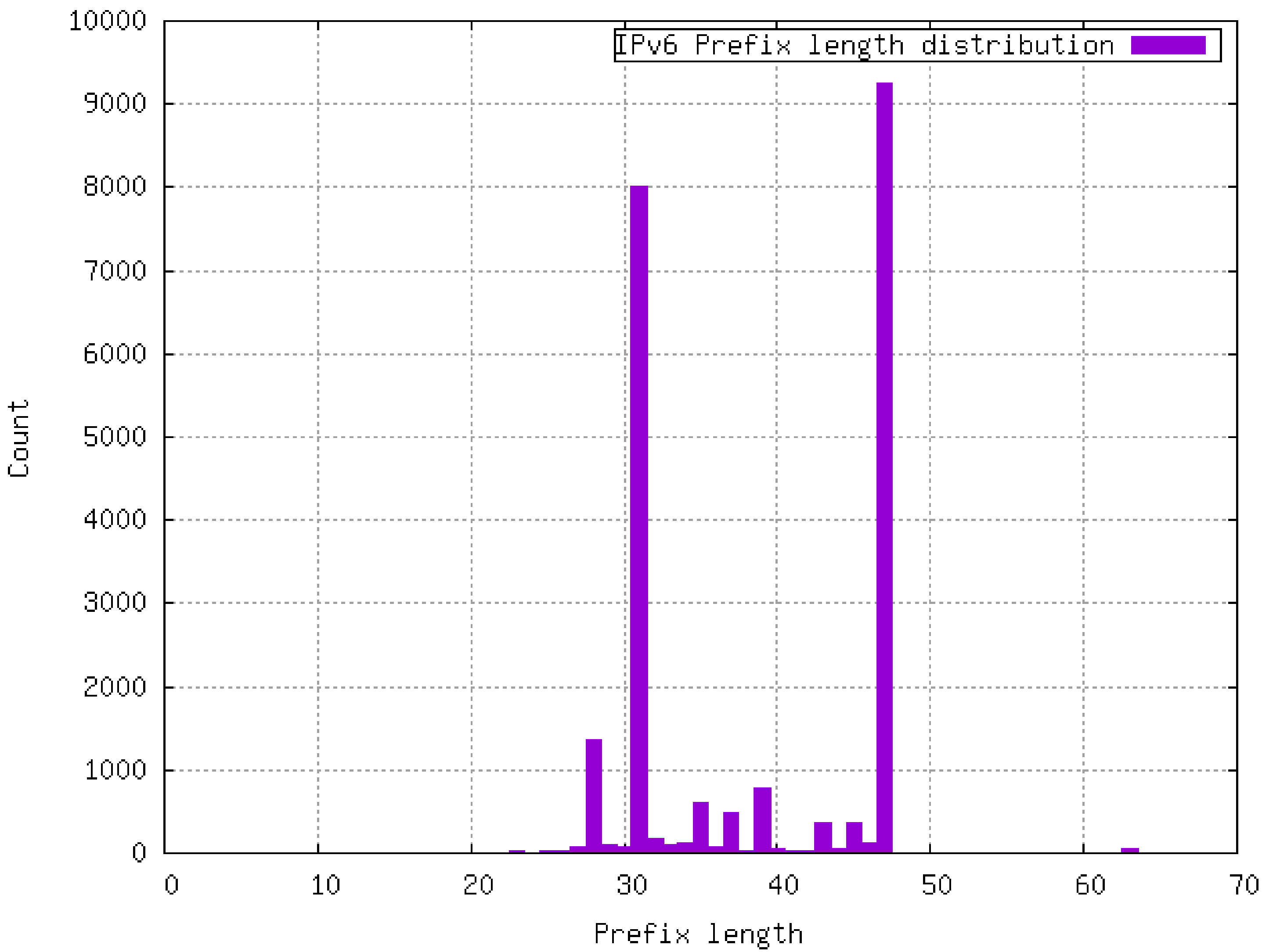
- **Consecutives of using PA addresses**
  - Better aggregation than with IPv4
    - Example: BELNET
      - with IPv4, announces several prefixes from its customers
      - with IPv6, announces a single prefix
    - Should lead to smaller global Internet routing tables
  - Provider change
    - if one network changes provider, it must re-number all its interfaces. Can be tricky and still an open research problem !
    - See for example `draft-ietf-6renum-enterprise-02.txt` (1/9/2012)
  - Multi-homed networks
    - one multi-homed network must receive a separate prefix from each of its providers and each interface will be assigned several addresses, one in each prefix.

Unicast Addresses - Allocation



dual-homed network
→ 2 prefixes

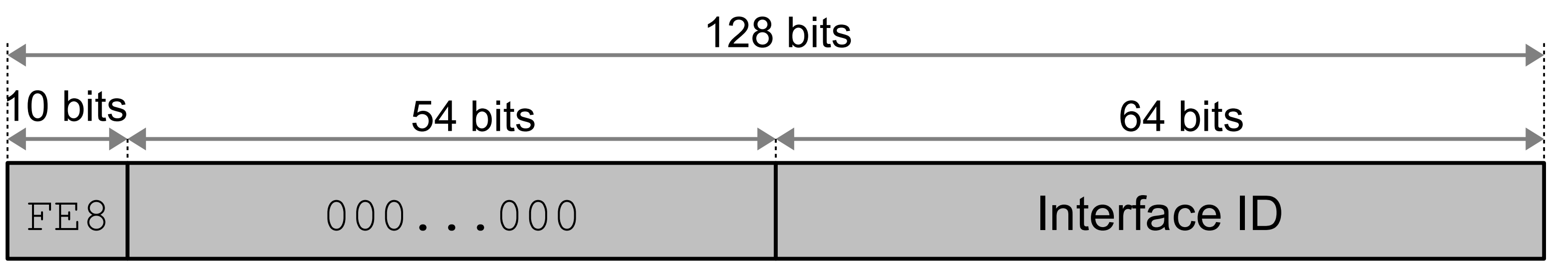
Unicast Addresses - Allocation



Source: based on data from
http://www.cidr-report.org (retrieved on 24/10/2016)

Unicast Addresses – Link-local

- **Link-local Addresses**
  - Reserved prefix `FE80::/10`
  - Similar to `169.254/16` addresses in IPv4 (RFC 3927)
  - Used by hosts and routers attached to the same link/LAN to exchange IPv6 packets when they don't have/need globally routable addresses
  - Each host/router must generate one link local address for each of its interfaces → Each IPv6 host will use several IPv6 addresses



(1111111010)

generated:
- deterministically (modified EUI-64)
- randomly
- cryptographically (CGA)

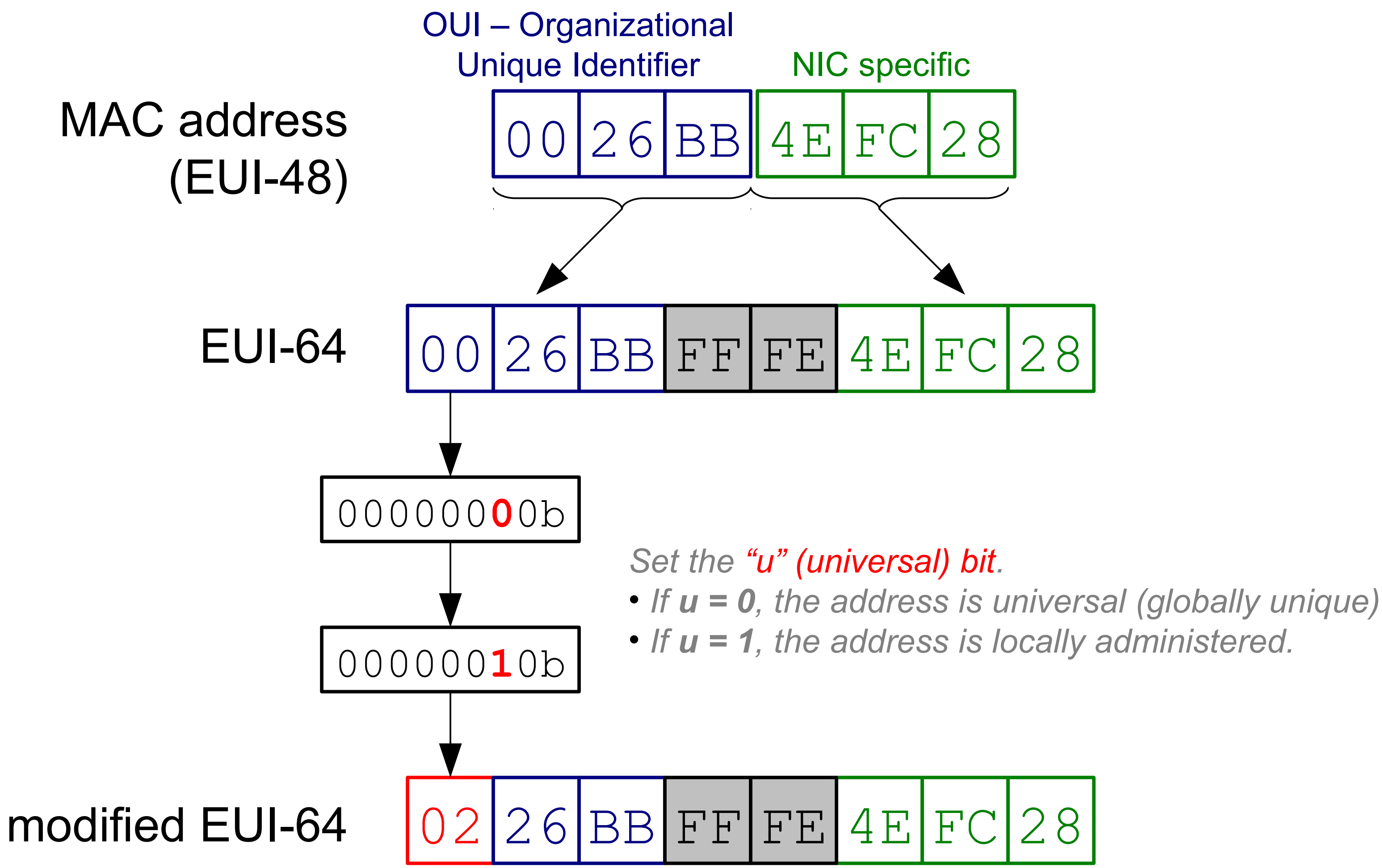
Unicast Addresses – Link-local

- **The modified EUI-64 format**
  - The *Extended Unique Identifier* (EUI) is an IEEE standard for link-layer addresses (e.g. Ethernet MAC addresses).
  - RFC2464 defines a scheme where the Interface ID of IPv6 addresses in the prefix range `001` to `111` (except for multicast) is derived from an EUI-64 address.
    - Other schemes exist, such as, for example CGA addresses (RFC3972) or IPv6 addresses in non-Ethernet networks such as IEEE 802.15.4 WPANs.
  - Example

```
bash-3.2$ ifconfig en0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet6 fe80::226:bbff:fe4e:fc28%en0 prefixlen 64 scopeid 0x4
        inet 192.168.2.2 netmask 0xffffff00 broadcast 192.168.2.255
        ether 00:26:bb:4e:fc:28
        media: autoselect (100baseTX <full-duplex,flow-control>)
        status: active
```

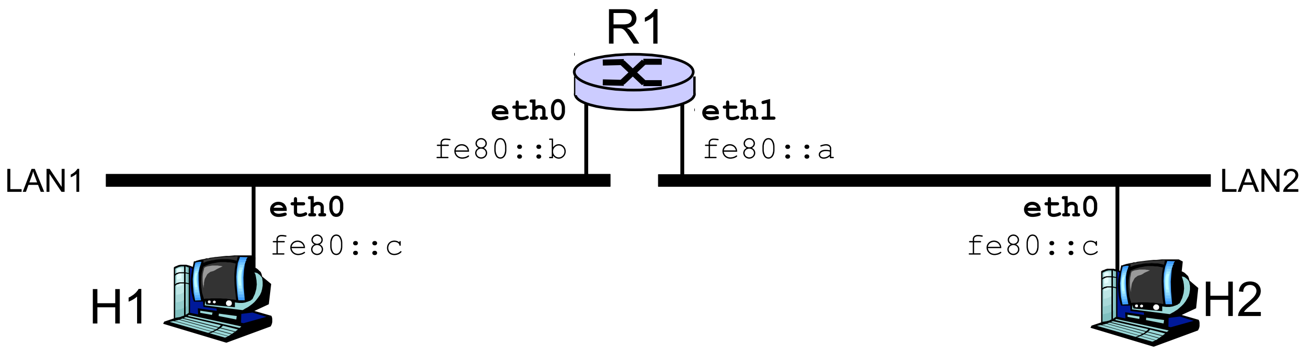
Unicast Addresses – Link-local

- **The modified EUI-64 format**



Unicast Addresses – Link local

- **Scope of addresses**
  - Link-local addresses have a *scope* limited to the link on which they are configured.



  - In the above example, the same link-local address is used by hosts H1 and H2 on different links (LANs)
  - From the perspective of R1 this can lead to ambiguities

```
# ping6 fe80::c
```

→ error : send left ? right ?

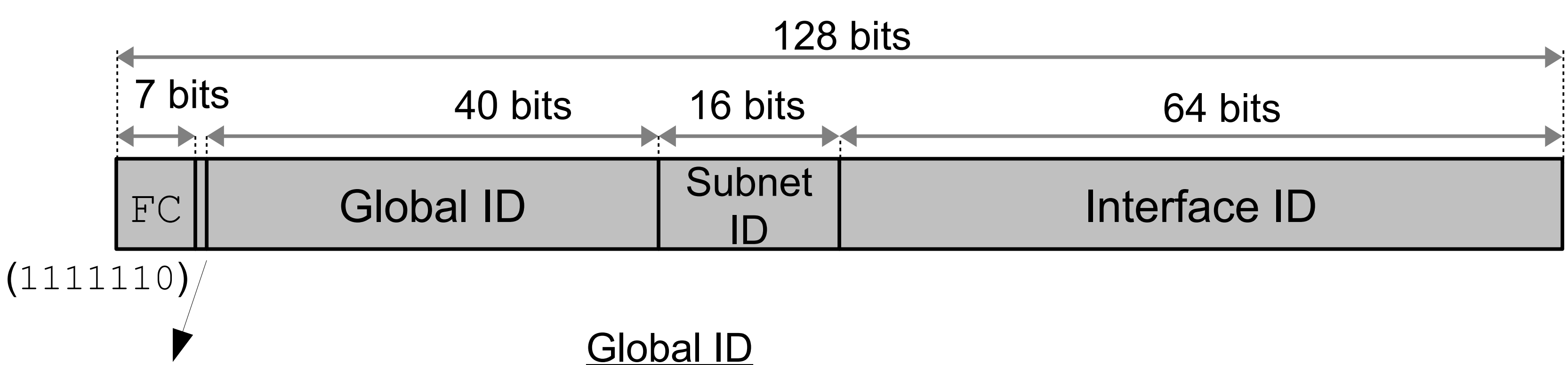
```
# ping6 fe80::c%eth0
# ping6 fe80::c%eth1
```

→ send to H1

→ send to H2

Unicast Addresses – Unique local

- **Unique Local Addresses (ULA)**
  - Defined in RFC 4193
  - Similar to IPv4 private addresses (RFC 1918)
  - Globally unique (with high probability), intended for local communications
  - Reserved prefix : `FC00::/7`



L bit
L = 1, locally assigned
L = 0, for future use

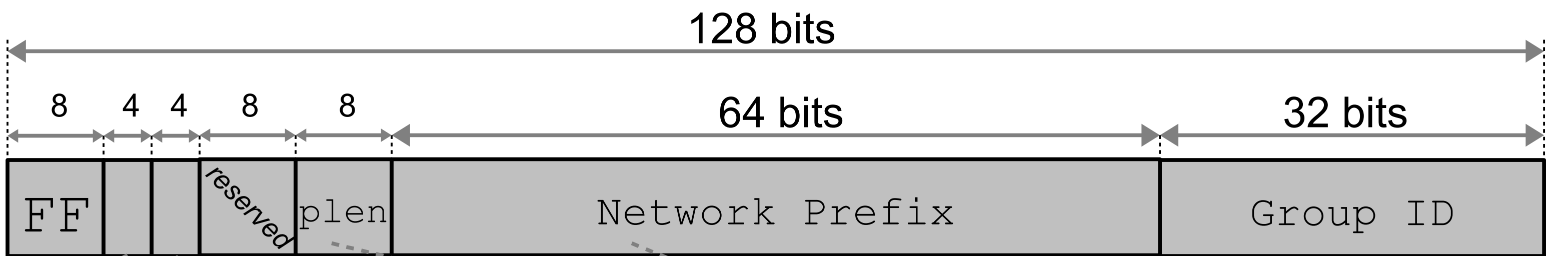
Global ID
randomly computed in order to build a globally unique prefix

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
  - 4.2.1 Unicast Addresses
  - 4.2.2 Multicast Addresses
  - 4.2.3 Anycast Addresses
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

Multicast Addresses

- **Multicast Addresses**
  - Reserved prefix `FF00::/12`
  - RFC 7371



ff1
(0RPT)
0: reserved (value=0)
R: 1=contains rendez-vous point
T: 0=permanent
1=temporary
P: 1=assigned based on prefix

scope
0, 3, F – reserved
1 – interface-local (a.k.a. node-local, used e.g. for testing)
2 – link-local
4 – admin-local
5 – site-local
6,7 – unassigned
8 – organization-local
9, A, B, D, D – unassigned
E – global

Note: the boundaries of scopes other than interface, link and global are to be configured by network administrators !

prefix information
used in combination with flag P = 1
plen = prefix length
(used for dynamic multicast address allocation)

ff1 : flag field 1
ff2 : flag field 2

Multicast Addresses

- **Well-known multicast groups**
  - Typically prefix `FF00::/12`
  - **Interface-local scope** (*loopback multicast*)
    - All-nodes address = `FF01::1`
    - All-routers address = `FF01::2`
  - **Link-local scope**
    - All-nodes address = `FF02::1`
    - All-routers address = `FF02::2`
    - Solicited-node address = `FF02::1:FF`*`xx`*`:`*`xxxx`*
      where *`xx:xxxx`* is formed by taking the less-significant 24 bits of an IPv6 address (more on this later)
    - All DHCP agents = `FF02::1:2`

Multicast Addresses

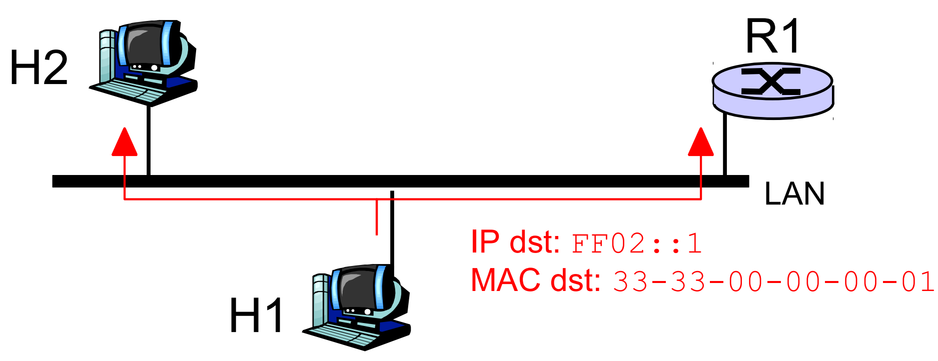
- **Multicast Ethernet addresses**
  - When an IP datagram must be carried in a link-layer frame (e.g. an Ethernet frame), multicast MAC addresses could be used (if supported).
  - Ethernet supports multicast addresses: the least significant bit of the first byte (high order) is set to 1.

00-26-BB-4E-FC-28	unicast Ethernet address
01-80-C2-00-00-00	multicast Ethernet address
FF-FF-FF-FF-FF-FF	broadcast Ethernet address

  - Any Ethernet controller that is registered in a multicast group should listen to frames sent to the corresponding multicast address in addition to its regular MAC address.
    - Modern Ethernet controller often provide support for multicast addresses.

Multicast Addresses & Ethernet

- **Sending IPv6 multicast packets**
  - The **33-33-** Ethernet prefix is reserved for IPv6 multicast.
  - To send an IPv6 multicast packet over Ethernet, a node simply takes the last 32 bits of the destination IPv6 address, prepends `33-33-` and uses that as the destination Ethernet address.
  - Example
    - An IPv6 packet addressed to `FF02::1` (all-nodes, link-local) would be sent to the Ethernet address **33-33-**`00-00-00-01`.



IPv6

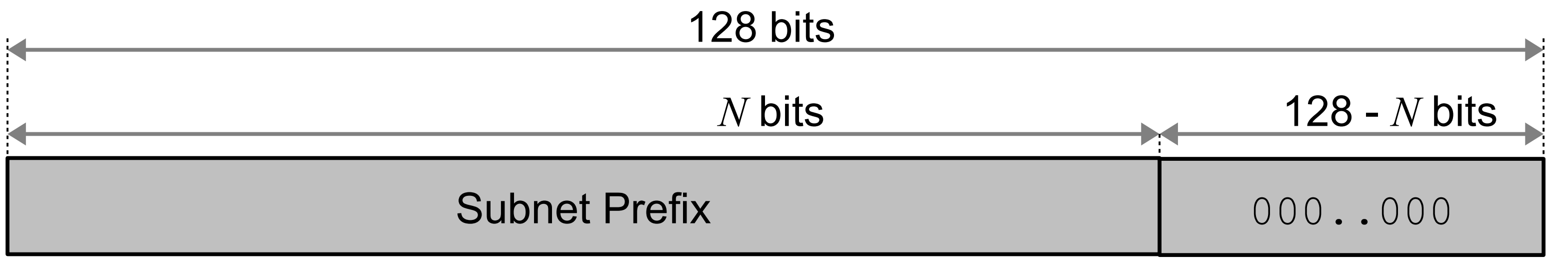
- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
  - 4.2.1 Unicast Addresses
  - 4.2.2 Multicast Addresses
  - 4.2.3 Anycast Addresses
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

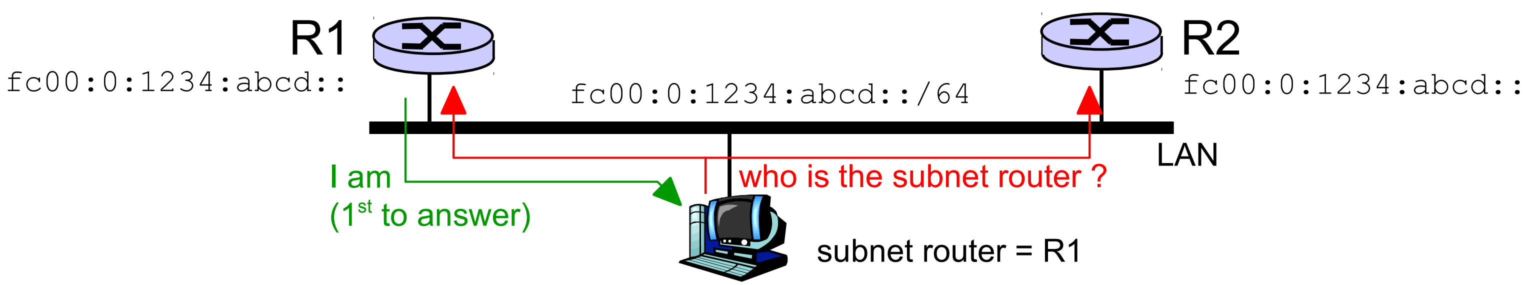
Anycast Addresses

- Definition
  - An IPv6 address that is assigned to more than one interface (typically belonging to different nodes
  - *A packet sent to an anycast address is routed to the “nearest” interface having that address, according to the routing protocols' measure of distance.*
  - Also called “*shared unicast address*”
- Usage
  - Multiple redundant servers using the same address
  - Example: DNS resolvers and DNS servers
- Representation
  - IPv6 anycast addresses are regular unicast addresses (syntactically undistinguishable from an unicast address)
  - Required subnet anycast address

Anycast Addresses

- **Subnet-Router Anycast Address**
  - Mandatory anycast address defined by RFC4291
  - *A packet sent to this address will be delivered to one router on that subnet. All routers on a subnet must support the subnet-router anycast address.*
  - Lowest address in subnet (interface part all `0`s)
  - Any subnet prefix size (not only 64)





IPv6 Addresses - Summary

	Prefix (binary)	Prefix (hex)	Fraction of space
Global unicast	010	2000::/3	1/8
Link-local unicast	1111111010	FE80::10	1/1024
Unique local unicast	1111110	FC00::/7	
Multicast	11111111	FF00::/8	1/256

IANA allocates out of this prefix

Required IPv6 addresses

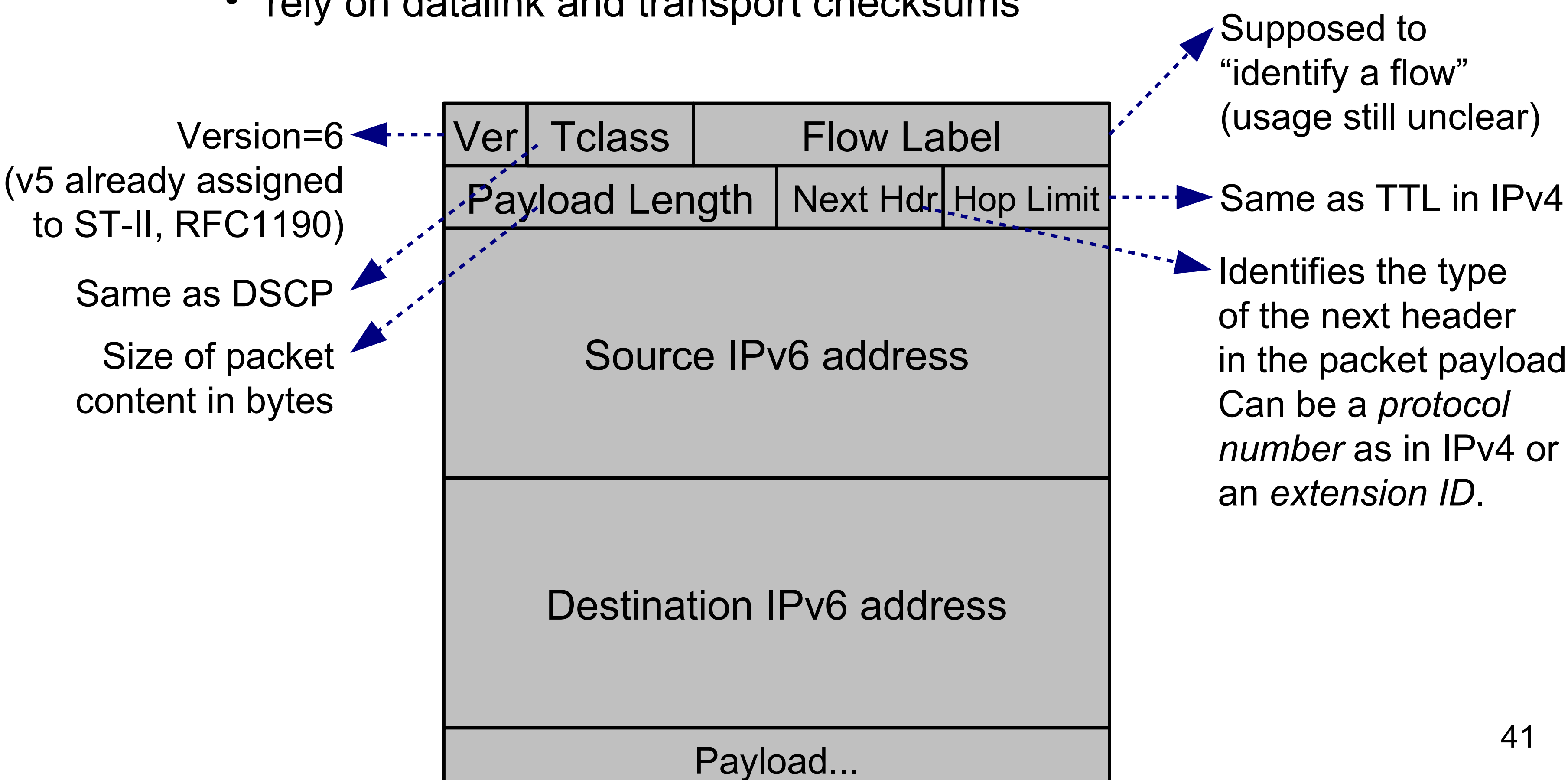
- **Summary**
  - Each **host** must consider the following addresses identify itself:
    - its link-local address for each interface i, `FE80::`$EUI_{64}(MAC_i)$
    - any assigned unicast and anycast addresses
    - the loopback address `::1`
    - the all-nodes multicast addresses `FF01::1, FF02::1`
    - solicited-node multicast address `FF02::1:FF`*`xx`*`:`*`xxxx`*
    - multicast addresses of groups to which the host belongs
  - Each **router**, in addition to the above addresses must support:
    - the all-routers multicast addresses `FF01::2, FF02::2`
    - multicast addresses of groups to which the router belongs
    - anycast addresses with which the router has been configured
    - subnet-router anycast addresses

IPv6

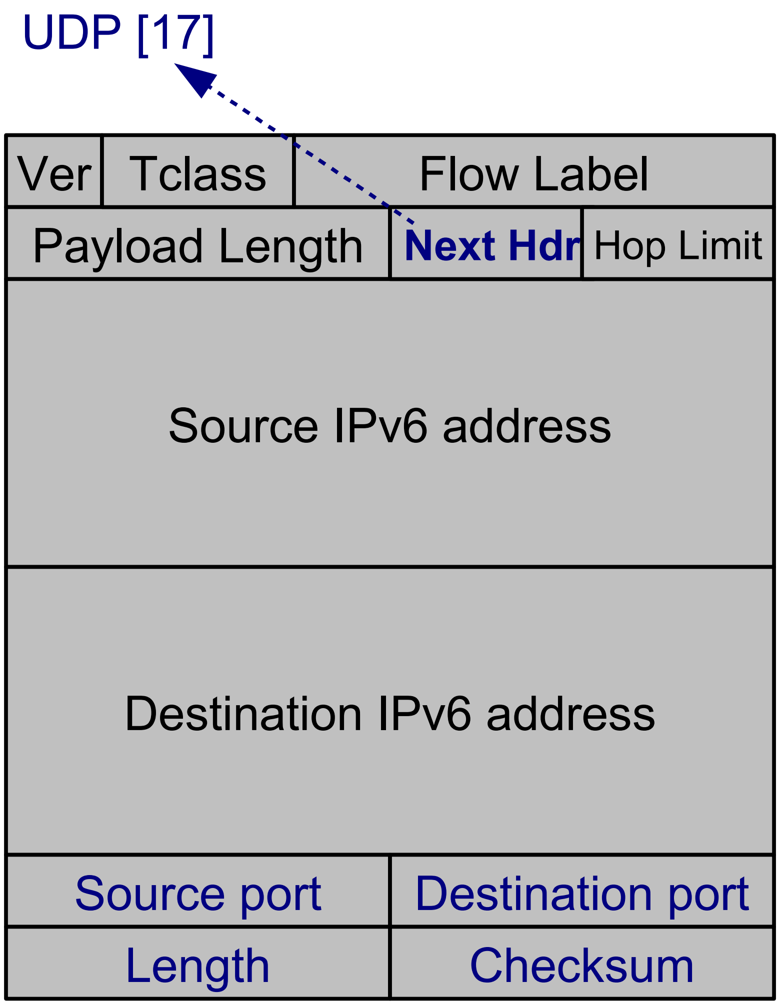
- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
  - 4.3.1 Header Format
  - 4.3.2 Extension Headers
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

The IPv6 Packet Format

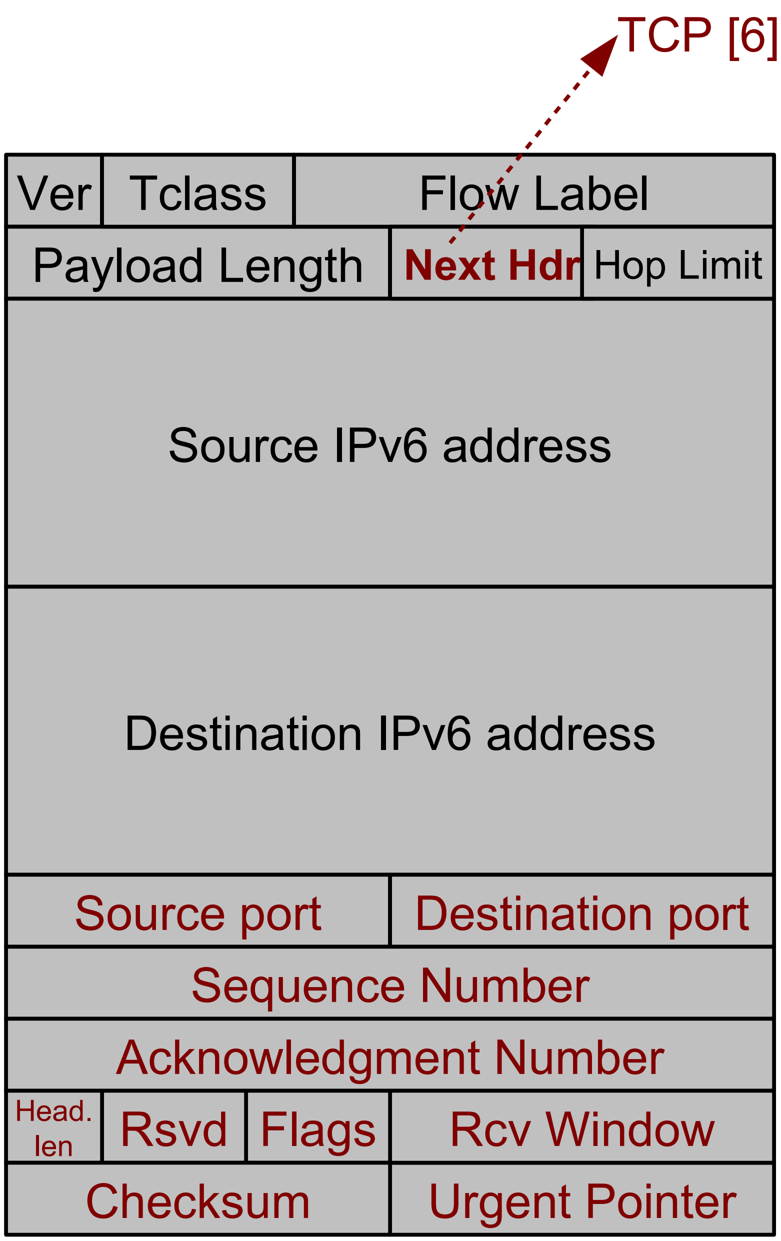
- Simplified packet format
  - Fields aligned on 32 bits boundaries to ease implementation
  - No checksum in IPv6 header
    - rely on datalink and transport checksums



Sample IPv6 Packets



UDP



TCP

IPv6

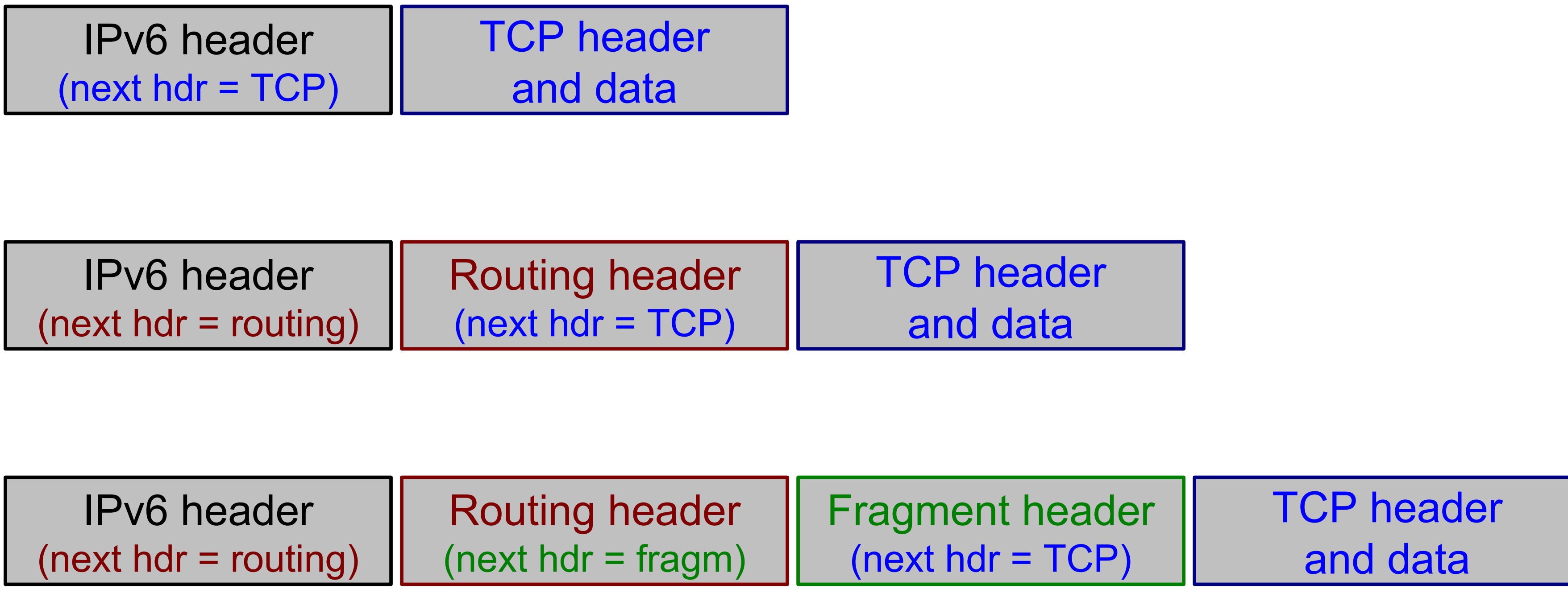
- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
  - 4.3.1 Header Format
  - 4.3.2 Extension Headers
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

IPv6 Extension Headers

- Several types of extension headers (RFC2460)
  - **Hop-by-hop Options** [ID=0]
    - contains information to be processed by each hop
  - **Routing header** (type 0 and type 2) [ID=43]
    - contains information affecting intermediate routers
  - **Fragmentation header** [ID=44]
    - used for fragmentation and reassembly
  - **Destination Options header** [ID=60]
    - contains options only relevant to destination
  - **Authentication header** [ID=51]
    - for IPSec
  - **Encrypted Security Payload** (ESP) header [ID=50]
    - for IPSec
- Each header must be encoded as a multiple of 64 bits (8 bytes)

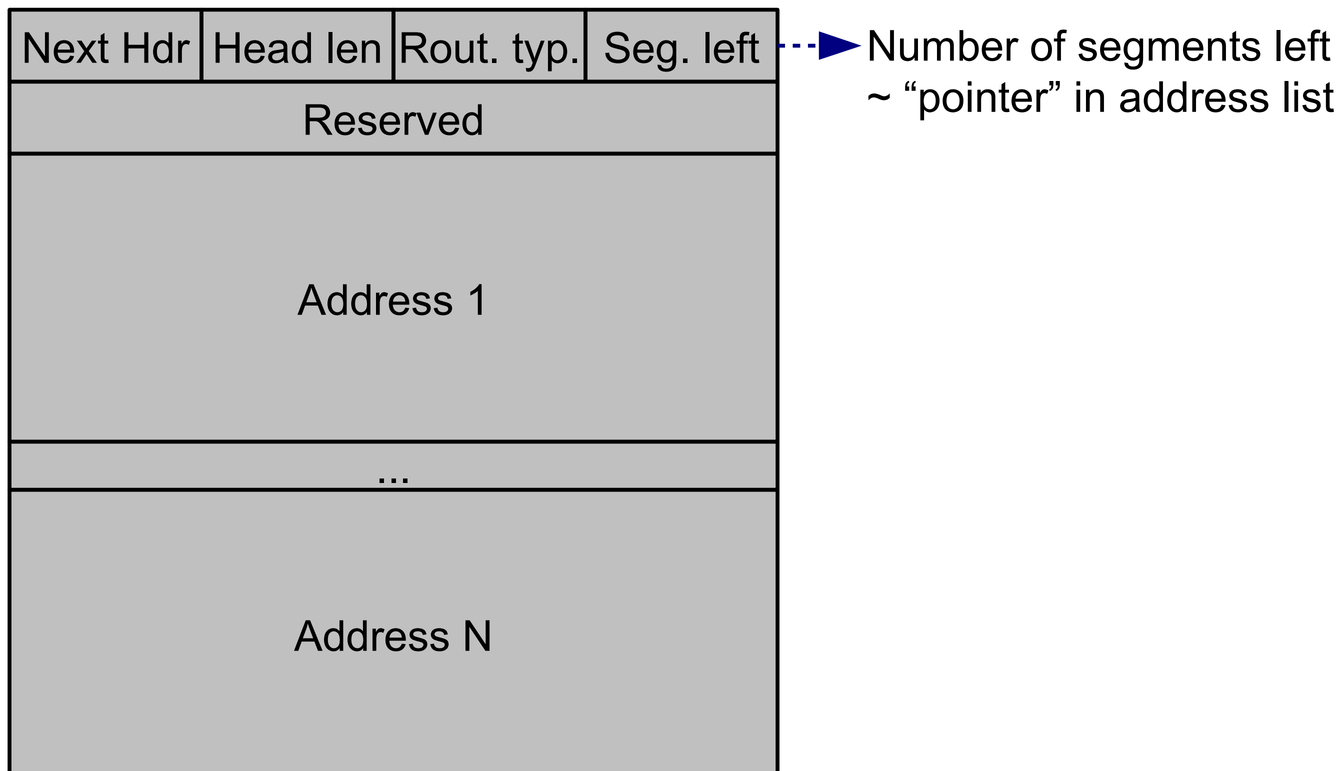
IPv6 Extension Headers

- Extension headers can be chained together

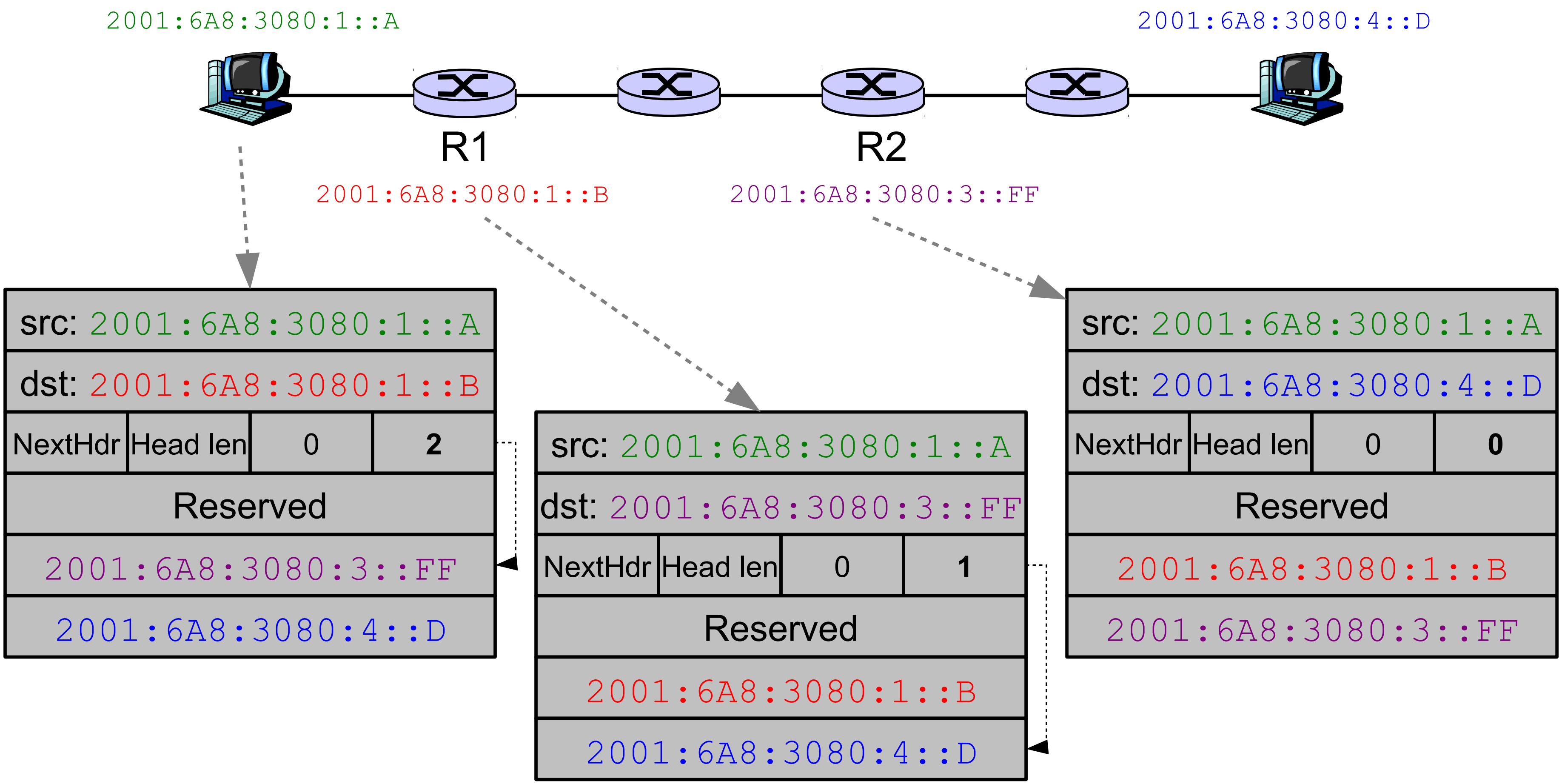


Type 0 Routing header

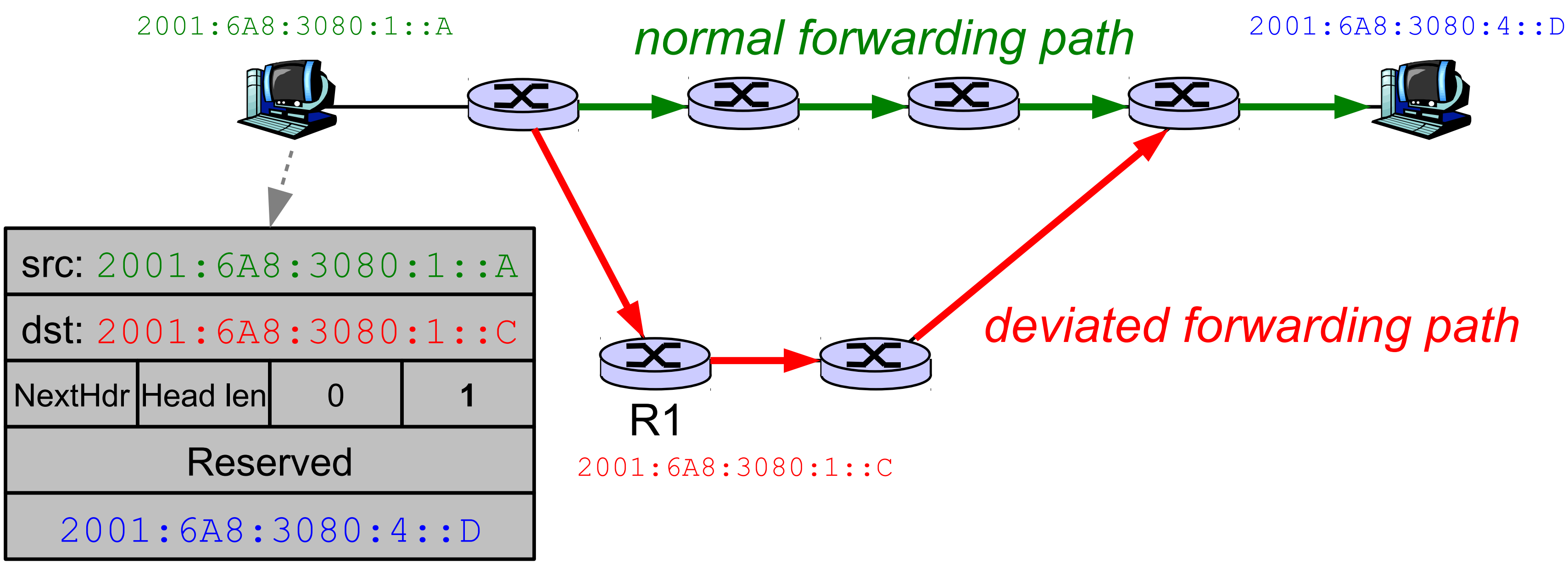
- Defines “*a mean for a source to list one or more intermediate nodes to be visited on the way to a packet's destination*”
  - similar to *loose source routing* in IPv4



Type 0 Routing header

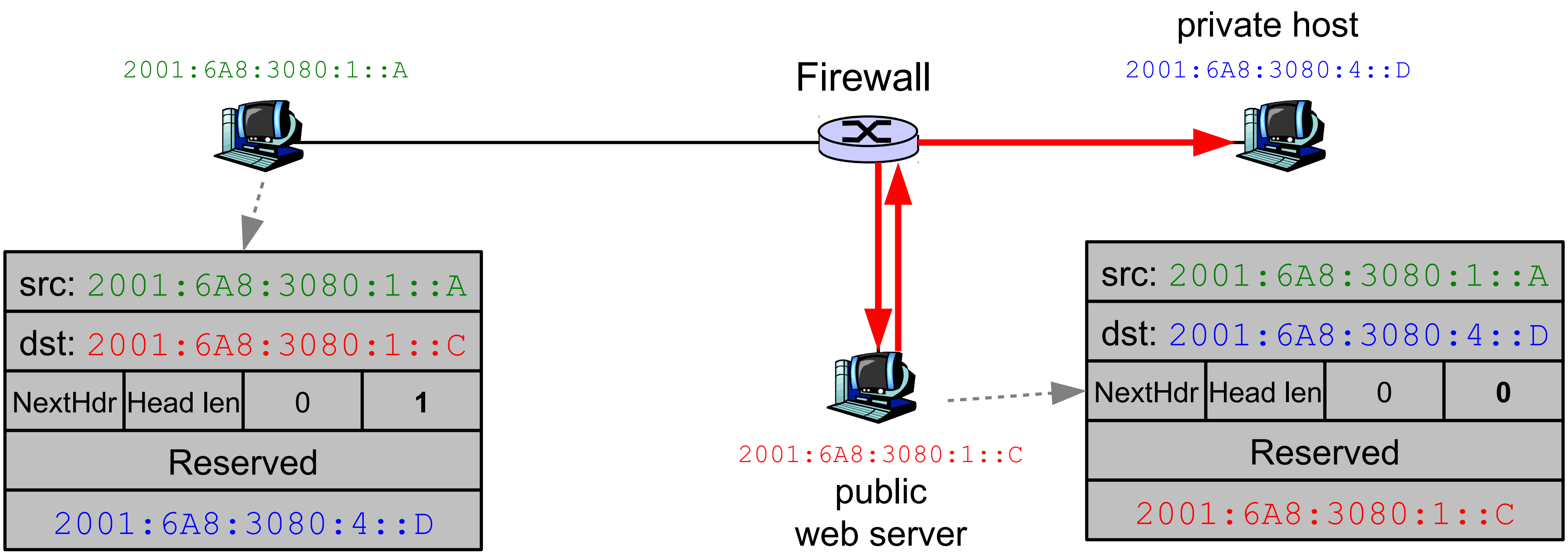


Type 0 Routing header



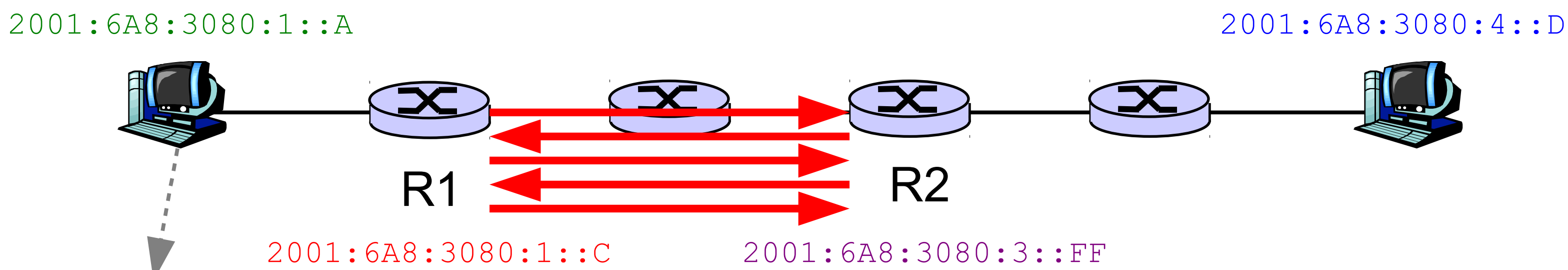
- The IPv6 specification is unclear: "*IPv6 nodes must accept and attempt to process extension headers in any order and occurring any number of times in the same packet, . . .*"
- what are IPv6 nodes ? routers ? end-hosts ?

Problems with RH0



- The IPv6 specification is unclear: "*IPv6 nodes must accept and attempt to process extension headers in any order and occurring any number of times in the same packet, . . .*"
- In practice, a lot of end-host OS support it.

Amplification Attack using RH0



src: 2001:6A8:3080:1::A			
dst: 2001:6A8:3080:1::C			
NextHdr	Head len	0	**73**
Reserved			
2001:6A8:3080:3::FF			
2001:6A8:3080:1::C			
2001:6A8:3080:3::FF			
2001:6A8:3080:1::C			
2001:6A8:3080:3::FF			
2001:6A8:3080:1::C			
2001:6A8:3080:3::FF			

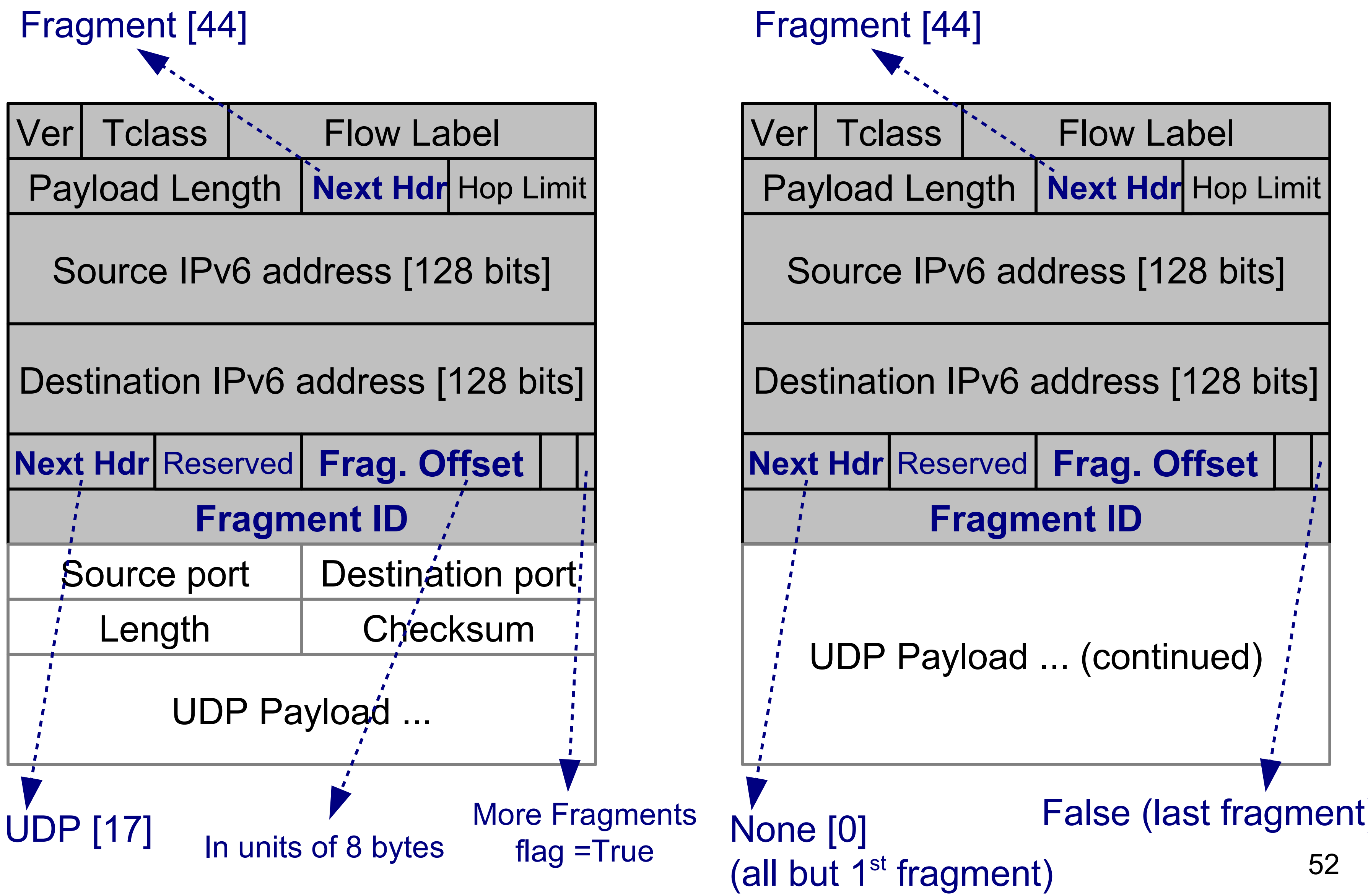
- A single packet can generate tens of packets between two routers/hosts
  - every IPv6 implem. must support at least 1280 bytes datagrams
  - (1280-40-8)/16=~77
  - 10Mbit/s → 770 Mbit/s

Packet Fragmentation

- **IPv4 supported packet fragmentation on routers**
  - All hosts must be able to handle packets that are at least 576 bytes (minimum MTU for IPv4)
  - Experience has shown that fragmentation is costly for routers and difficult to implement in hardware
  - PathMTU discovery is now widely implemented

- **IPv6**
  - Minimum MTU of 1280 bytes required
    - Otherwise, link-specific fragmentation and reassembly must be provided (IPv6 should not be aware of this)
  - Routers do not perform fragmentation
    - Only end hosts perform fragmentation and reassembly by using the fragment extension header
    - But PathMTU discovery should avoid fragmentation most of the time

Packet Fragmentation



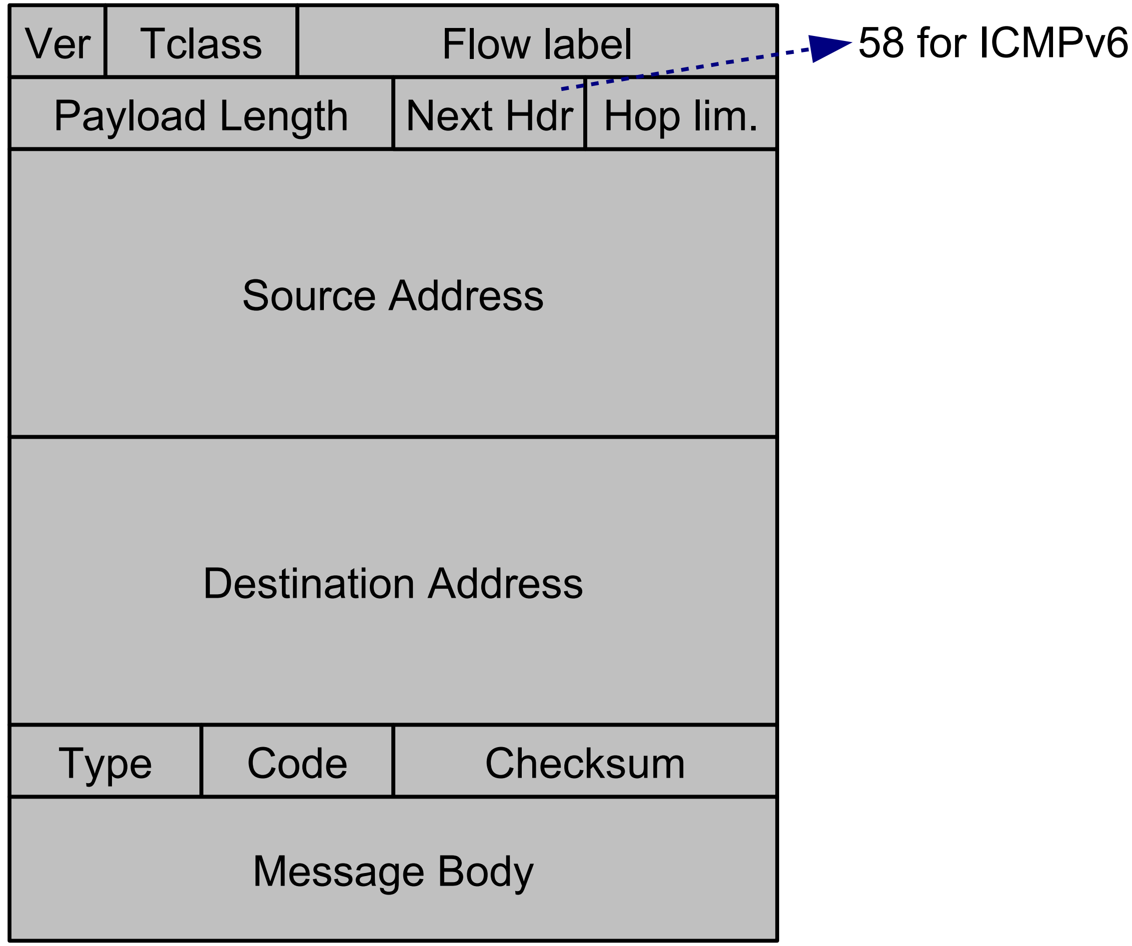
IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
- 4.4 ICMPv6
  - 4.4.1 Introduction and Packet Format
  - 4.4.2 Auto-configuration
  - 4.4.3 Security
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

ICMPv6

- Provides the same functions as ICMPv4, IGMP and ARP
- Types of ICMPv6 messages
  - Error: Destination unreachable [type=1]
  - Error: Packet too big [type=2]
    - used for PathMTU Discovery
  - Error: Time Exceeded (Hop limit exhausted) [type=3]
    - traceroute v6
  - Info: Echo request and echo reply [types=128,129]
    - ping v6
  - Info: Multicast group membership [types=130..132]
    - equivalent to IPv4's IGMP
  - Auto-configuration: Router advertisements, Neighbor discovery

ICMPv6 Packet Format

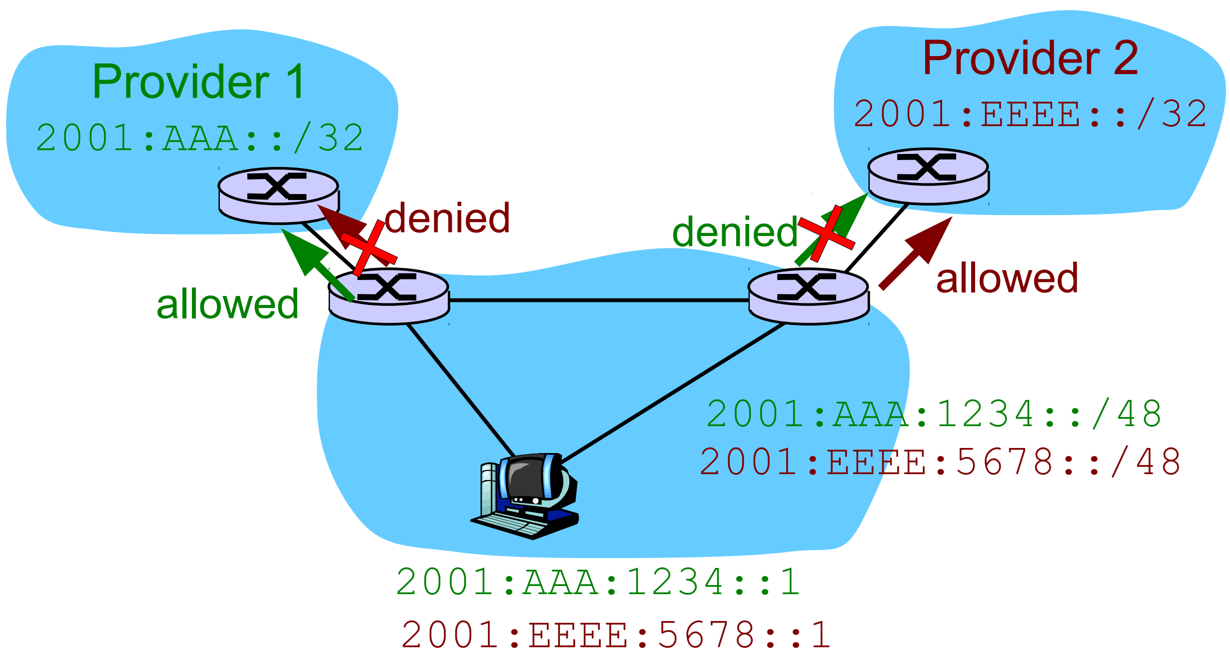


ICMPv6 Destination unreachable

- Destination Unreachable [type=1]
  - no route to destination [code=0]
    - no route for destination in forwarding table
  - communication with destination administratively prohibited [code=1]
    - typically sent by a firewall upon a “reject” action
  - beyond scope of source address [code=2]
  - address unreachable [code=3]
    - typically IPv6 address unknown on destination LAN
  - port unreachable [code=4]
    - transport-layer port not listening
  - source address failed ingress/egress policy [code=5]
    - see next slide :-)
  - reject route to destination [code=6]

Ingress and Egress Policies

- For security reasons, a provider should only accept packets from sources belonging to allocated prefixes.
  - Prevents IP spoofing
  - Typically implemented using *Reverse Path Forwarding* (RPF) checks in IPv4.



IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
- 4.4 ICMPv6
  - 4.4.1 Introduction and Packet Format
  - 4.4.2 Auto-configuration
  - 4.4.3 Security
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

Auto-configuration

- **Introduction**
  - IPv6 provides support for *StateLess Address Auto-Configuration* (SLAAC)
  - Based on ICMPv6 messages
  - Components of SLAAC
    - Neighbor Discovery (ND): similar to ARP in IPv4
    - Duplicate Address Detection (DAD)
    - Router Discovery
    - Link parameters and prefixes discovery

ICMPv6 Neighbor Discovery

- Replacement for IPv4's ARP
- **Neighbor Solicitation** [type=135]
  - usually sent to solicited-node multicast address `FF02::1:FFxx:xxxx`

Type=135	Code=0	Checksum
Reserved		
Target IPv6 Address		

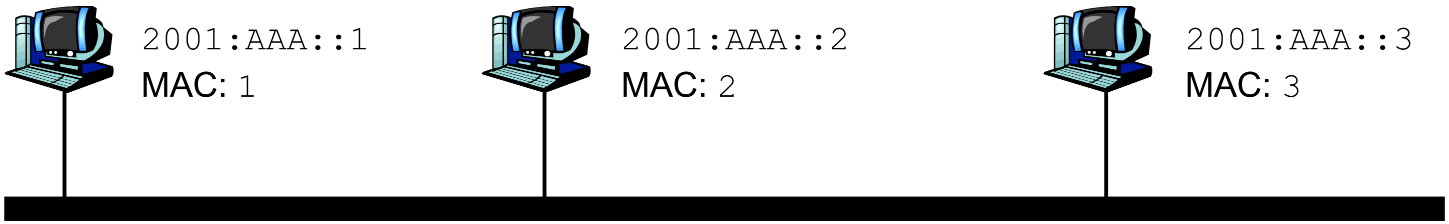
- **Neighbor Advertisement** [type=136]
  - sent to requesting node (if source address of solicitation is not the unspecified address `::`)
  - sent to all-nodes multicast address (if src of neighbor solicitation was `::`)

Type=136	Code=0	Checksum
Flags	Reserved	
Target IPv6 Address		
Target Link Layer Address		

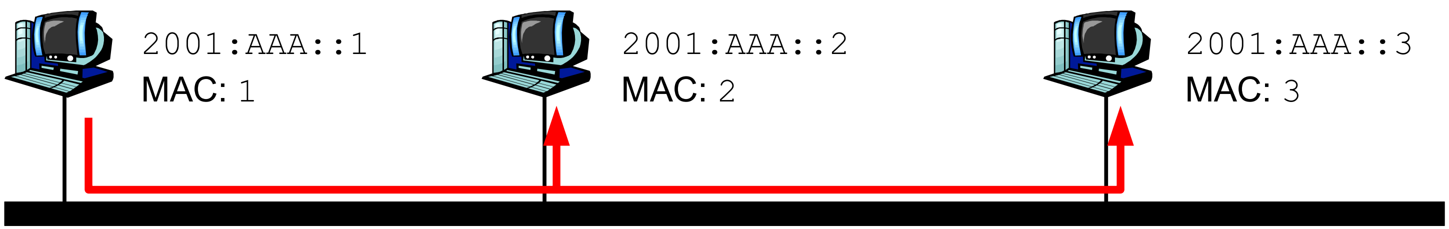
R: Router
S: Solicited
O: Override

IPv6 over Ethernet

①

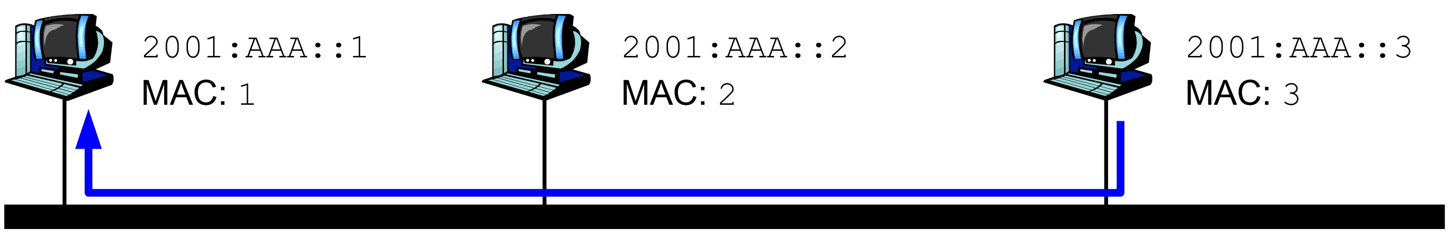


②



Neighbor Solicitation: `2001:AAA::3` sent to multicast address `FF02::1:FF00:3`

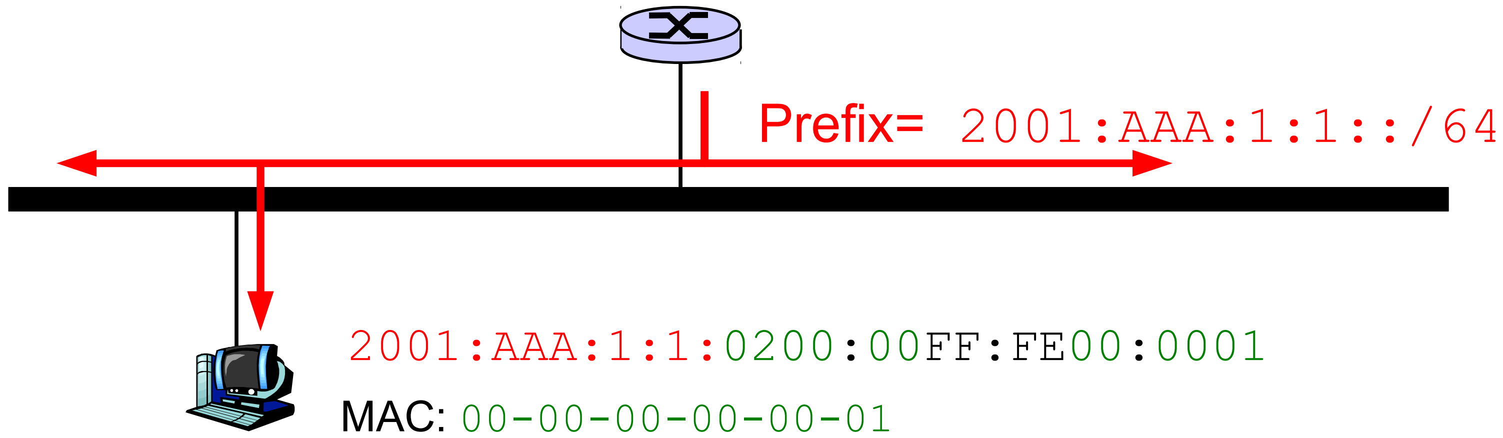
③



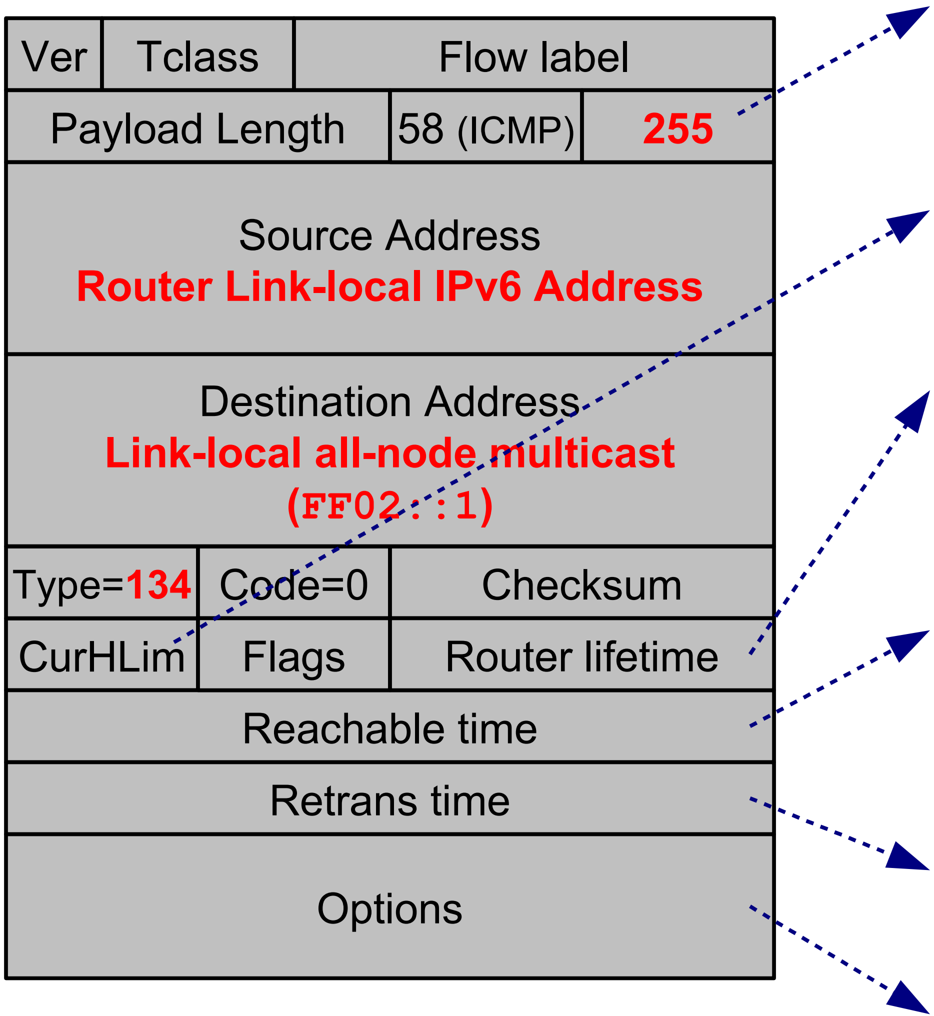
Neighbor Advertisement: (`2001:AAA::3`, `3`) sent to `2001:AAA::1`

IPv6 Auto-configuration - Overview

- How can a node obtain its IPv6 address ?
  - Manual configuration
  - From a server by using DHCPv6 as in IPv4
- Automatically
  - Router advertises prefix on LAN by sending ICMPv6 messages to “all IPv6 hosts” multicast address (`FF02::1`)
  - Hosts build their address by concatenating the prefix with their MAC address converted in 64 bits format (EUI-64)



Router Advertisements



Maximum hop limit to avoid spoofed packets from outside of LAN

Value of hop limit to be used by hosts when they send IPv6 packets

The lifetime associated with the default router in seconds. If the router is not a default router, lifetime = 0.

The time (in ms) that a node can assume a neighbor is reachable after having received a reachability confirmation (see *Neighbor Unreachability Detection* algorithm, NUD).

The time (in ms) between retransmitted *Neighbor Solicitation* messages.

Prefix information
MTU to be used on the LAN

Router Advertisement Options

- General Option format

Type	Length	Options
Options (continued)		

Length in units of 8 bytes

- MTU Option

5	1	Reserved
MTU		

- Prefix option

3	4	PreLen			Res.
Valid lifetime					
Preferred lifetime					
Reserved					
IPv6 prefix (128 bits)					

Number of bits in IPv6 prefix that identify subnet

The validity period of the prefix (in seconds)

The duration (in seconds) that addresses generated from the prefix via stateless address autoconfiguration remain preferred

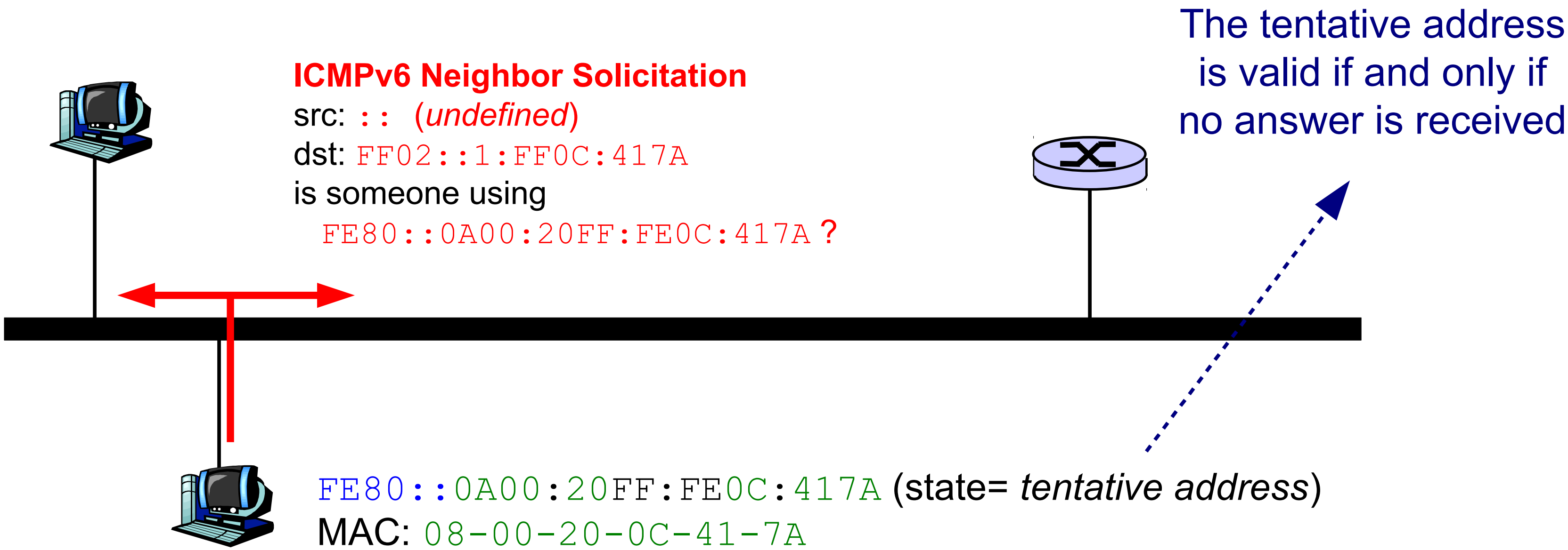
IPv6 Stateless Auto-configuration (1)

- What happens when an end-system boots ?
  - It knows nothing about its current network
    - but needs an IPv6 address to send ICMPv6 messages

  - Use Link-local IPv6 address (FE80::/64)
    - Each host, when it boots, has a link-local IPv6 address
    - But another node might have chosen the same address !
      - Ethernet cards with same MAC address or manually configured

IPv6 Stateless Auto-configuration (2)

- ***Duplicate Address Detection*** **(DAD)**
  - Send a Neighbor solicitation message to the tentative address's solicited-node multicast address.



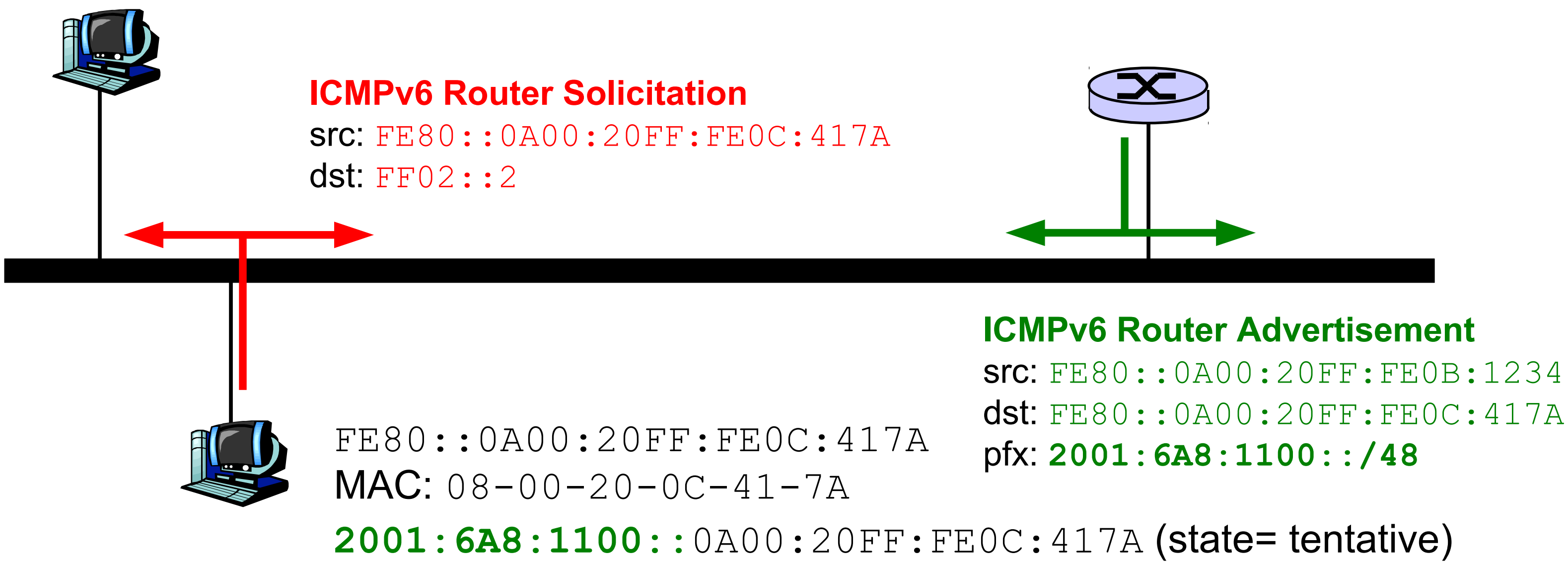
  - Before sending the Neighbor Solicitation, the node must register on the <u>all-node</u> *multicast* address (`FF02::1`) and the <u>solicited-node</u> *multicast* address (`FF02::1:FF`*`xx`*`:`*`xxxx`*) for the tentative address

IPv6 Stateless Auto-configuration (3)

- **DAD Algorithm**
  1. Node X is going to assign address A on interface I
  2. Interface I first joins the following multicast groups:
     - `FF02::1` (all-nodes link-local)
     - `FF02::1:FF`*xx*:*xxxx* (solicited-node)
  3. If X receives a *Neighbor Solicitation* for address A (on the solicited node address), then another node is performing DAD for address A. STOP
  4. X sends a *Neighbor Solicitation* for address A
     - to the corresponding *solicited-node* address and with the *undefined* source address
  5. If X receives a *Neighbor Advertisement* for A (on the all-nodes address), then another node has selected address A. STOP
  6. Address A is now in use.

IPv6 Stateless Auto-configuration (4)

- How to obtain the IPv6 prefix of the subnet ?
  - Wait for Router Advertisements (e.g. 30 seconds)
  - Solicit Router Advertisement



  - Need to perform DAD for newly created address(es)

Privacy Issues with IPv6 Auto-configuration

- **Issue**
  - Auto-configured IPv6 addresses contain the MAC address of the hosts
    - MAC addresses are “fix and unique”
    - A laptop/user could be identified by tracking the lower 64 bits of its IPv6 addresses

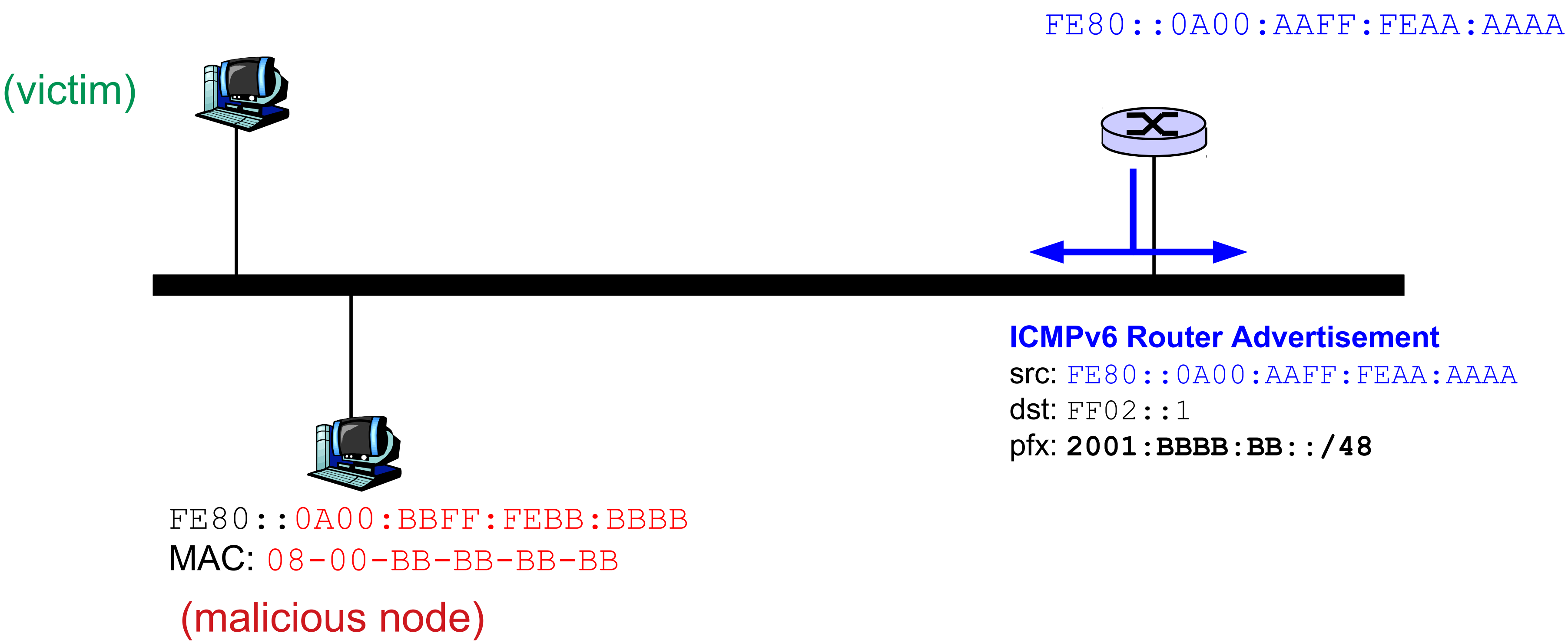
- **How to maintain privacy with IPv6 ?**
  - Use DHCPv6 and configure server to never reallocate the same IPv6 address
  - Allow hosts to use random host IDs in lower 64 bits of their IPv6 addresses
    - algorithms have been implemented to generate such random hosts IDs on nodes with and without stable storage

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
- 4.4 ICMPv6
  - 4.4.1 Introduction and Packet Format
  - 4.4.2 Auto-configuration
  - 4.4.3 Security
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

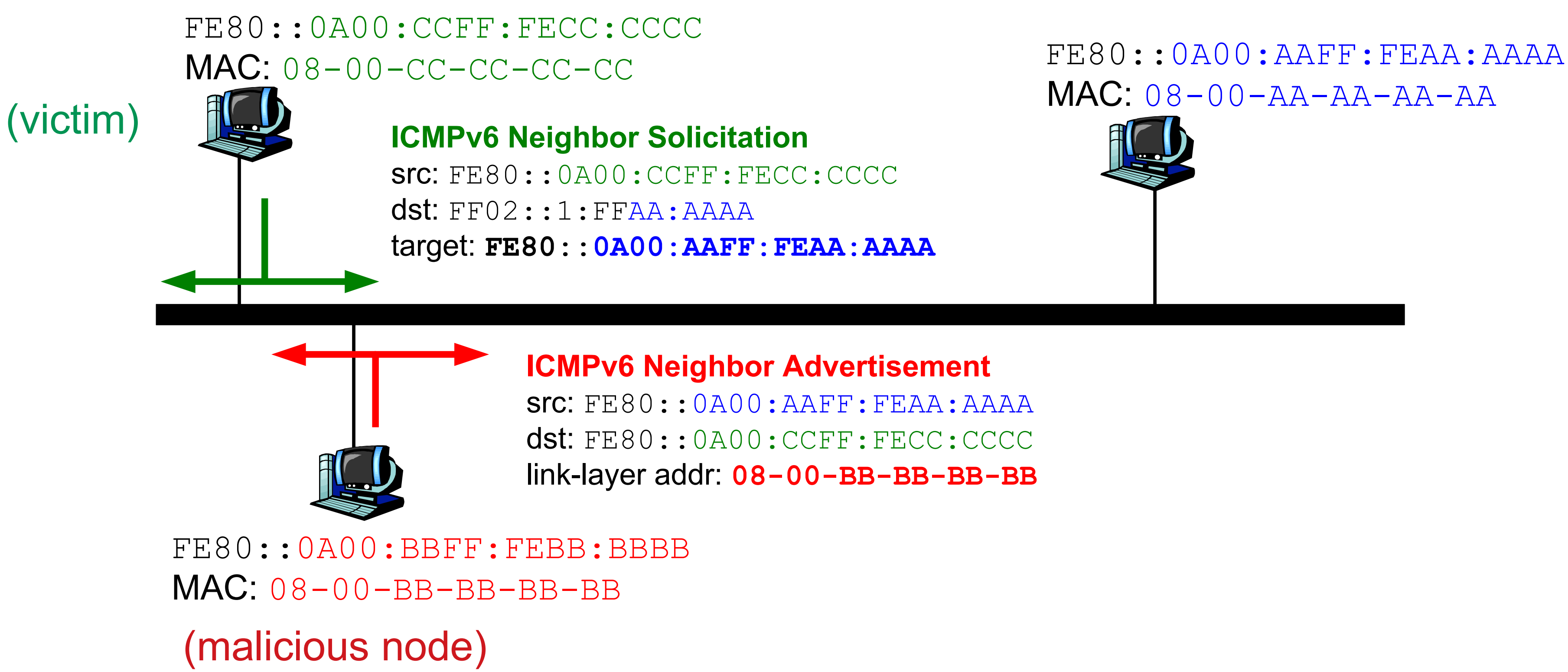
Security Risks

- What happens if an attacker sends **fake router advertisements** on LAN ?



Security Risks

- What happens if an attacker sends **fake neighbor advertisements** on LAN ?



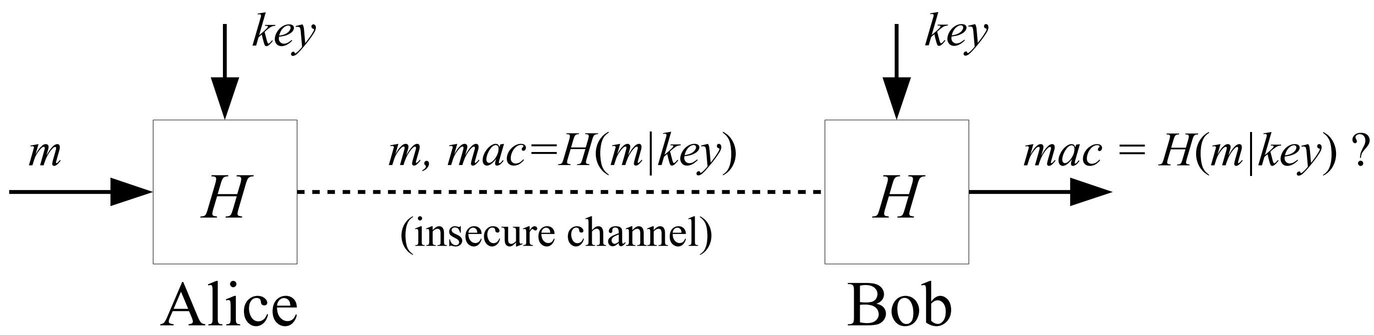
Securing ICMPv6

- **Principle of the solution**
  - Need to secure *Router Advertisements* and *Neighbor Advertisements*
    - A host that replies to an ICMPv6 *Neighbor Solicitation* should be able to prove that it owns the corresponding IPv6 address
    - A router that sends ICMPv6 *Router Advertisements* should be able to prove that it is authorized to serve as a router **and** that is authorized to announce the advertised prefixes

- **Issues**
  - How to exchange these proofs and authorizations ?
  - Is IPSec a solution ?

Cryptographical building blocks

- **Hash functions**



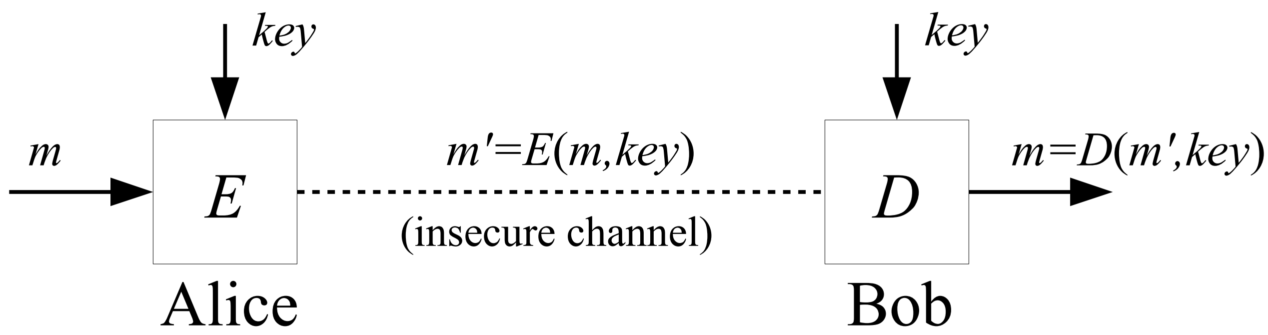
- Properties
  - Easy to compute $H(\mathrm{m})$
  - Very difficult to invert, i.e. find m_2 such that

    $H(\mathrm{m}) = H(m_2)$
  - Used to provide *Message Authentication Codes* (MAC)

- Example hash functions
  - MD5, MD4, SHA-1

Cryptographical building blocks

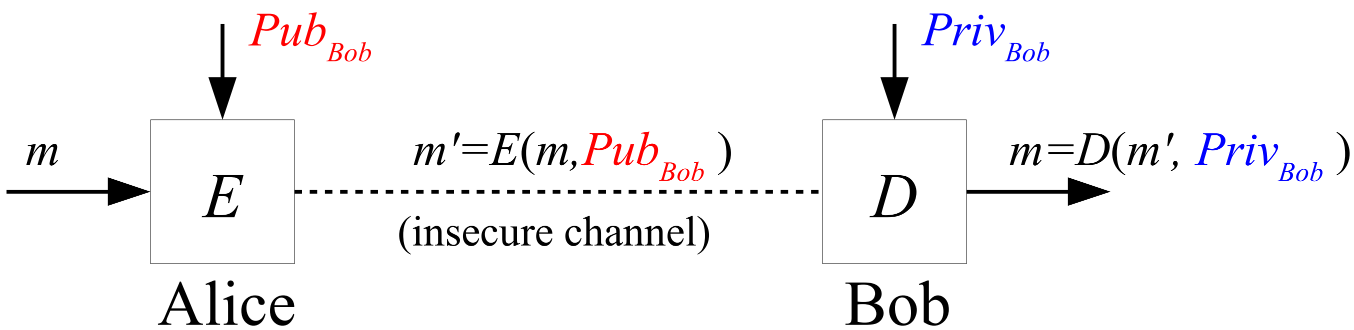
- **Secret-key Cryptography**



- Advantages
  - Efficient algorithms exist
  - Security is function of implementation and key size
- Drawbacks
  - Key must be distributed securely
  - Does not provide any authentication scheme
- Examples
  - DES, AES, RC-4, IDEA, ...

Cryptographical building blocks

- **Public-key Cryptography**



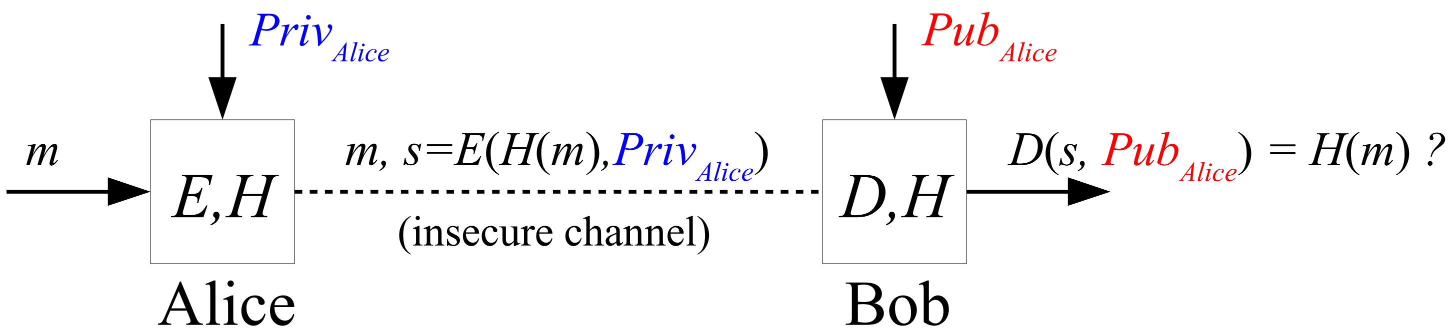
- Each user maintains two keys
  - A public key ($Public_{Key}$) which can be made public and can be used by any user to send him/her encrypted messages
  - A private key ($Private_{Key}$) which is kept secret and can be used to decrypt information encrypted with the public key.

Cryptographical building blocks

- **Public-key Cryptography**
  - Advantages
    - Users do not need to share a secret key to be able to encrypt messages
    - Public-key cryptography allows signatures
    - Security is function of implementation and key size
    - Can be used to sign messages (using private key)
  - Drawbacks
    - Public-key cryptography is 10 or 100 times slower than secret-key cryptography
  - Examples
    - RSA, DSS

Cryptographical building blocks

- **Digital Signature**



- Property of public-key cryptography
  - public-key and private-key can be exchanged in the encryption and decryption processes. This still works!
  - Basis for digital signatures
- Note
  - Digital signature only proves that message was originated by owner of private key used for signature. Not sufficient for authentication.

First Solution: Certificates

- **Principle**
  - Each router has a public/private key pair
  - A certificate is generated for each router to confirm
    - that the key pair belongs to the router
    - that the owner of the key pair is a valid router
  - Certificates must be anchored on an authority that is trusted by both routers and hosts
  - ICMPv6 router advertisement messages are signed by the router

- **Protocol issues**
  - Need to extend ICMPv6 to support signatures and certificates

Cryptographically Generated Addresses (CGA)

- **Observations**
  - Placing certificates on all hosts is too difficult
  - We usually don't need to prove that a host is a host

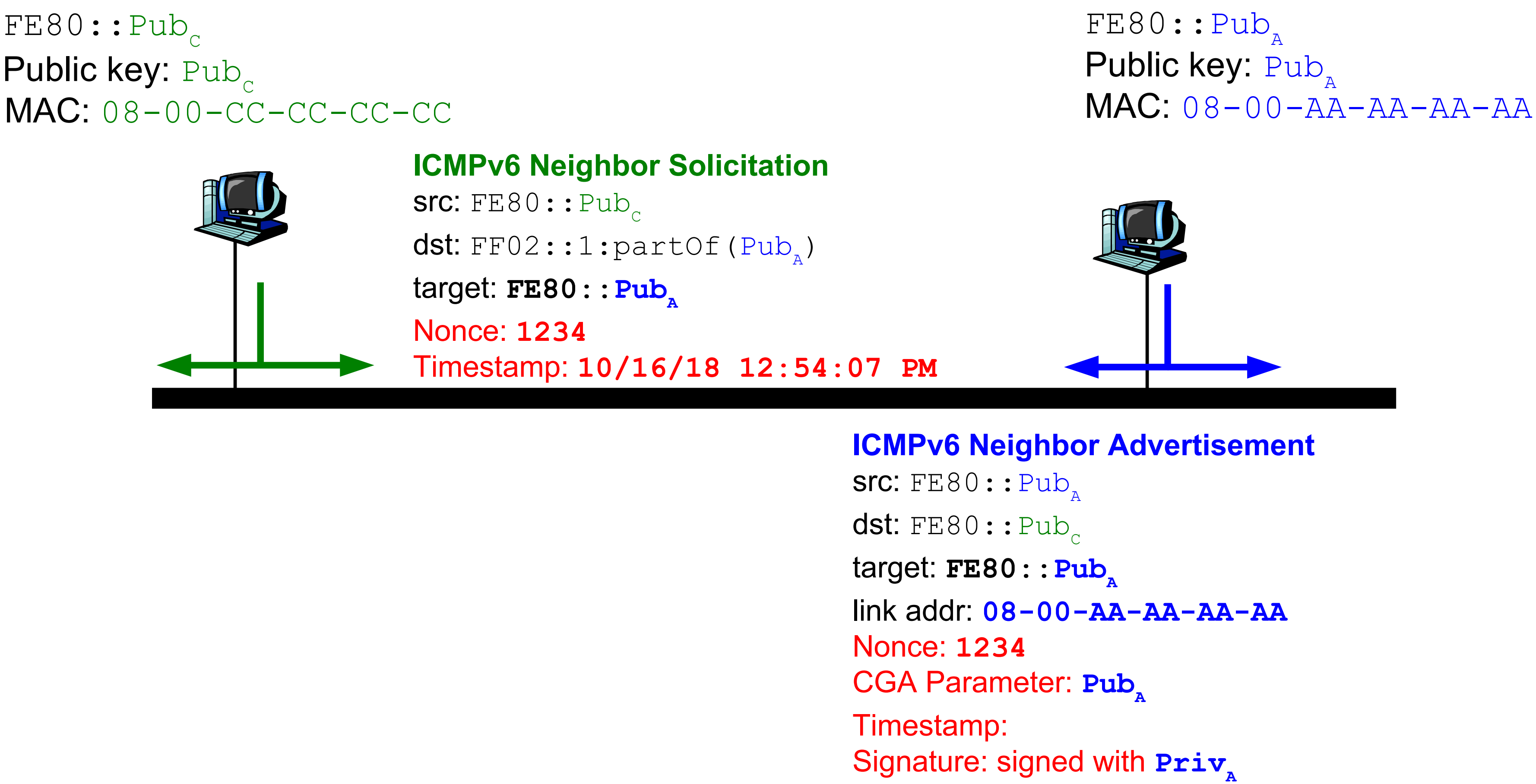
Can we verify the validity of signed messages without relying on a PKI ?

- **Principle of the solution**
  - Assume that IPv6 addresses are variable-length and unlimited length
  - Generate IPv6 addresses as follows

Global prefix + subnet ID	Host's public Key

  - Use private key to sign ICMPv6 neighbor advertisement messages

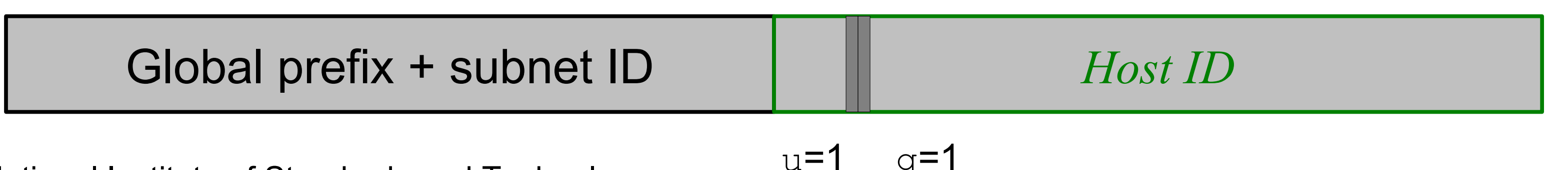
Principles of Secure Neighbor Discovery



Cryptographically Generated Addresses (CGA)

- **IPv6 addresses have a fixed size**
  - The Host ID is 64 bits and 2 bits (`u`,`g`) are reserved in (MEUI-64) → only 62 bits remain.
  - A 62 bits RSA public-key is not secure
    - NIST[(1)] recommends > 1024 bits

- **Solution**
  - To secure a binding between a MAC address and an IPv6 address, each host
    - generates its (Pub_{Key}, $Priv_{Key}$) key pair
    - uses a special $HostId = Hash_{64}(Pub_{Key})$, ignore (`u`,`g`) bits
    - signs its Neighbor Advertisements by using its $Priv_{Key}$



(1) NIST : National Institute of Standards and Technology

Cryptographically Generated Addresses (CGA)

- **Simplified version**

1). **host A** picks $(Pub_A, Priv_A)$

address' $HostId_A = Hash_{64}(Pub_A)$

2). later, **host B** sends *Neighbor Solicitation* (NS) for address $Prefix{:}HostID_A$

3). **host A** answers with *Neighbor Advertisement* (NA), signed with

$s = E(\,H(\mathrm{NA}), Priv_A)$

In addition, it provides the public key Pub_A and its link-layer address

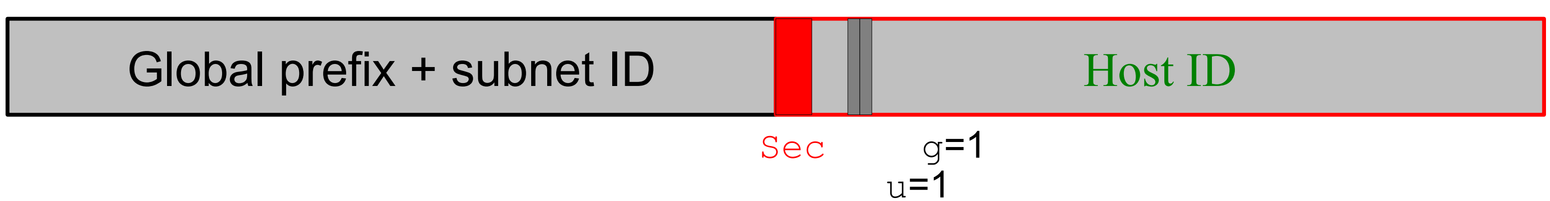
4). **host B** checks that

a). $Hash_{64}(Pub_A) = HostID_A$

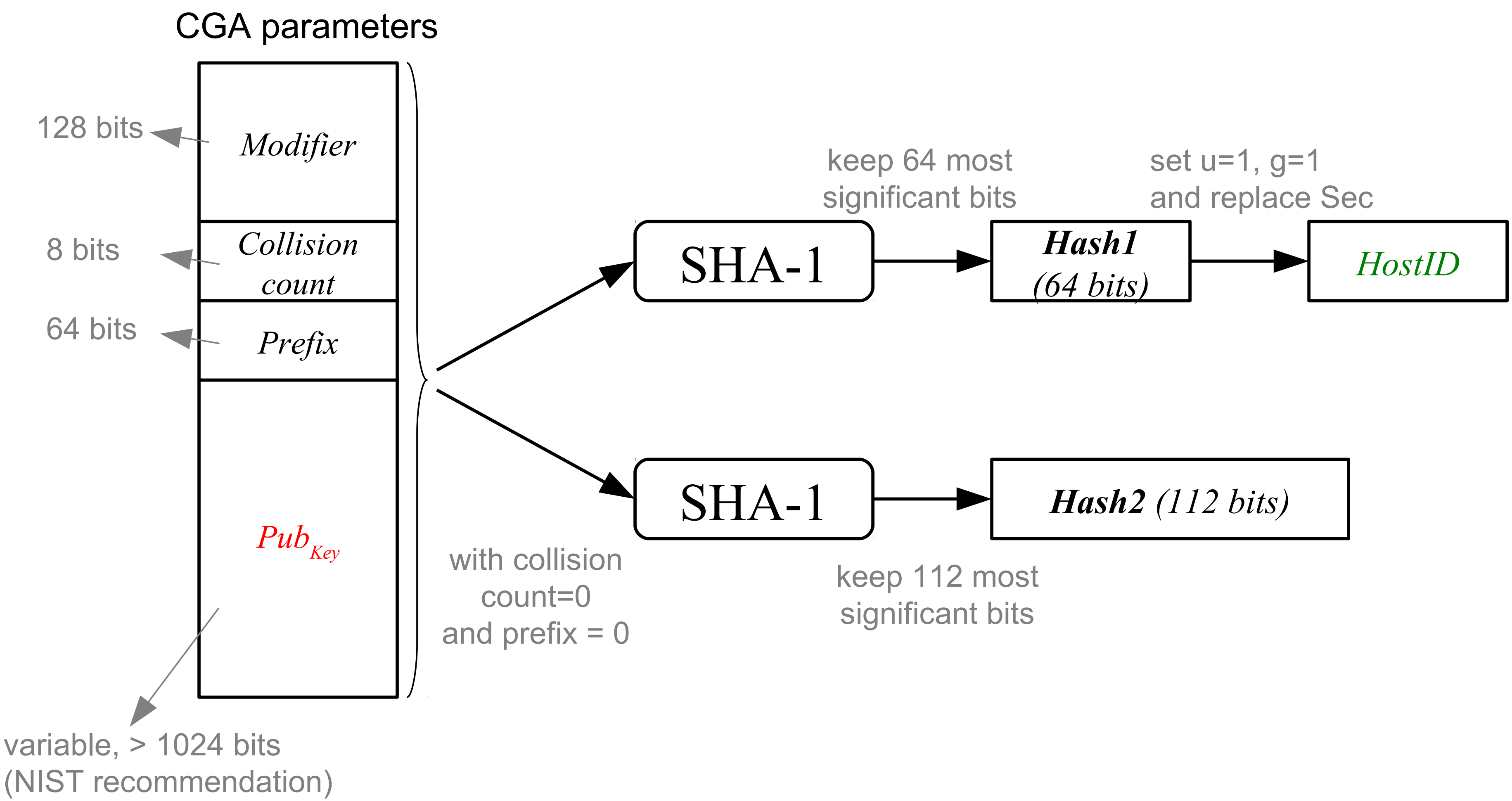
b). $D(s, Pub_A) = H(\mathrm{NA})$

Cryptographically Generated Addresses (CGA)

- **Issue with CGA**
  - A 62 bits hash is not very secure
    - an attacker could use brute-force to find a public-key whose hash is equal to a given value (dictionary attack)
- **Improving CGA security beyond 62 bits (RFC3972)**
  - Objective: Increase the difficulty for an attacker of computing the Host Id.
  - Basic idea: compute a large hash value and transmit only a part of this hash value, the remaining part being known (0).
    - $HostId = Low_{64}(Hash_{80}(Modifier \mid Pub_{Key}))$ and such that $High_{16}(Hash_{80}(Modifier \mid Pub_{Key}))=0$
    - Random *Modifier* value generated iteratively
    - Security level (*Sec*) encoded in Host Id



Cryptographically Generated Addresses (CGA)



CGA Generation

- **Algorithm (RFC3972)**
  1. Pick a random *modifier*
  2. Set the *collision count* to 0
  3. Hash the concatenation of CGA parameters with *collision count* and *prefix* set to 0

     $$Hash2 = Left_{112}(\text{ SHA-1}(\ modifier \mid \text{coll. count} \mid \text{prefix} \mid Pub_{Key}\ )\ )$$

  4. Compare the $16*Sec$ leftmost bits of $Hash2$ with 0. If they differ, increment *modifier* and go back to step (2).
  5. Hash the concatenation of *modifier*, *collision count*, *prefix* and $\mathrm{Pub}_{\mathrm{Key}}$

     $$Hash1 = Left_{64}(\text{ SHA-1}(\ modifier \mid \text{coll. count} \mid \text{prefix} \mid Pub_{Key}\ )\ )$$

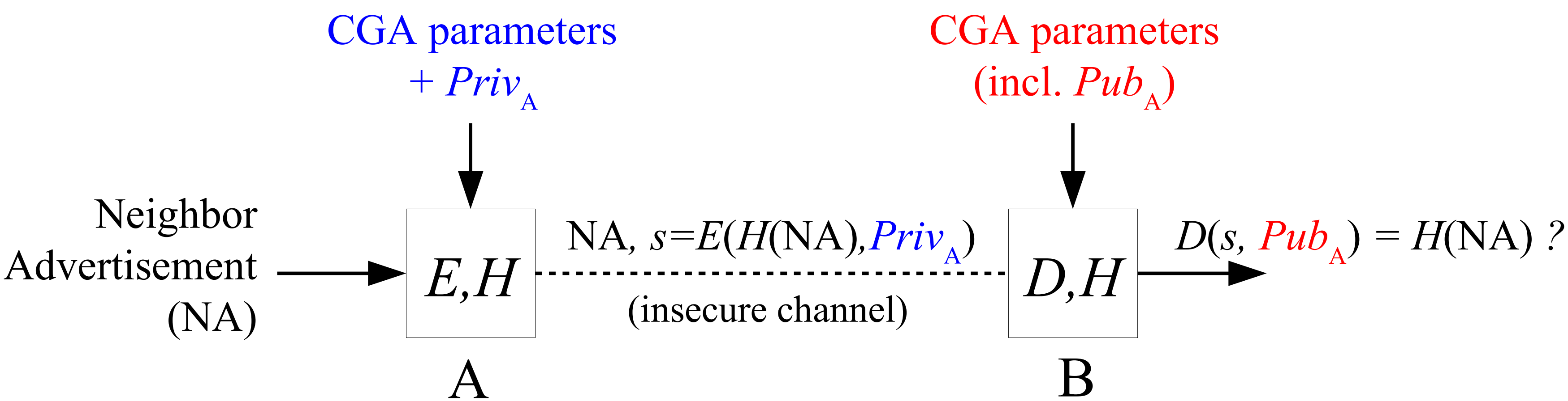
  6. Obtain the *HostID* by setting the “u” and “g” bits to 1 and the *Sec* bits.
  7. Perform DAD. If there is a collision, increment the *collision count* and go back to step 5.

CGA Verification

- **Algorithm (RFC3972)**
  1. Hash the concatenation of modifier, subnet prefix, collision count and PubKey obtained from the CGA parameters. The 64 leftmost bits are $Hash1$.
  2. Compare $Hash1$ with the HostID, ignoring the "u", "g" and Sec bits. If the comparison fails, the verification fails.
  3. Hash the concatenation of the CGA parameters ($collision$ $count$ and prefix set to 0). The 112 leftmost bits are $Hash2$.
  4. Compare the $16{*}Sec$ leftmost bits of $Hash2$ with zero. If the comparison fails, the verification fails.

CGA Signature

- **Generation and Verification**



- The receiver must
  - first verify the CGA address, i.e. check that the CGA parameters would lead to the same HostId
  - then check the digital signature

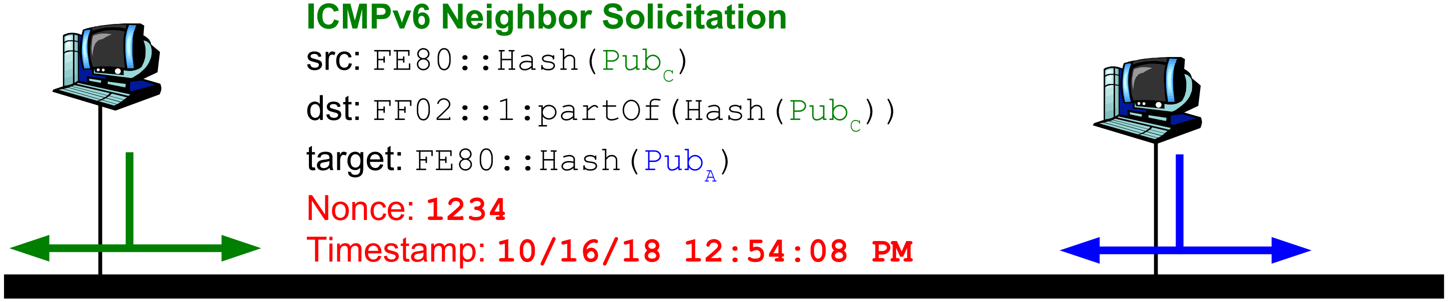
Extensions to ICMPv6

- **SEcure Neighbor Discovery (SEND)**
  - RFC3971
  - RSA Signature option
    - 128 most significant bits of SHA-1 of PubKey used to sign
    - digital signature computed over (random message tag, source address, destination address, ICMPv6 type, code and checksum, NDP header and options)
  - Timestamp option
    - used to avoid replay attacks
  - Nonce option
  - CGA parameter option
    - used to exchange CGA generation parameters: Modifier, collision count, subnet prefix and Pub_{Key}.

Secure Neighbor Discovery (SEND)

FE80::Hash(Pub$_C$)
Public key: Pub$_C$
MAC: 08-00-CC-CC-CC-CC

FE80::Hash(Pub$_A$)
Public key: Pub$_A$
MAC: 08-00-AA-AA-AA-AA



ICMPv6 Neighbor Solicitation

src: FE80::Hash(Pub$_C$)

dst: FF02::1:partOf(Hash(Pub$_C$))

target: FE80::Hash(Pub$_A$)

Nonce: **1234**

Timestamp: **10/16/18 12:54:08 PM**

ICMPv6 Neighbor Advertisement

src: FE80::Pub$_A$

dst: FE80::Pub$_C$

target: Hash(FE80::Pub$_A$)

link addr: **08-00-AA-AA-AA-AA**

Nonce: **1234**

CGA Parameter: **Pub$_A$**

Timestamp:

Signature: signed with **Priv$_A$**

CGA

- **The attacker's perspective**
  - To steal a CGA address, a malicious host should be able to either
    - find the private key used
    - or find a public/private key pair that hashes to the same target hostId
  - Both problems are now very difficult.

- **Note**
  - CGA addresses are also extended for use beyond SEND. See RFC4581

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

DNSv6

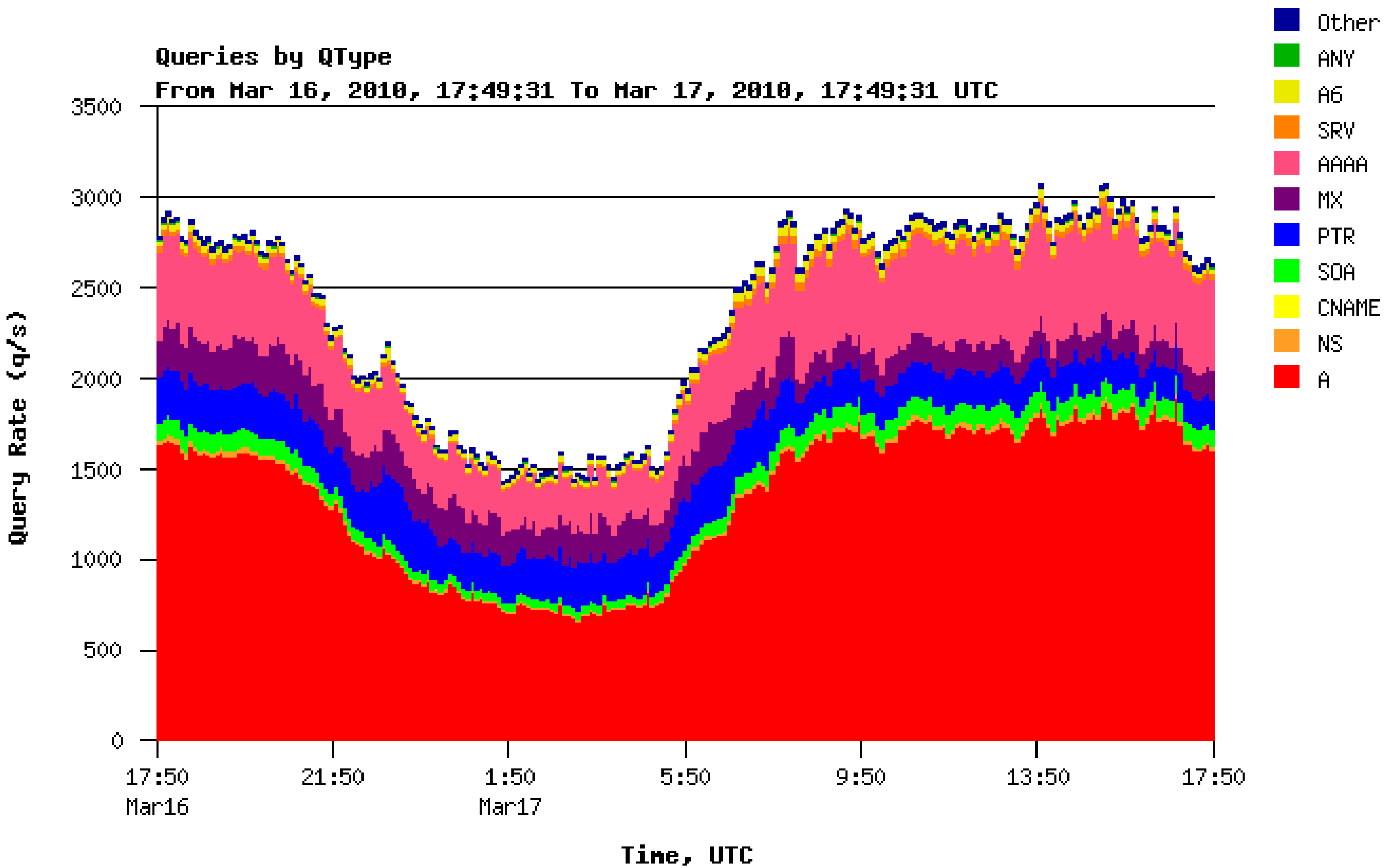
- Three problems to solve
  - How to encode IPv6 addresses in the DNS ?
  - How to support reverse DNS ?
  - How to perform all DNS requests by using only IPv6 ?

DNSv6

- Each DNS message is composed of *Resource Records* (RR) encoded as *Type-Length-Value* (TLV) <Name, Value, Type, TTL>

- Types of RR
  - A (IPv4 Address)
    - Name is a hostname and Value an IPv4 address
  - **AAAA (IPv6 Address)**
    - Name is a hostname and Value an IPv6 address
  - NS (NameServer)
    - Name is a domain name and Value is the hostname of a DNS server responsible for this domain
  - MX (Mail eXchange)
    - Name is a domain name and Value is the name of an SMTP server that must be contacted to send emails to this domain
  - CNAME (Canonical Name)
    - Alias

DNSv6



Source: http://k.root-servers.org/statistics/denic/daily/
(retrieved on 17/03/2010)

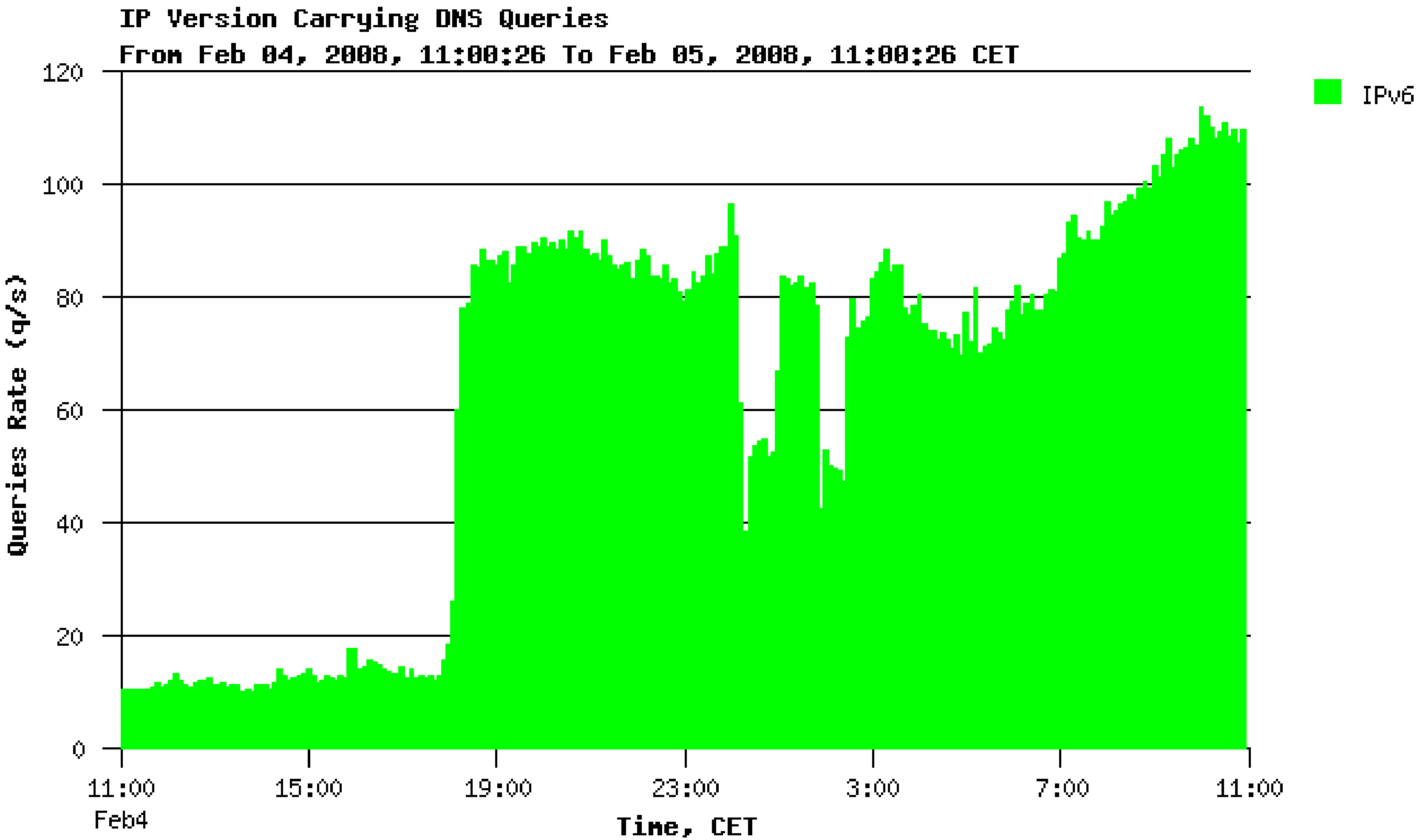
Supporting Reverse DNS

- Solution: special domain `IP6.ARPA`
  - Encode IPv6 address in reverse order, one character per group of 4 bits
  - perform classical DNS query
  - Example
    - `4321:0:1:2:3:4:567:89AB`
    - `b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.IP6.ARPA.`

Adding IPv6 addresses to the DNS root

- Took a much longer time than expected
  - Initially DNS root was only reachable via IPv4
    - List of DNS root servers is encoded in one DNS reply
    - All DNS implementations must support DNS replies of 512 bytes, but encoding of the 13 IPv4 DNS root servers already consumes 400 bytes. Adding IPv6 for all DNS root servers requires 811 bytes in the reply
  - Several TLD moved quickly to IPv6
    - One IPv6 authoritative server for `.be` since sept. 2004
  - DNS was extended to support larger replies
  - February 2008
    - 6 root DNS servers support IPv6
    - IPv6-only hosts can at last use the DNS
    - Today (9/2014), 11 among 13 root servers

Adding IPv6 addresses to the DNS root



Source: k-root server (retrieved on 17/03/2010)

IPv6

- 4. 1 Motivations
- 4.2 IPv6 Addressing Architecture
- 4.3 IPv6 Packets
- 4.4 ICMPv6
- 4.5 DNS support for IPv6
- 4.6 Transition Mechanisms

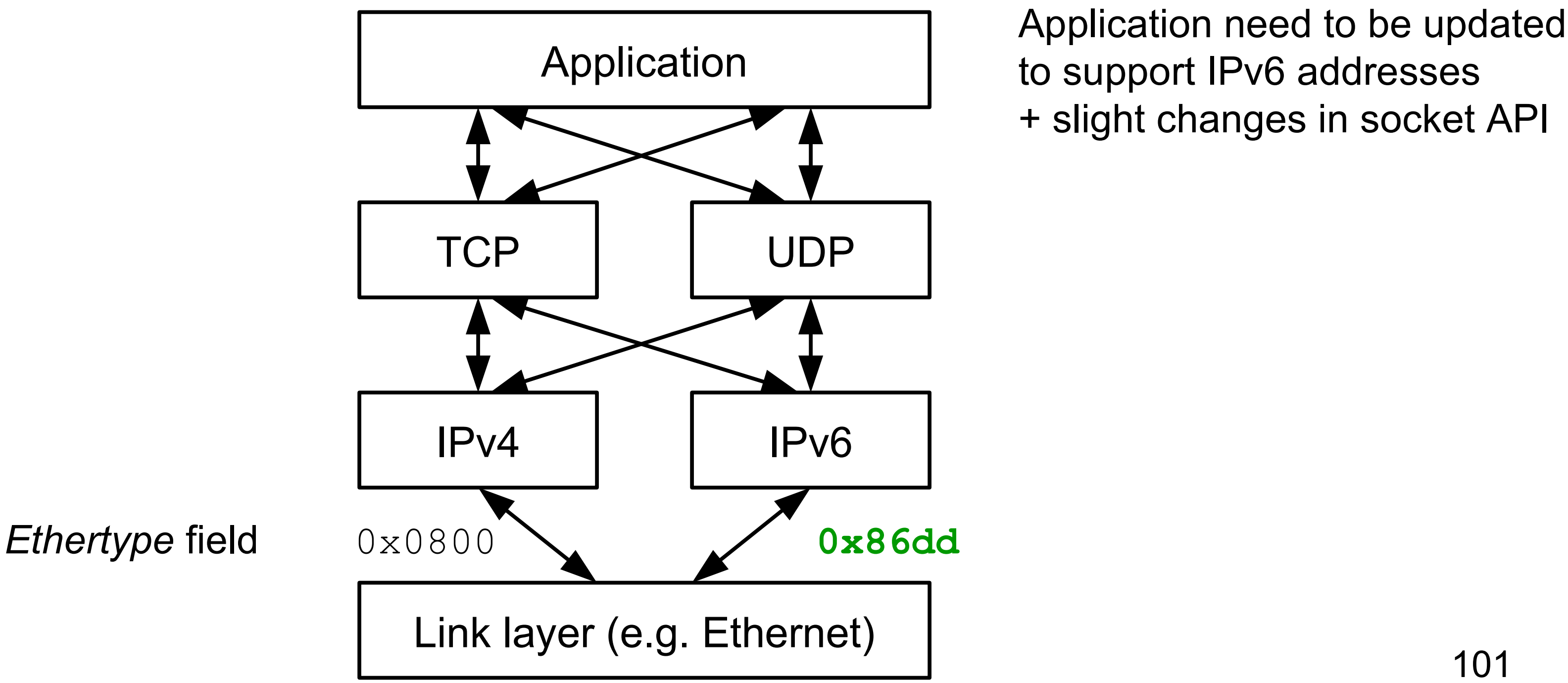
Transition From IPv4 To IPv6

- Not all routers and end-hosts can be upgraded simultaneously
  - There will be no “flag days”
  - How will the network operate with mixed IPv4 and IPv6 routers?

- Several techniques have been proposed
  - **Dual-stack**
  - **Tunneling**: IPv6 carried as payload in IPv4 datagram among IPv4 routers
  - **Translation**

Dual Stack Techniques

- **Dual-stack node**
  - has support for both protocol versions
    - behaves as an IPv6 node when in communication with another IPv6 node
    - behaves as an IPv4 node when in communication with another IPv4 node



Dual Stack Techniques

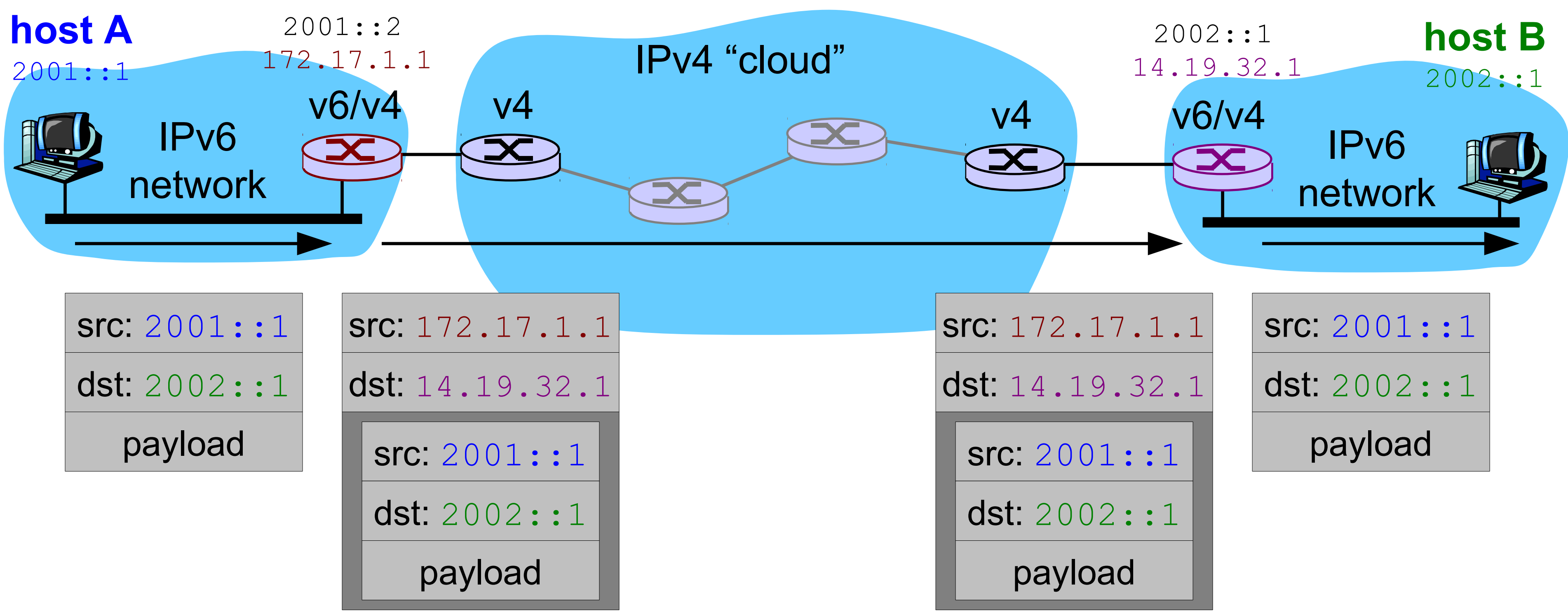
- **Dual-stack node**
  - Most end-host OS today (linux, FreeBSD, Mac OS X, and even Windows) support IPv6 !
  - Usually, the DNS is used to detect if the remote host supports IPv4 and/or IPv6.
  - Very often, the local OS has a priority for the IP version to use.
    - For example, Mac OS X used to first try the IPv6 address.
  - Benefit:
    - straightforward
  - Drawbacks:
    - need to store state for both network protocol stacks (i.e. routing table, ...); some v4/v6 commands/options differ; ...
    - People don't know IPv6 enabled on their machine → don't configure firewall accordingly → Danger !

Dual Stack Techniques

- **Dual-stack network**
  - Need to update core devices so that they can support IPv6
  - **Switches**
    - operate at layer 2 (below v6) → should be transparent
    - But most switches have a management layer (using IP)
    - But IP Multicast help from switches (IGMP snooping)
  - **Routers**
    - routing protocols need to be updated
      - RIP → RIPng
      - OSPF → OSPFv3
      - BGP → Multi-Protocol BGP (MP-BGP)
    - Need to support hardware based IPv6 forwarding (in particular IPv6 lookup)
  - Other infrastructure services
    - Security (firewalls, DPI), DNS, NAT, ...

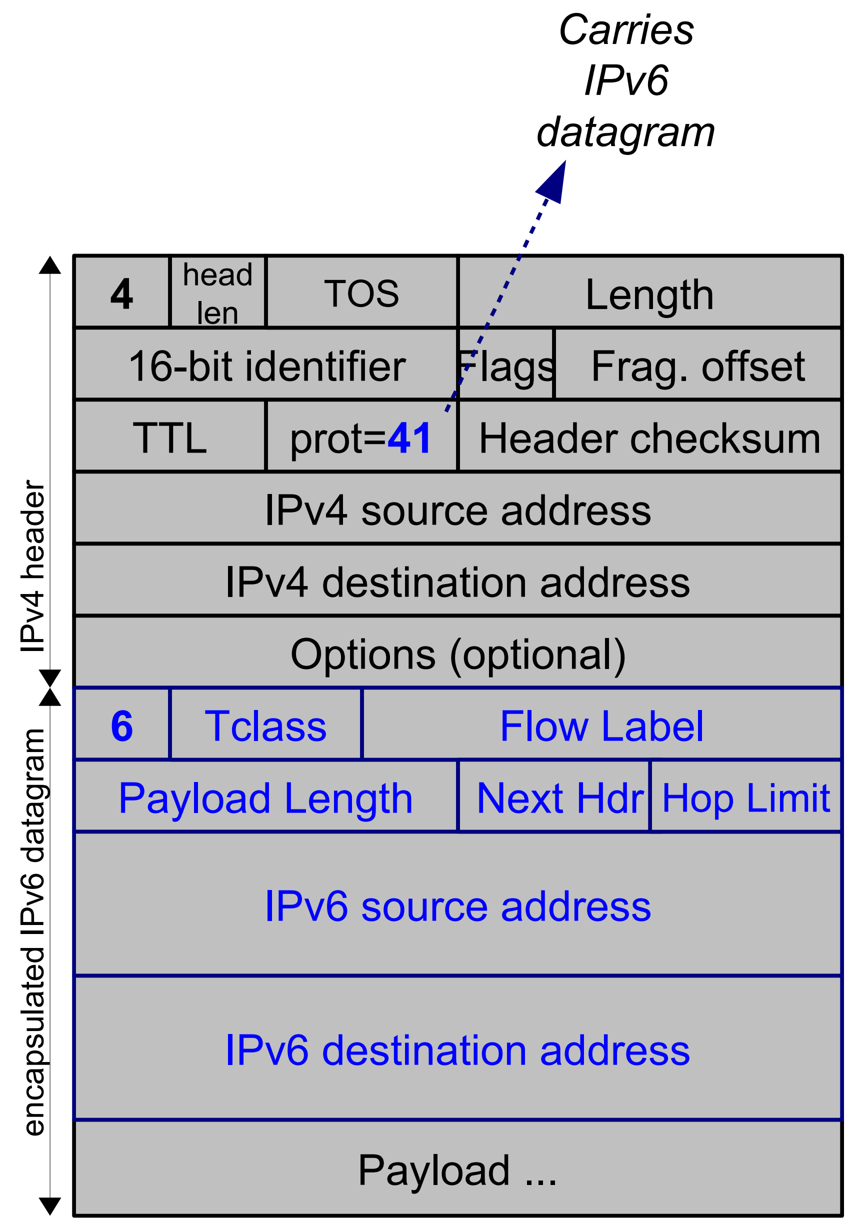
Tunneling Techniques

- **Objective**
  - Typically deploy an IPv6 forwarding infrastructure over the existing IPv4 infrastructure.
  - Tunneling = encapsulation



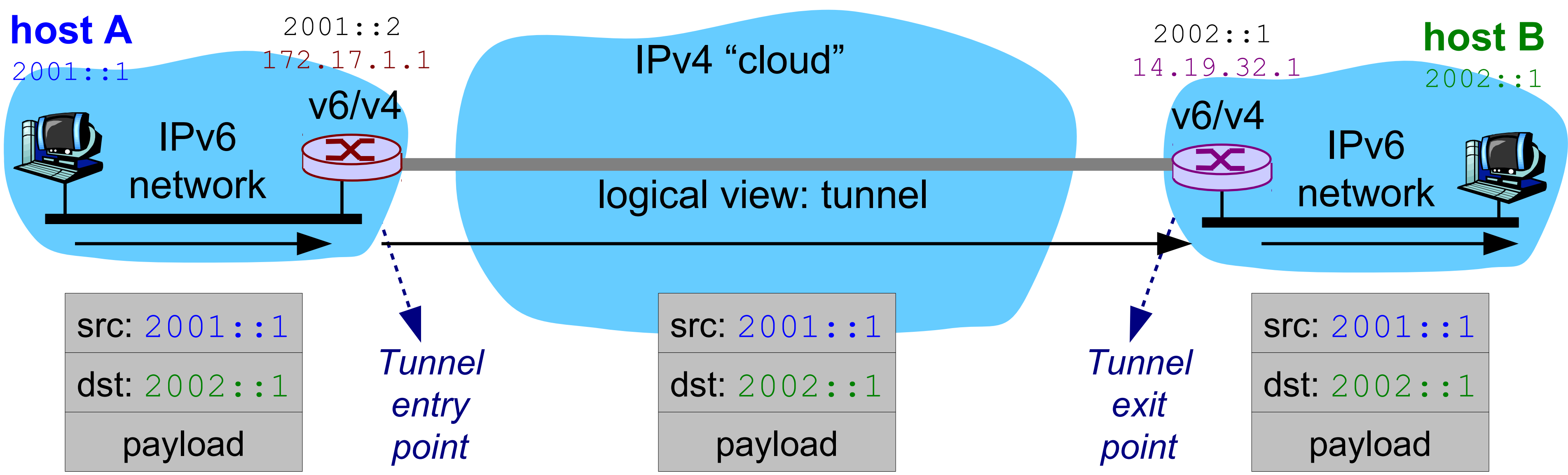
Tunneling Techniques

- Different encapsulation standards
  - IP in IP
  - *Generic Routing Encapsulation* (GRE), IPSec, ...
- Issues
  - MTU
    - IPv6 requires 1280 bytes
    - IPv4 requires 576 bytes
  - TTL
    - what value to use ? (implementation dependent)
  - ICMPv4 errors within tunnel



Tunneling Techniques

- **Objective**
  - Typically deploy an IPv6 forwarding infrastructure over the existing IPv4 infrastructure.
  - Tunneling = encapsulation



Tunneling Techniques

- **Two general types of tunneling**
  - Manually configured
    - tunnels head- and tail- ends are configured manually
  - Automatic tunneling
    - 6to4, ISATAP, Teredo
    - relies on the use of special addresses to dynamically tunnel IPv6 packets over an IPv4 routing infrastructure.
    - IPv6 addresses usually embed an IPv4 address.

Tunneling Techniques

- **6to4** (RFC3056)
  - transition mechanism that allows IPv6 islands to communicate with each other over an IPv4 infrastructure without explicit setup
  - 6to4 routers/gateways
  - Special IPv6 prefix assigned by IANA: `2002::/16`
    - 32 bits after prefix represent IPv4 address of gateway

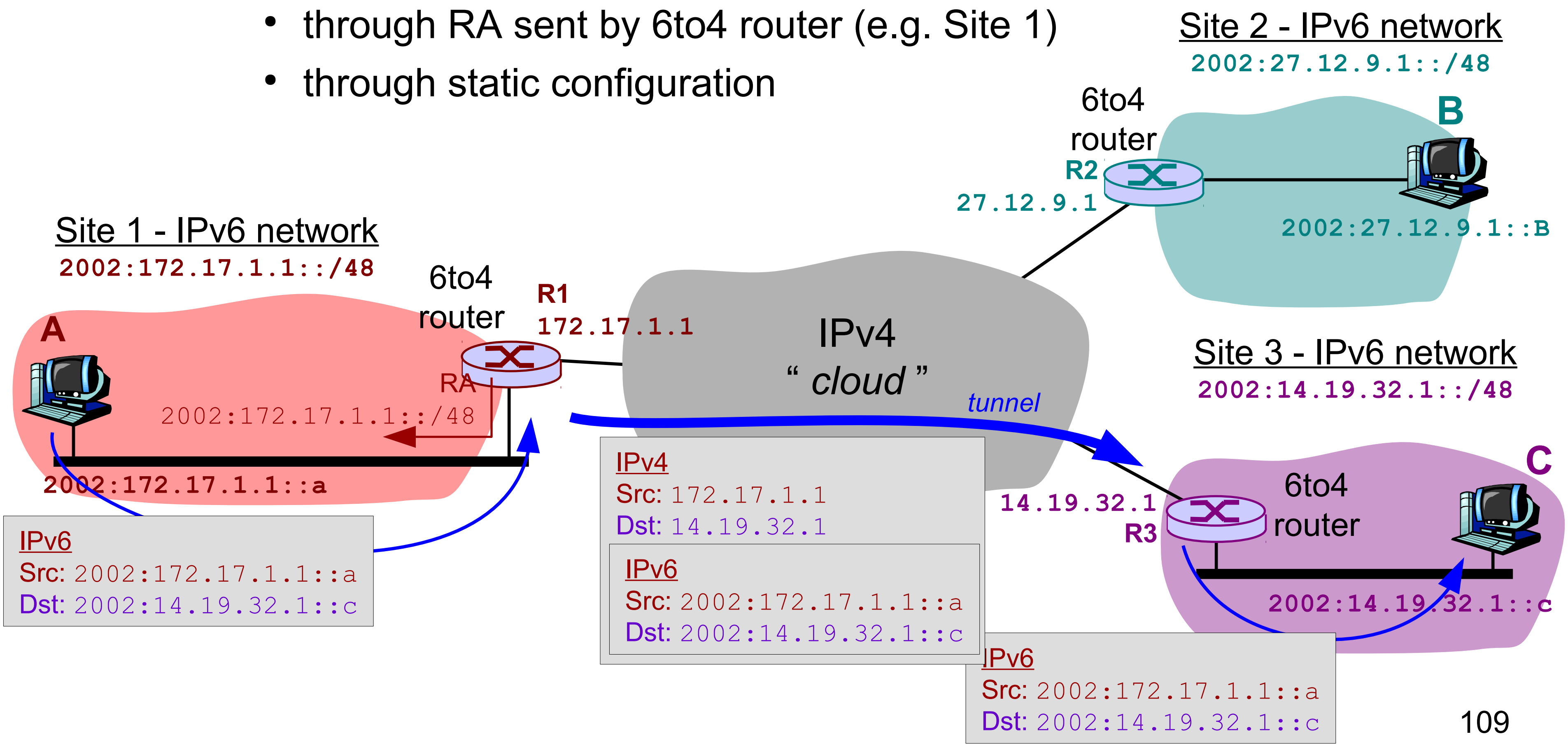
2002	IPv4 addr	Subn. ID	Interface ID

    - Example prefix of a 6to4 network: `2002:172.17.1.1::/48`

  - for some statistics, see also ***Observations of IPv6 Traffic on a 6to4 Relay***, Pekka Savola, ACM SIGCOMM Computer Communications Review, Volume 35, Number 1: January 2005

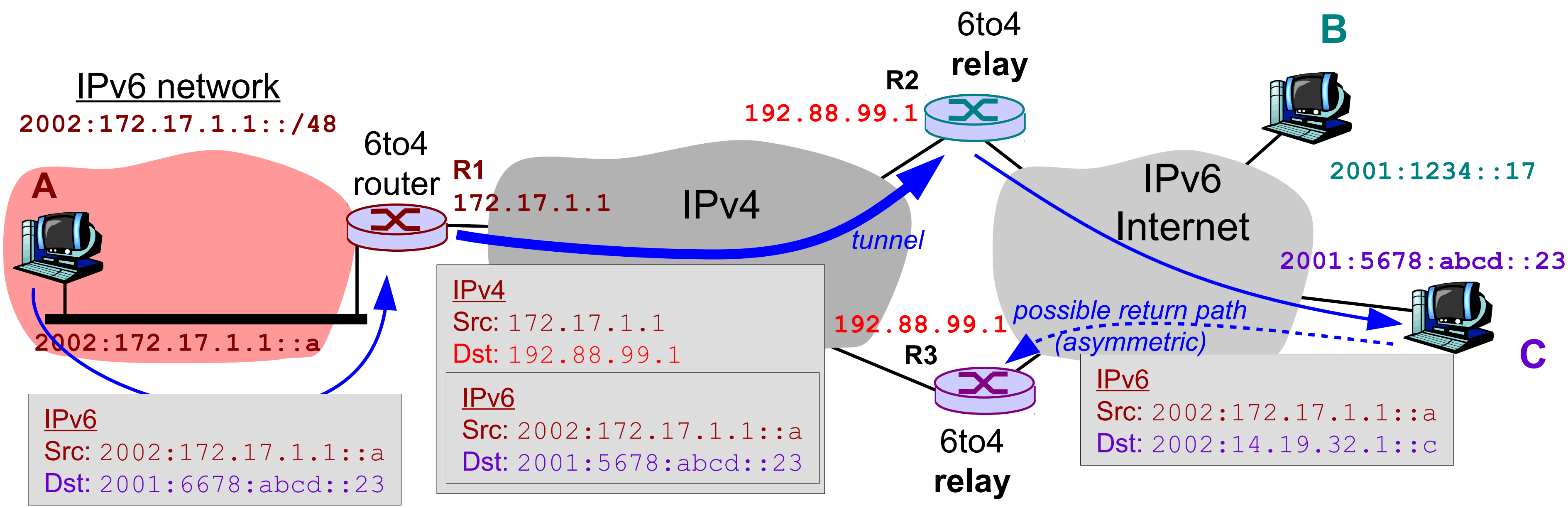
Tunneling Techniques

- **6to4 – peer-to-peer deployment**
  - allows isolated IPv6 sites to be connected
  - internal IPv6 prefix obtained
    - through RA sent by 6to4 router (e.g. Site 1)
    - through static configuration



Tunneling Techniques

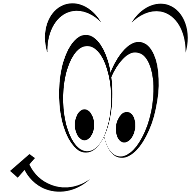
- **6to4 – access to IPv6 Internet**
  - allows isolated IPv6 sites to connect to IPv6 Internet
  - 6to4 relays
  - discovery of relays : anycast address `192.88.99.1`
  - routes to `2002::/16` through 6to4 relays



Tunneling Techniques

- **6to4 – access to IPv6 Internet**
  - too many issues (RFC 6343)
    - long RTT to relays (I measure 300ms at home)
    - asymmetric routes (different relays used for inbound/outbound)
    - firewall rules
    - slow 6to4 tunnel failure detection (up to several seconds)
→ Happy Eyeballs
    - does not work behind NATs (protocol 41)
  - Recent decision to deprecate 6to4 (RFC 7526, May 2015)
  - 6rd: Improved 6to4 used by ISPs as a temporary solution

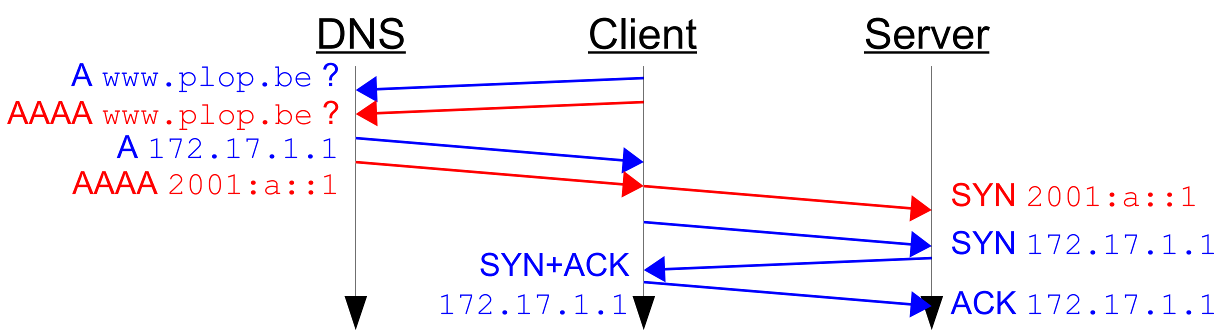
Happy Eyeballs



Which IP version shall be used by the OS / application for a new connection ?

- First approach
  - Try IPv6 first. If IPv6 fails, try IPv4.
  - Issue: failure detection can be quite long (several seconds) → activation of IPv6 can worsen user's network experience !

- Happy Eyeballs approach
  - Try IPv6. If no answer within say 300ms, try IPv4 in parallel.



Tunneling Techniques

- **Teredo (RFC4380)**
  - Issue with 6to4 : hosts need a public IPv4 address
  - 6to4 does not work with hosts behind NAT
  - **Teredo** = encapsulate IPv6 datagrams in UDP.
    - *Teredo clients* register with a *teredo relay*
    - Relay can be found through *teredo servers*
    - Special addresses
      `PFX : IPv4_srv : flgs : UDPport_NAT : IPv4_NAT`(1)
    - Special prefix used `2001::/32`
  - Warning : Supported and enabled on some Windows versions (e.g. Vista) using a default server (`teredo.ipv6.microsoft.com`, now deprecated ?)

(1) UDPport_NAT and IPv4_NAT are "obfuscated" (inverse bit values)

Tunneling Techniques

- **Teredo**

(1) IPv6/UDP/IPv4
src6=A, dst6=B
srcp=1234, dstp=3544 (teredo)
src4=A, dst4=S1

(2) IPv6/UDP/IPv4
src6=A, dst6=B
srcp=2345, dstp=3544
src4=A, dst4=S1

(3) ICMPv6 echo request
src6=A, dst6=B

(4) ICMPv6 echo reply
src6=B, dst6=A

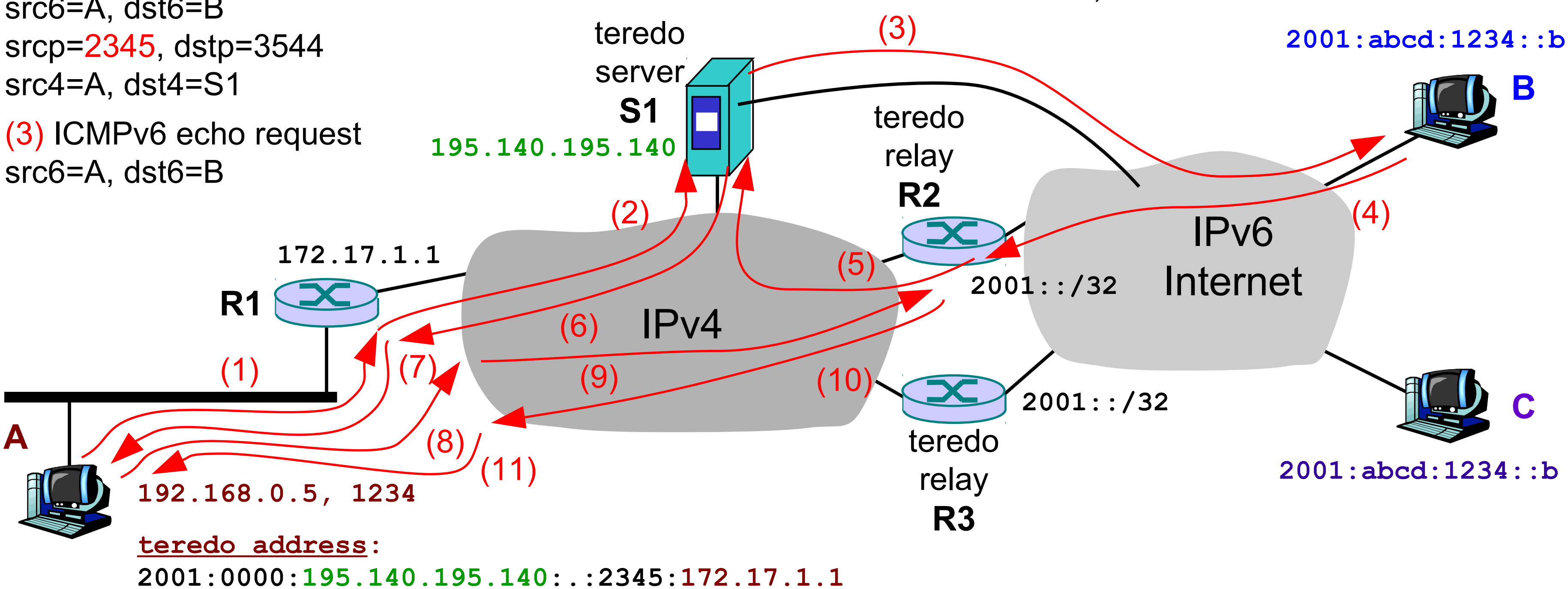
(5) IPv6 "indirect bubble"
src6=R1, dst6=A

(6) IPv6/UDP/IPv4 "ind. bubble"
src6=R1, dst6=A
origin indication=R2
srcp=3544, dstp=2345
src4=S1, dst4=A

(7) IPv6/UDP/IPv4 "ind. bubble"
src6=R1, dst6=A
srcp=3544, dstp=1234
src4=S1, dst4=A

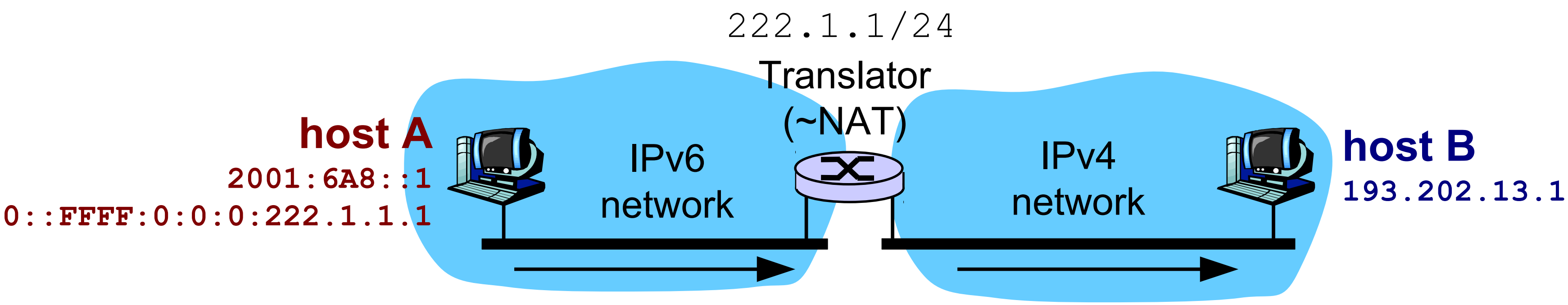
(8) IPv6/UDP/IPv4 "bubble"
src6=A, dst6=R1
srcp=1234, dstp=3544
src4=A, dst4=R2

(9) IPv6/UDP/IPv4 "bubble"
src6=R1, dst6=A
srcp=1234, dstp=3544
src4=A, dst4=R2



Translation

- **Stateless IP/ICMP Translation** (SIIT, RFC2765)
  - Special IPv6 addresses in prefix `0::FFFF:0:0:0/96`
  - Interface ID part is an IPv4 address



src: 0::FFFF:0:0:0:222.1.1.1
dst: 0::FFFF:0:0:0:193.202.13.1
payload

IPv6 datagram

src: 222.1.1.1
dst: 193.202.13.1
payload

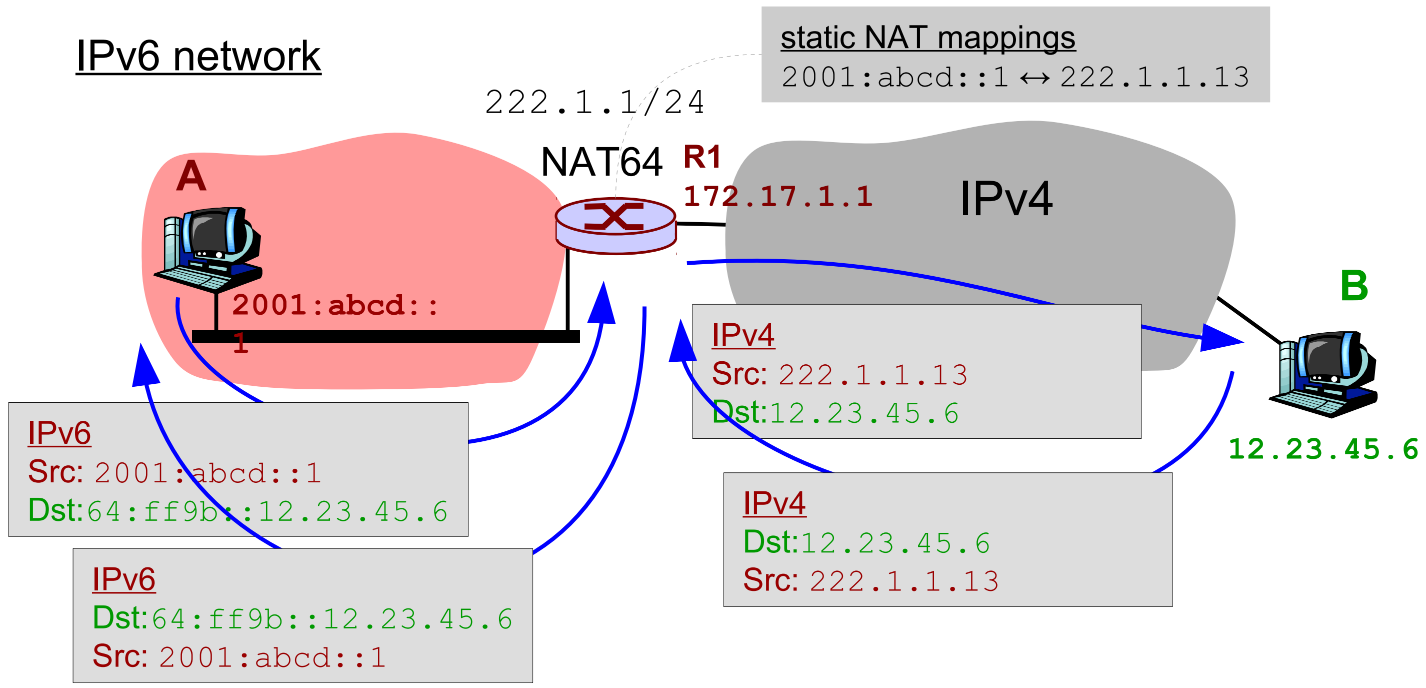
IPv4 datagram

Translation

- **Issues with IPv4 ↔ IPv6 translation**
  - some fields have no counterpart, IPv6 options, fragmentation vs PMTUD, ...
  - translation is expensive (CPU, memory)
  - constraints on topology (replies have to come through same NAT)
  - breaks applications with IP addresses in the payload
  - you don't want NAT in IPv6

Translation

- **NAT64 (RFC6144)**
  - Allocated prefix `64:ff9b::/96`
  - Global prefix can be used as well
  - NAT64 maintains 1:1 mappings for internal hosts (drawback)
  - Typical use case: provide IPv4 access to IPv6 server



Translation

- **Stateful NAT64**
  - Pool of IPv4 addresses shared among internal hosts
  - Use of DNS64 (RFC 6147) to synthesize AAAA records for remote IPv4 hosts