# Summary: why we need CIDR, subnetting

important

and how to distinguish the two

- Two different solutions to the same goal: more efficient use out of limited IP address space
  - By deciding the #bits in an IP address for network ID
- Subnetting: network operators configure a subnet mask to the routers within the destination network the length of each address block (network ID) in the router's forwarding table
- Classless InterDomain Routing:
  - Routing protocols tell routers the length of each address block (network ID) in the router's forwarding table (e.g. 31.179.0.0/16)

# Some notes and clarifications

- You configure router interfaces with address and subnets first
  - Configure link1 223.1.1.4  255.255.255.0
    - Another presentation of /24, used **only in subnetting**
  - Configure link2 …
- Then configure individual host
  - Printer
    - address 223.1.1.1/24
    - router 223.1.1.4 (if dest not in local subnet, go this way)
  - PC
    - address 223.1.1.2/24
    - router 223.1.1.4
- At the end, router runs routing algo to build FIB

# Few more words on subnets

- Subnets are determined by physical connectivity
  - Subnets pre-exist before you address them
- Subnets are identified by address block
- A random address block != a subnet

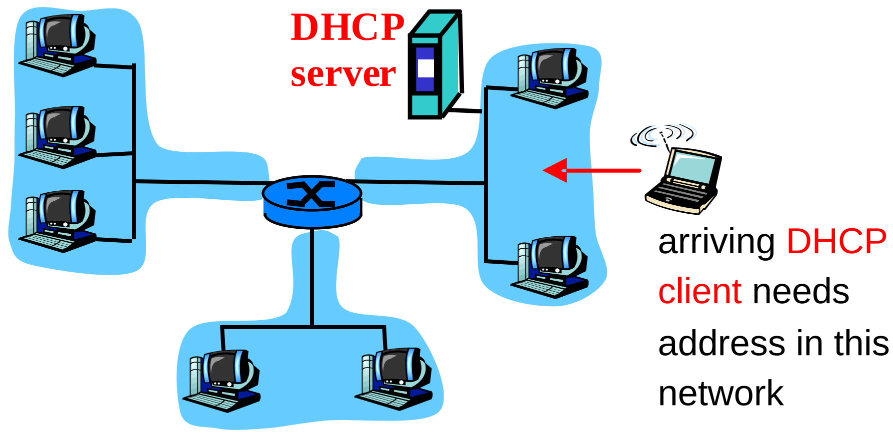
# Host Configuration

- An IP host must be configured with the following information to be able to send and receive data
  1. *IP address of an interface*
  2. *subnet mask*
  3. *Default router's IP address*
  4. *DNS caching resolver IP address(es)*
- Can be hard-coded by system admin in a file
  - Windows:

    control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config
- Can obtain the above info from Dynamic Host Configuration Protocol (DHCP)

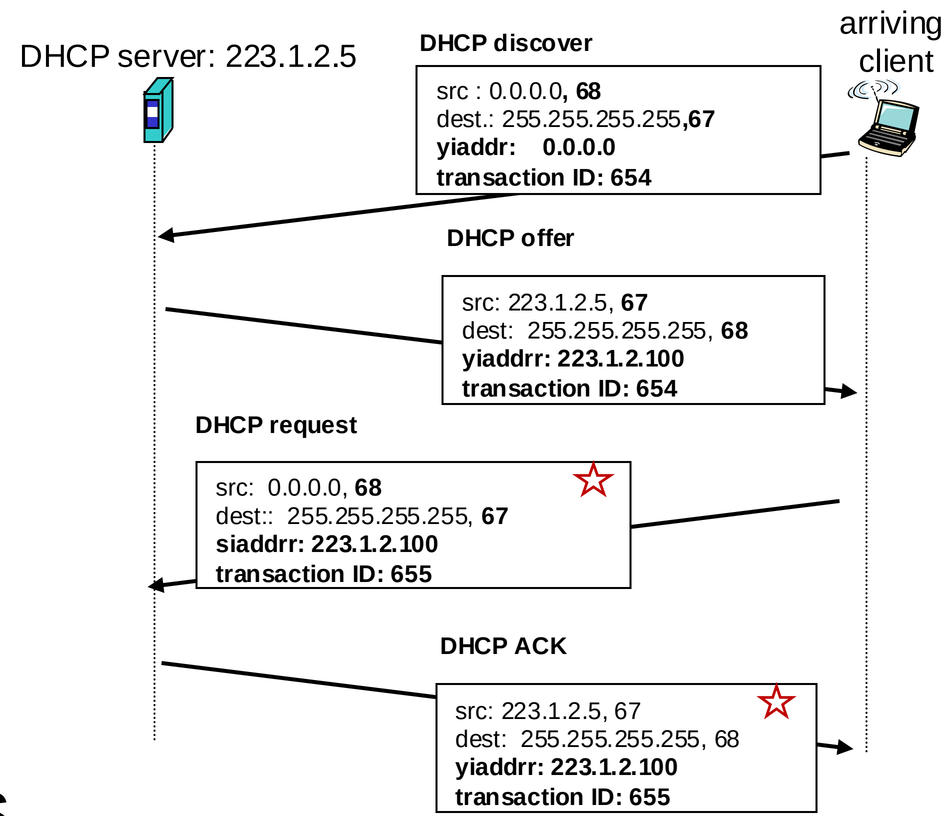
# DHCP Overview

- A new host sends "**DHCP discovery**"
  - Each network must have one DHCP server or its proxy
    - The proxy helps forward the DHCP discovery msg to the server
- DHCP server responds with "**DHCP offer**"
- Host sends: "**DHCP request**"
  - In reality this is accepting the offered config. parameter
- DHCP server sends address: "**DHCP ack**"
  - Confirmation of the offer



# DHCP Client-Server Scenario

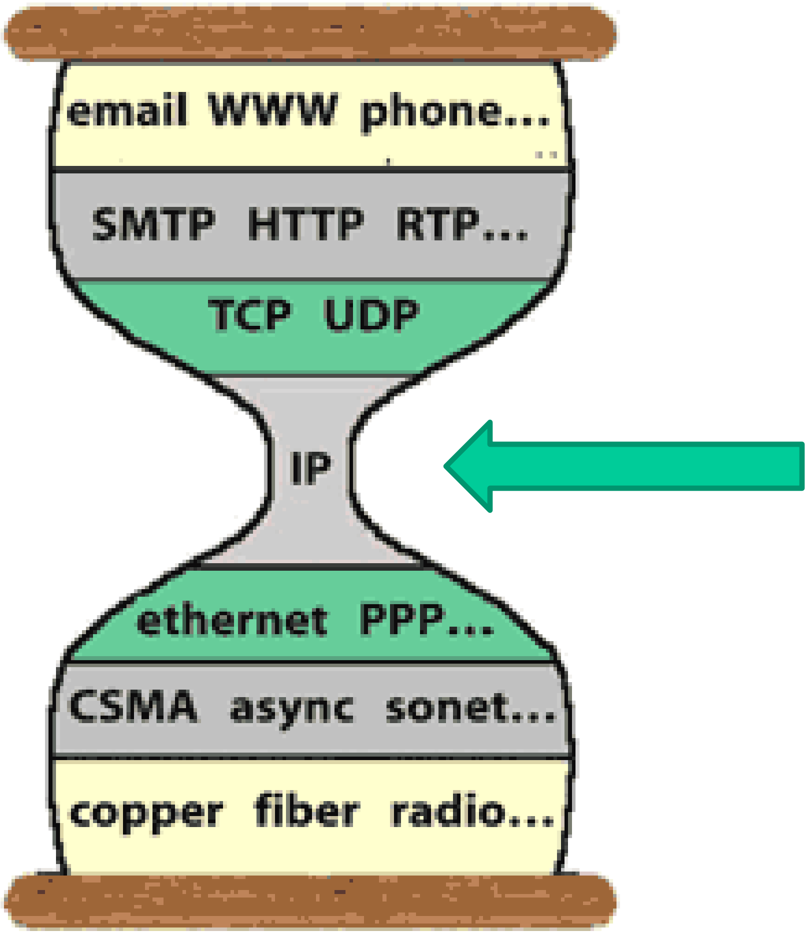
- DHCP runs over UDP
- All messages (discover, offer, request, ack) sent to
  - 255.255.255.255
- Client's (discover, request) src address always
  - 0.0.0.0
- Server's (offer, ack) src address is a valid IP
- Network configuration is "leased" for a given time period
  - Can be "renewed"




☆This msgs could be unicast instead of broadcast

Address, netmask, gateway, (optionally) DNS

# Lecture 10: NAT and IPv6




**Chapter 4:**

**4.1 Overview of Network layer**

4.2 What's inside a router

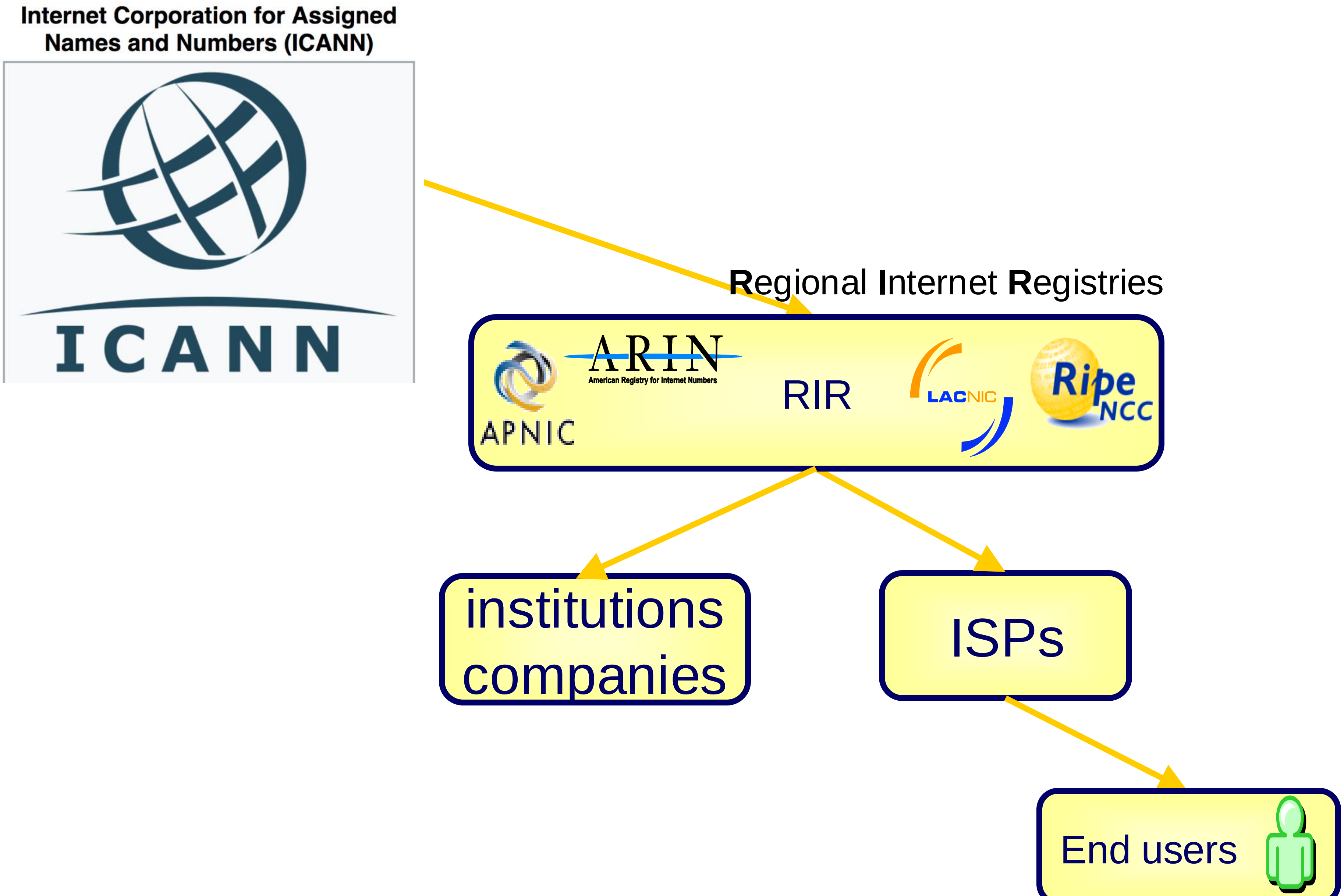
**4.3 IP: Internet Protocol**

- IP packet format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN

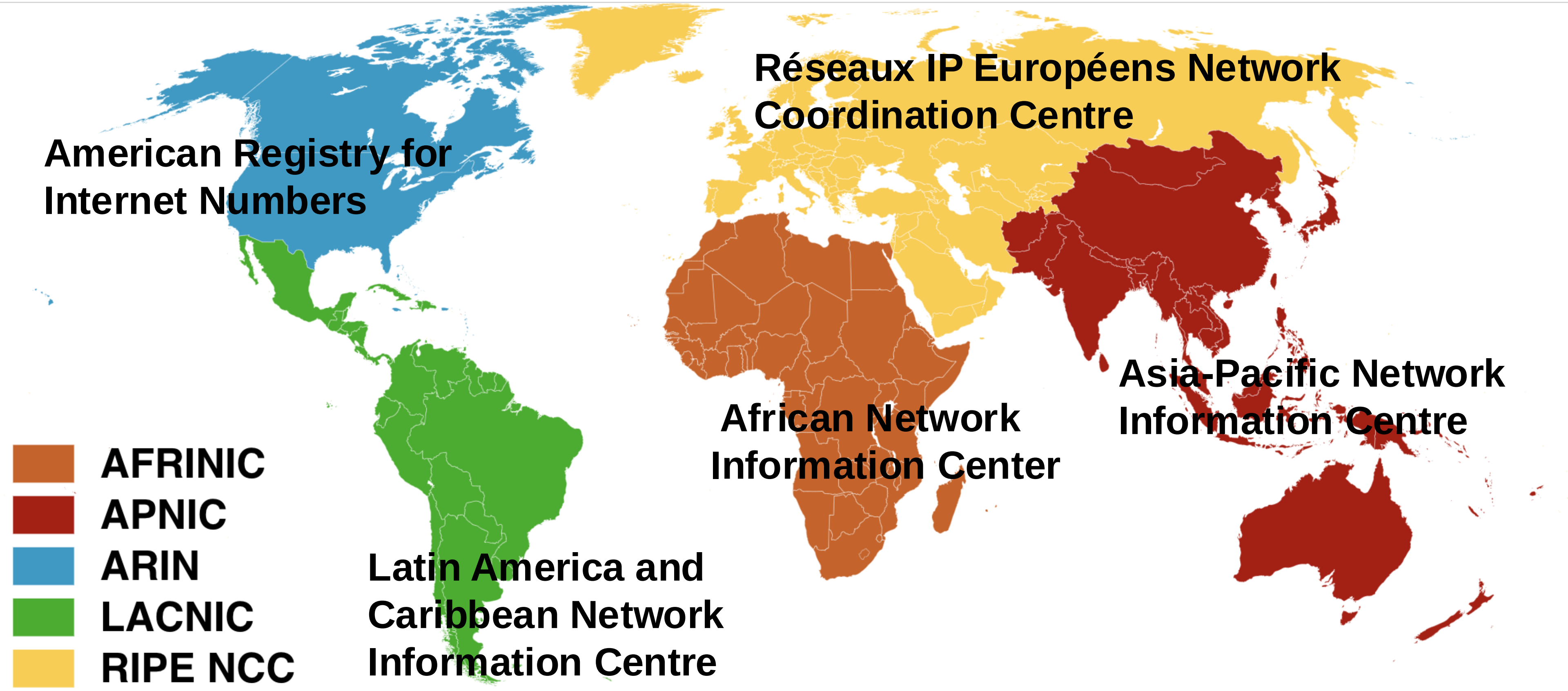
# IP address space management

FYI



# The 5 Regional Internet Registries

FYI



# How many IPv4 addresses ICANN still has

FYI

NEWS

## Update: ICANN assigns its last IPv4 addresses

By Stephen Lawson FOLLOW

IDG News Service | Feb 3, 2011 3:01 PM PT ← **Feb 3, 2011**

RELATED TOPICS

Internet

Mobile & Wireless

Networking

The Internet Assigned Numbers Authority (IANA) has handed out its last IPv4 addresses, leaving the remaining blocks to regional registries that in some cases may exhaust them within a few months.

MORE LIKE TI

The 6 biggest misconceptior

Internet body may use up IPv
week

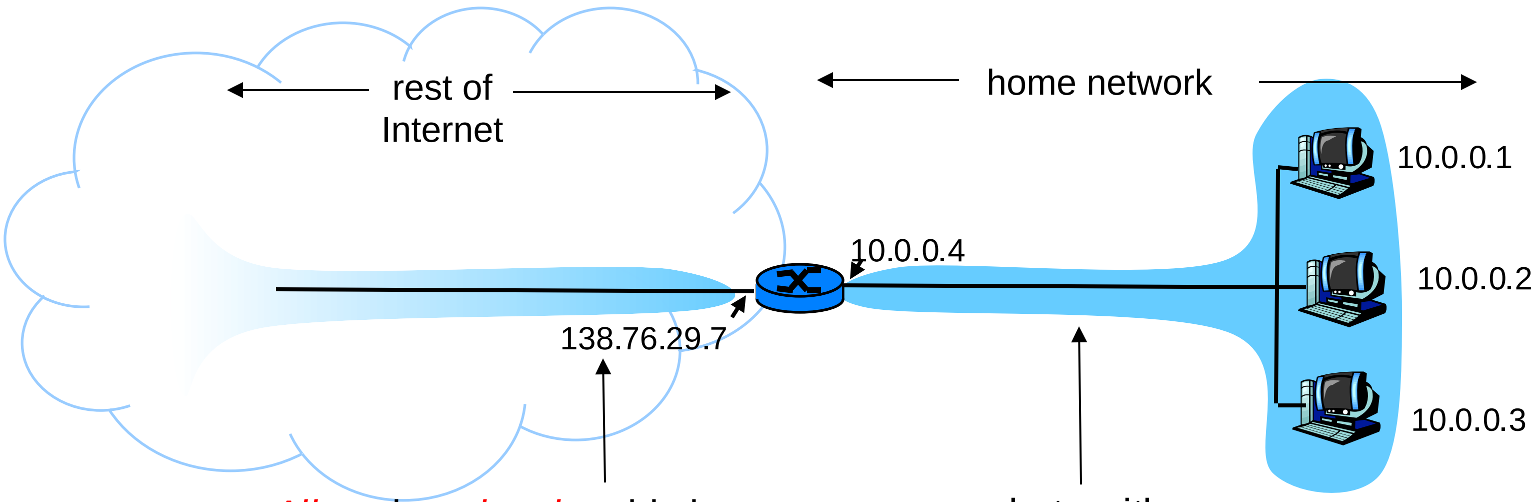
Address allocation kicks off II

The world did not breaking apart after ICANN ran out IPv4 addresses, by utilizing Network Address Translation (NAT)

# Network Address Translation (NAT)

- *(originally) considered* a “short-term” solution to the depletion of IP addresses
  - Longer-term solution: design & deploy IPv6
- Private address spaces:
  - 10.0.0.0-10.255.255.255 (10.0.0.0/8)
  - 172.16.0.0-172.**31**.255.255 (172.16.0.0/12)
  - 192.168.0.0-192.168.255.255 (192.168.0.0/16)
- Anyone can use these address blocks *internally* to hide a number of hosts behind a single *public IP address*
  - without requesting permission from anyone

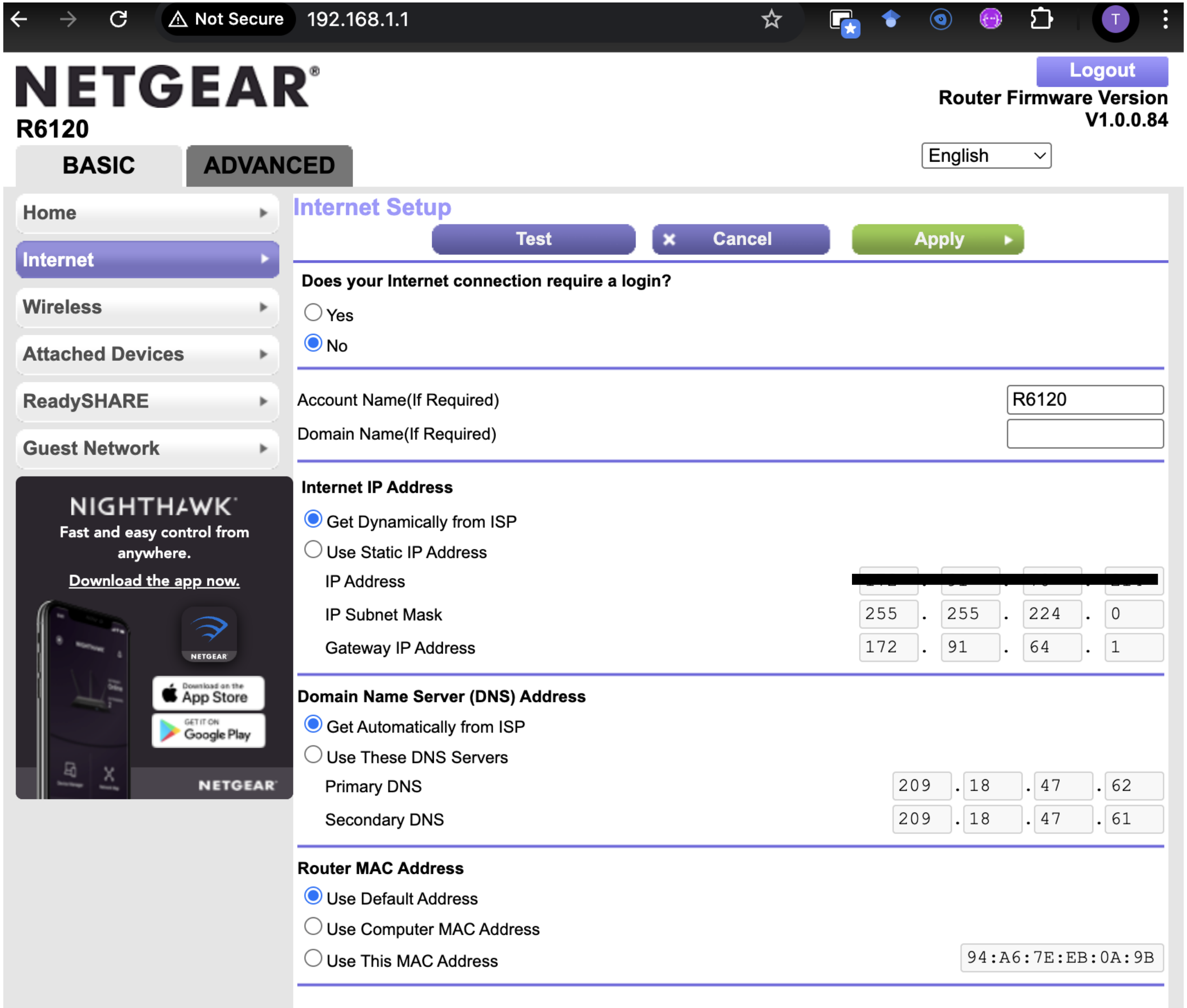
# Using Private Addresses at Home




*All* packets *leaving* this box must have the same source IP address 138.76.29.7

packets with source or destination in this network have 10.0.0.0/8 address for source, destination (as usual)

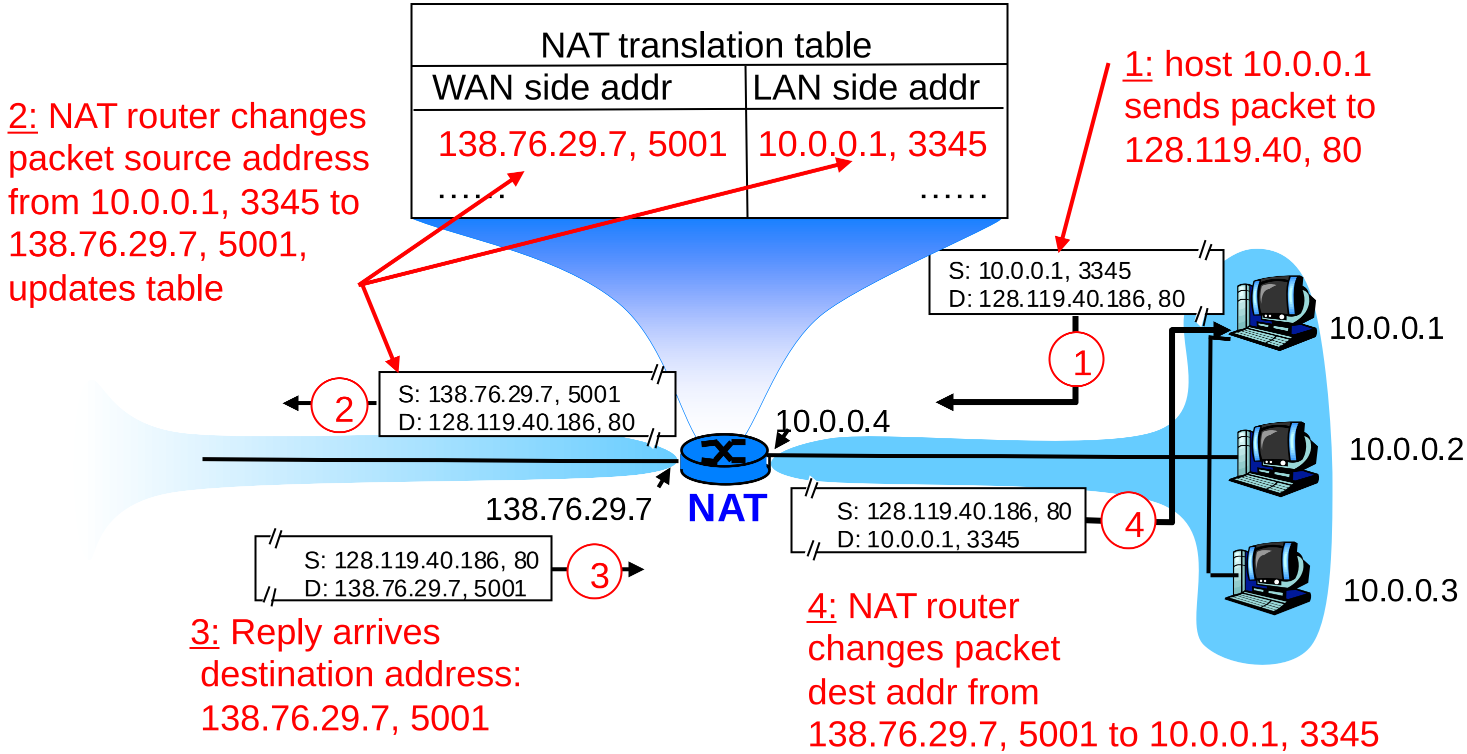
# Using Private Addresses at Home



# NAT: Network Address (and port) Translation

important

Would the NAT table overflow after running for a long time?




FYI:
- RFC 6056 says that the range for ephemeral ports should be 1024–65535.
- Internet Assigned Numbers Authority (IANA) and RFC6335 suggests the range 49152–65535

# NAT implementation

- *Outgoing* packets: replace (*source* IP address, source port #) of every outgoing packet to (NAT IP address, new port #)
  - . . . remote clients/servers will respond using (NAT IP address, new port #) as destination address
  - remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *Incoming* packets: replace (destination NAT IP address, destination port #) of every incoming packet with corresponding (source IP address, port #) stored in NAT table
- Delete NAT table entries that have not been used for *some time*
  - need to do this due to limited available port numbers
  - "some time" not defined until late, with big consequences

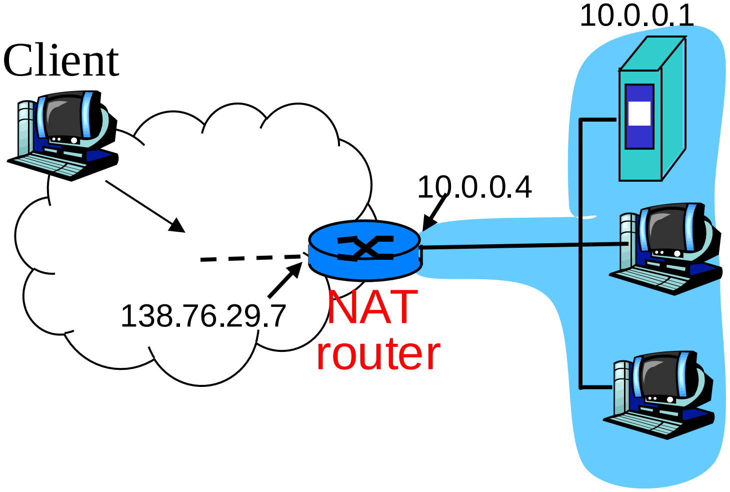
When running out available port numbers: wont be able to accommodate new transport sessions

# Problems due to NAT

- Increased complexity (e.g. router has to keep the NAT table)
- Single point of failure
  - also limited scalability due to port# and NAT address block limitations
- Cannot run services inside a NAT box
  - All application designs have to worry about NAT traversal problem

An example

- client wants to connect to the server with address 10.0.0.1
- server address 10.0.0.1 is only visible within the LAN
- The only externally visible address for all internal hosts: 138.76.29.7



# NAT Traversal from outside to inside?

**Solution 1**: statically configure NAT to forward incoming connection requests at <u>a given port</u> to server, e.g.

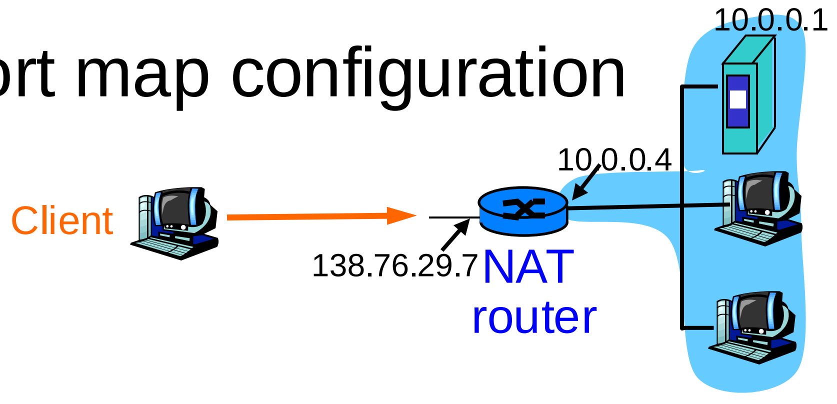
- (138.76.29.7, <u>port 2500</u>) always forwarded to 10.0.0.1 port 2500

**Solution 2**: Universal Plug and Play Protocol (UPnP)

Allow host behind a NAT to:

- learn the public IP address (138.76.29.7)
- add/remove port mappings dynamically (with lease times)

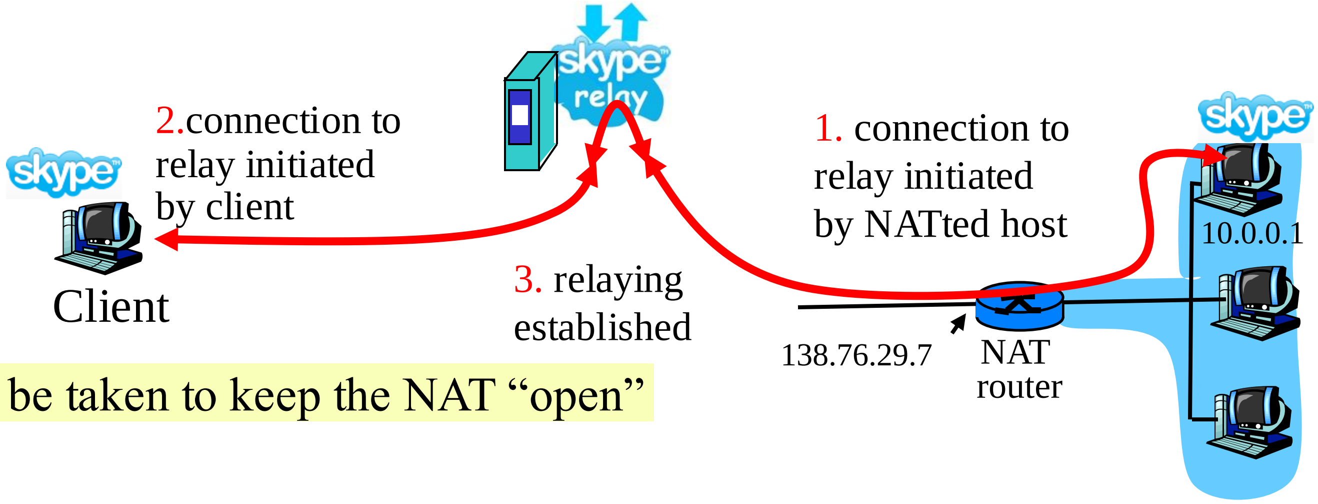
i.e., automate static NAT port map configuration



# NAT Traversal (II)

**Solution** 3: relaying (e.g. used in Skype)

- NATed client app establishes connection to relay
- External client connects to relay
- relay bridges packets between two end points
  - to let 2 clients communicate directly: the Skype relay needs to inform them the other's public IP address and port# being used

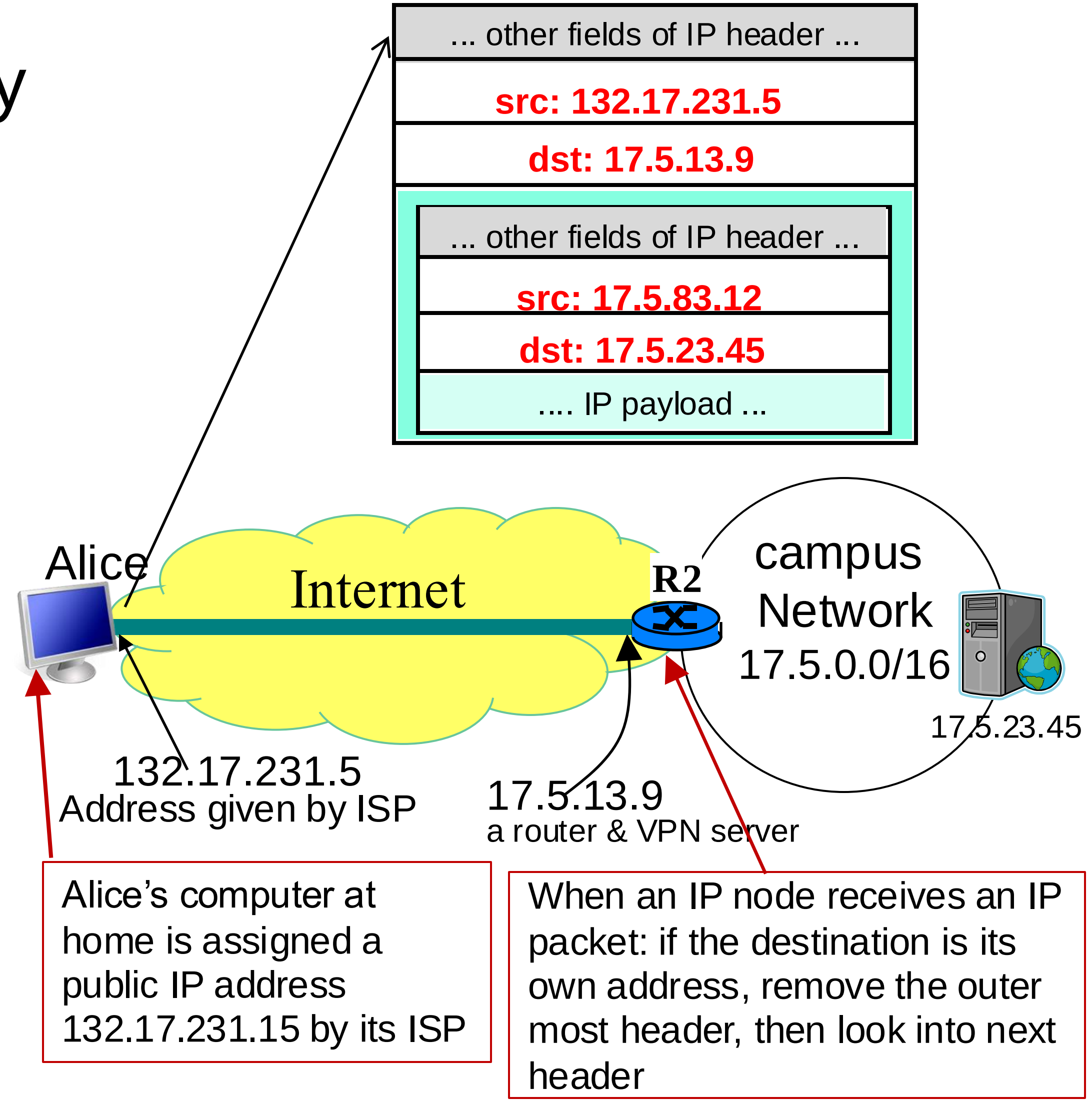



Care must be taken to keep the NAT "open" important

# IP-in-IP: IP tunneling

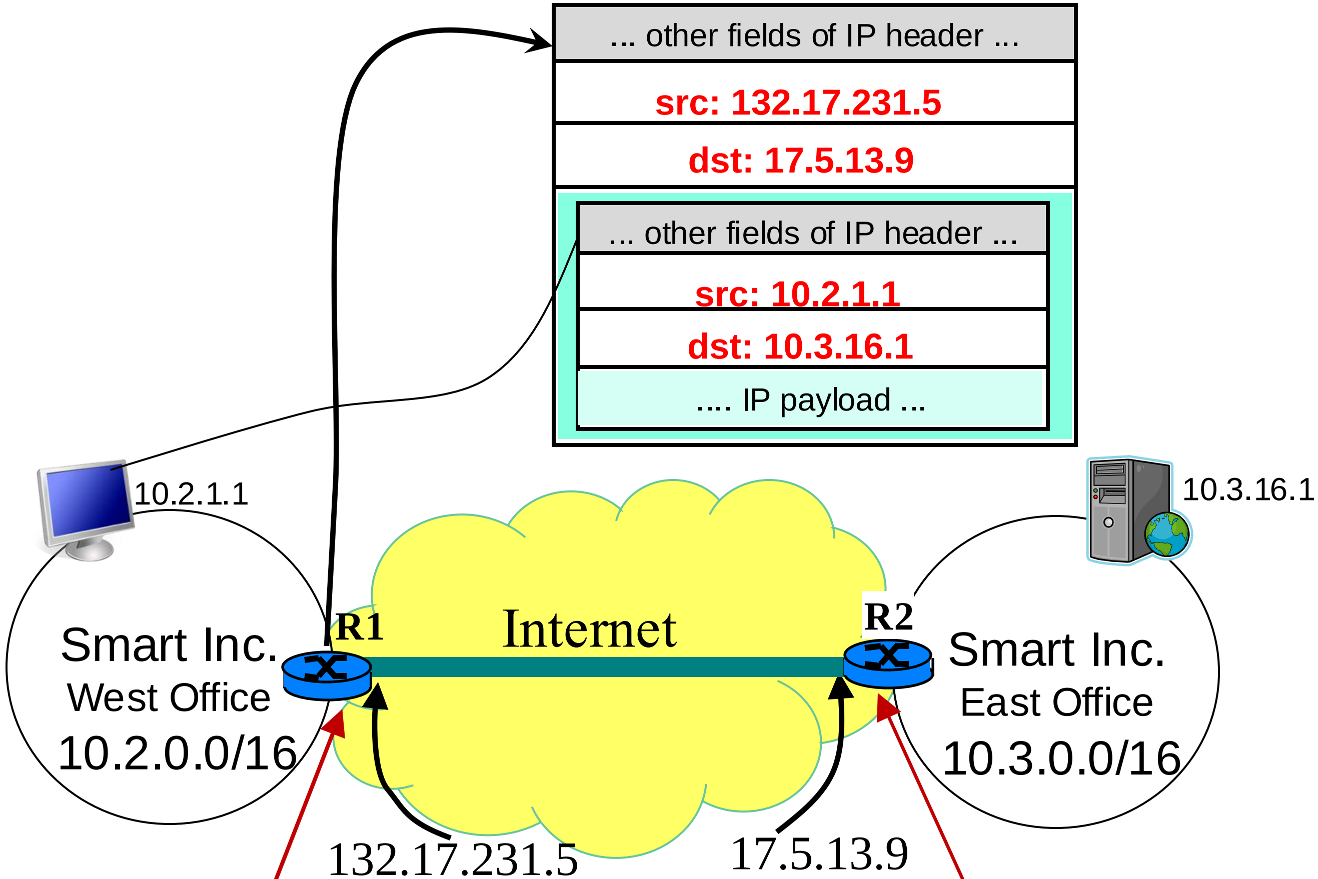
Example:

- the campus server only allows access by computers on-campus
  - i.e. host addresses within the range of 17.5.0.0-17.5.255.255
- Alice can gain access at home using IP-in-IP tunneling
  - Obtain a campus address 17.5.83.12
  - Encapsulate packets with the ISP address as source



# Connecting private network via *IP tunneling*

important




This NAT router knows the Smart Inc. East Office's (private) address block and its NAT router's public IP address (the same is true for R2)

When an IP node receives an IP packet and see the destination is its own address, it removes the outer most header and look into next header

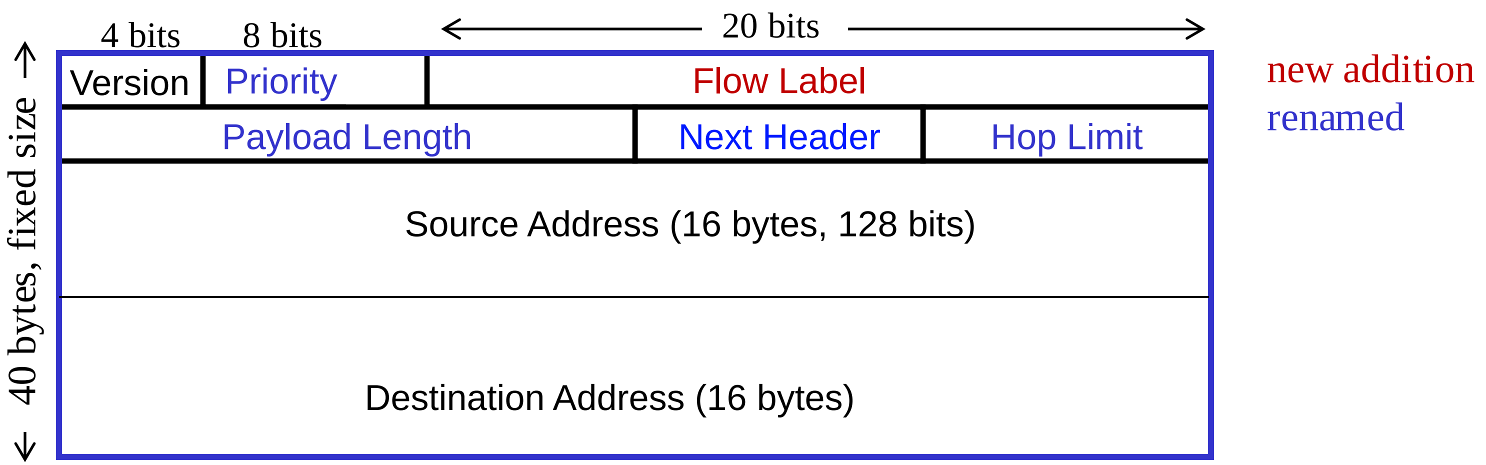
# IPv6: the *planned* long term solution

- Originally motivated by IPv4 address space exhaustion
- *Also taking the opportunity for some clean-up*
- IPv6 packet format:
  - Fixed-length 40byte header, length field excludes header
    - Address length: 32 bits → 128 bits

$2^{96}$ bigger address space

  - Moved fragmentation & IP options *out of* the base header
  - *Eliminated* header checksum
  - Type of Service → Traffic Class
  - TTL → Hop Limit, Protocol → Next Header
  - *Added* Flow Label field

# IPv6 header format




**Changes from IPv4:**

**Priority**: *usage yet to be finalized*

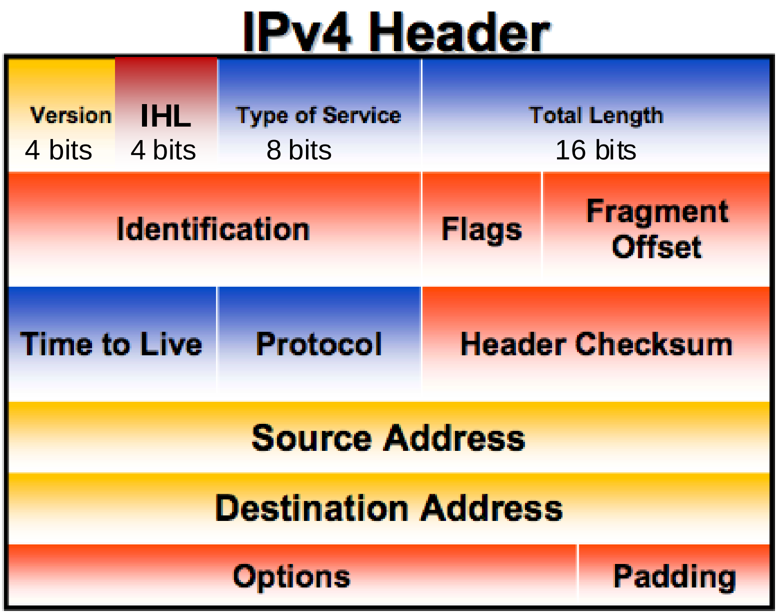
**Flow Label**: identify packets in same "flow" (but flow is yet to be defined)

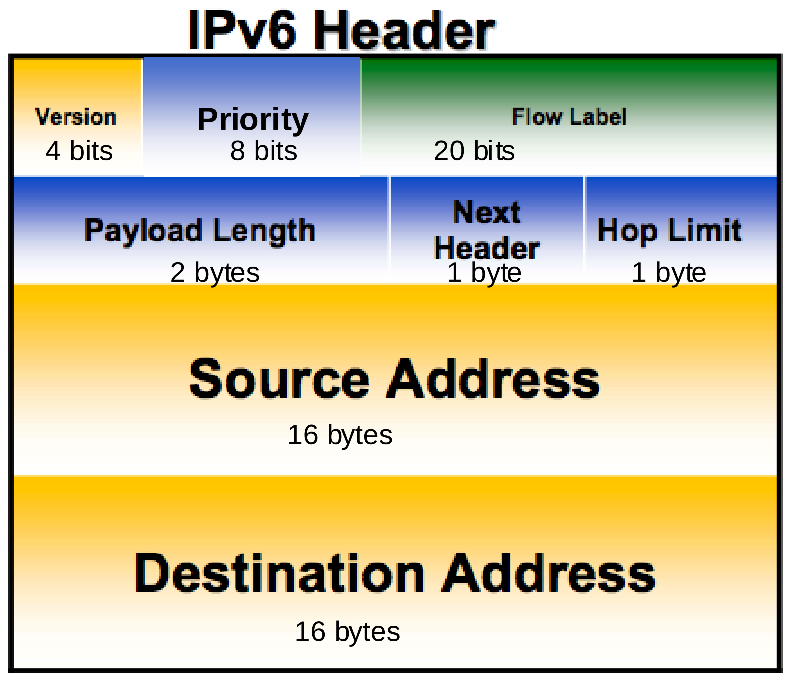
**Next header**: identify upper layer protocol for data

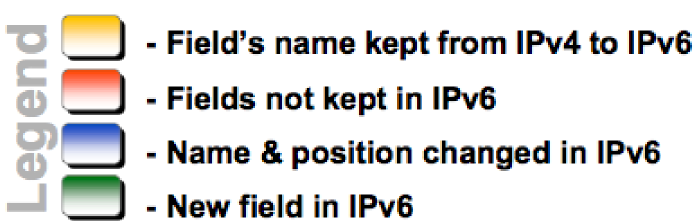
Options: outside of the basic header, indicated by "Next Header" field

Header Checksum: removed

# IPv4 : IPv6 Header Comparison




(IPv6 address size in the above figure is out of proportion)

Fragmentation: performed end-to-end

Options: outside the basic header (fixed size)

- indicated by "Next Header" field
- no need for header length any more

Checksum: removed

# IPv6 extension headers

FYI

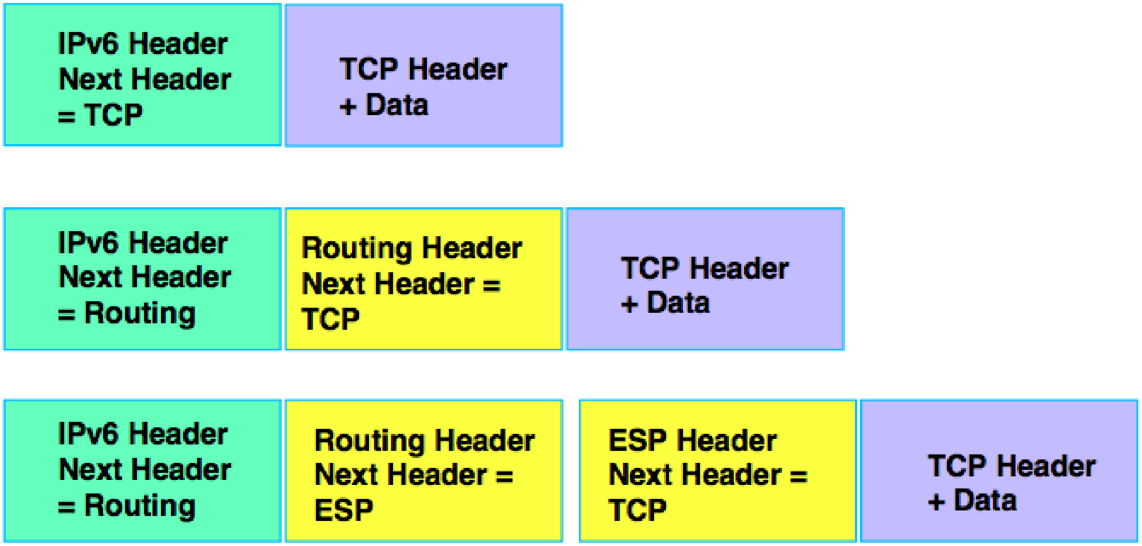
The following are IPv6 extension header types

- Routing: Loose or tight source routing
- Fragmentation: only source can fragment
- Authentication
- Hop-by-Hop Options
- There are several more
  - Most extension headers are examined only at destination

# Encoding options in IPv6 extension header

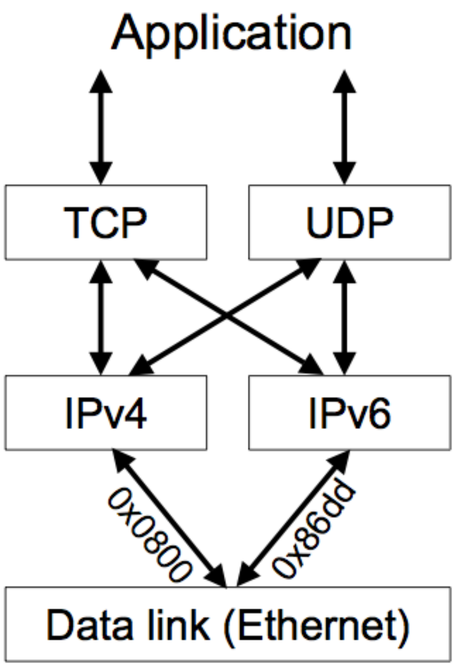
FYI

- Basic header: fixed length
- "next header" field specifies how long it may be
- Daisy chained



# Transition from IPv4 to IPv6 in host

- Hard fact: not all routers on the Internet can be upgraded to IPv6 *simultaneous*
  - or even ever ...
- Must operate the Internet with mixed IPv4 and IPv6 nodes



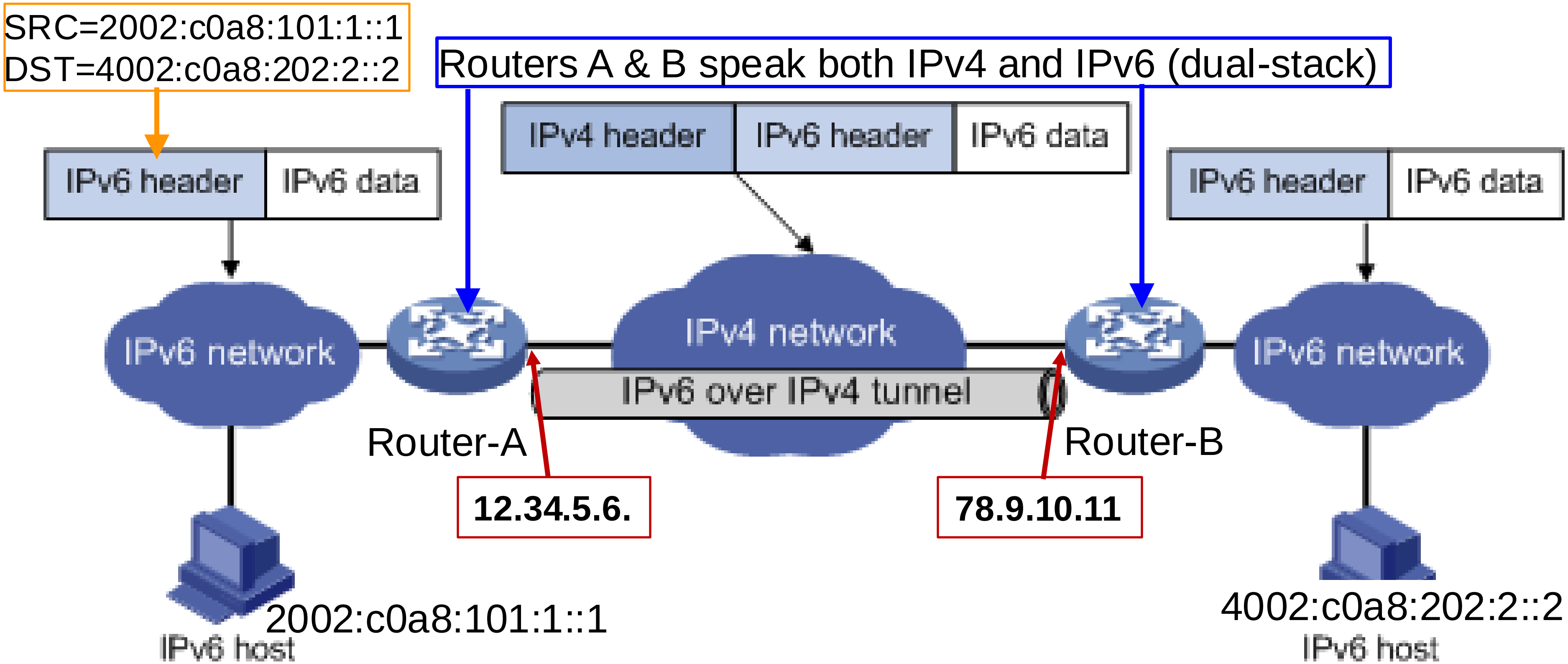

**Proposed Solution**: Dual stack

- Assumption: duel-staked nodes have *both* IPv4 & IPv6 addresses
- Roadblock: doesn't solve the IPv4 address shortage problem

**Next solution**: interconnecting IPv6 hosts by IPv4 networks

# Transition IPv4 → IPv6 in network: tunneling

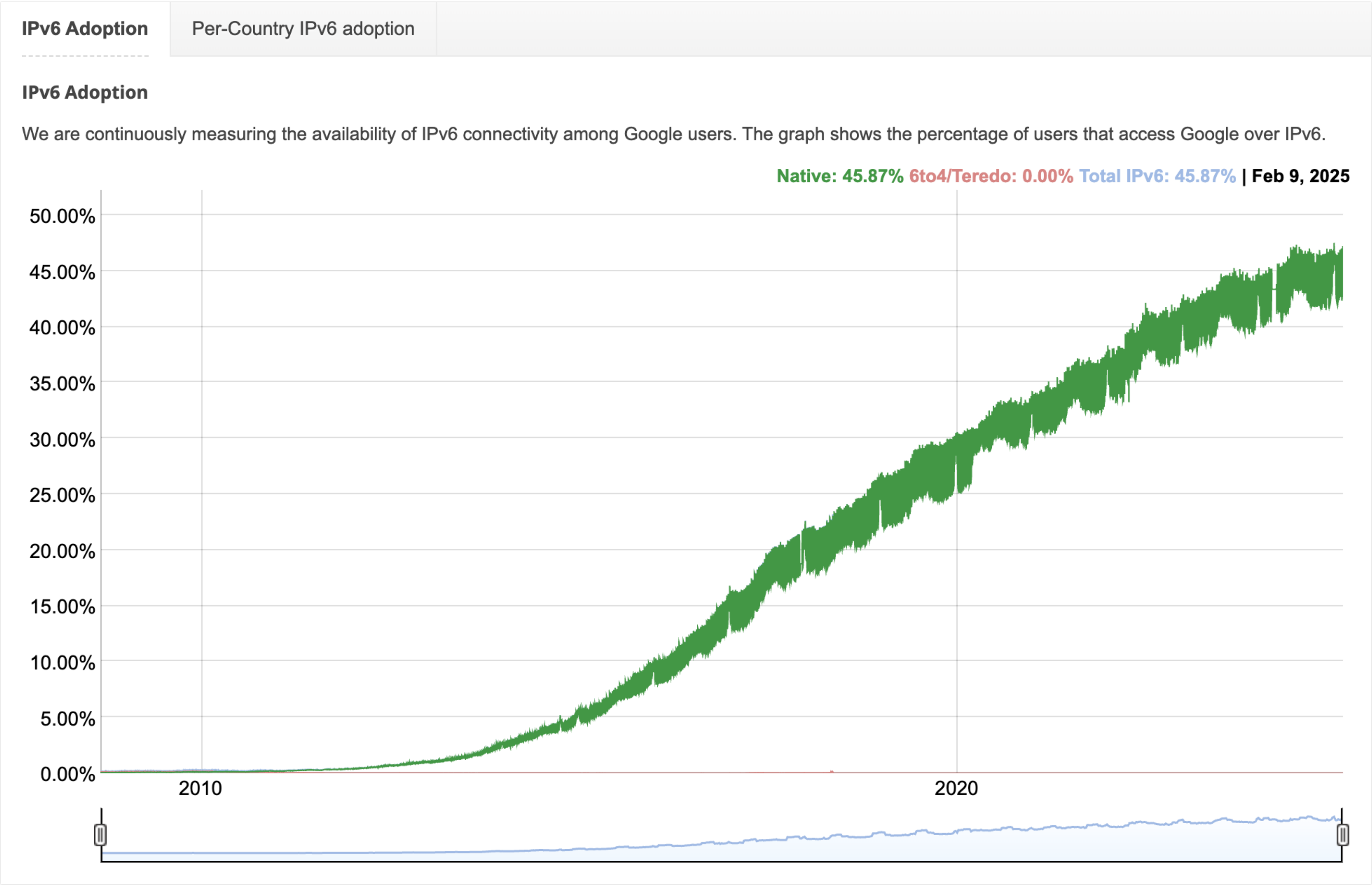
## IPv6 packet through IPv4 network




- Routes A & B keep both IPv4 and IPv6 FIB
- When the above IPv6 packet arrives, A looks up IPv6 FIB: point to an IPv4 next hop address (78.9.10.11)
  - Encapsulate the IPv6 packet with IPv4 header
- Router B decapsulates, forward the original IPv6 packet to destination network

# Data from Google

https://www.google.com/intl/en/ipv6/statistics.html

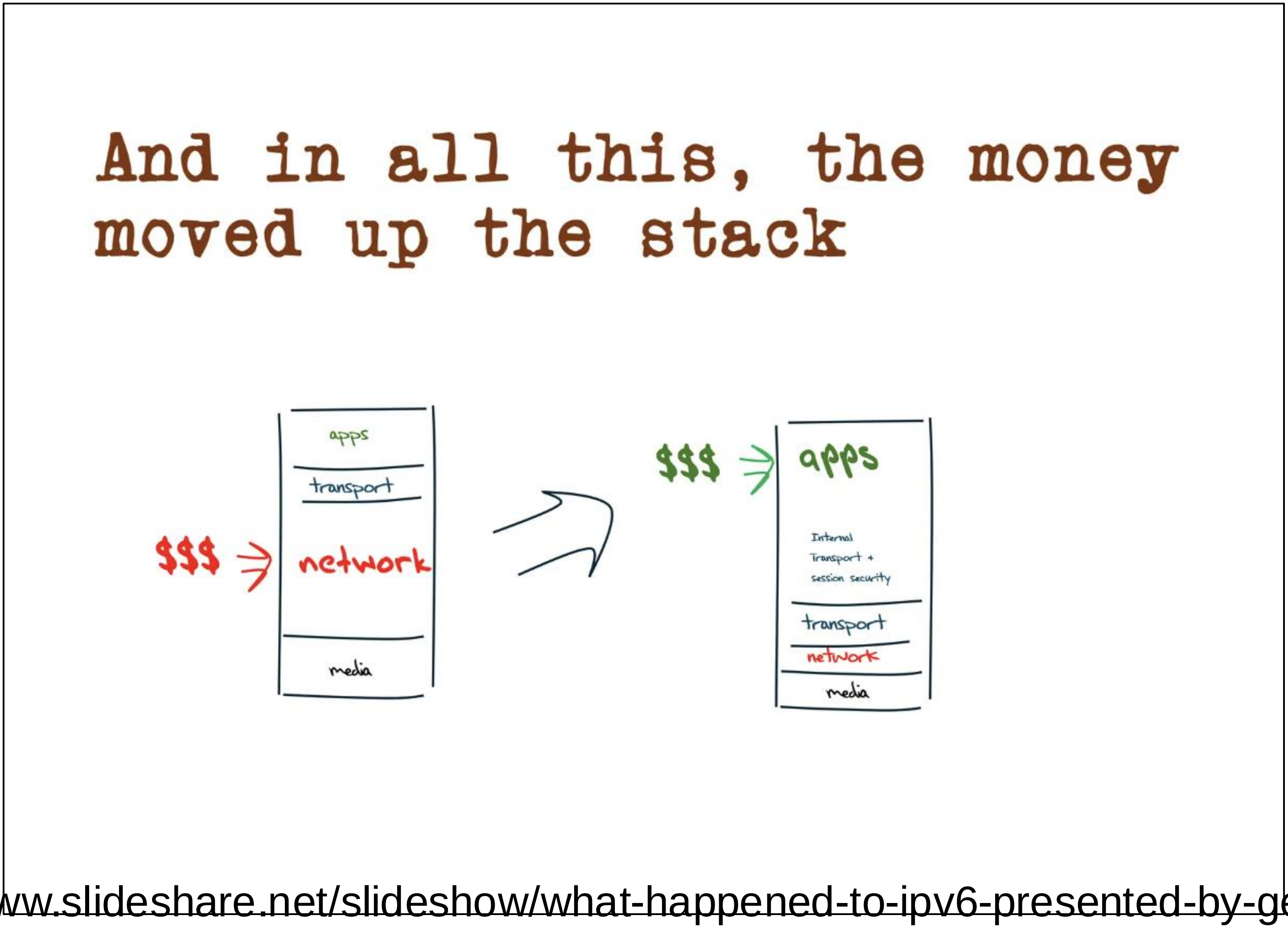



data bias: based on Google user base

# Why

Pure FYI

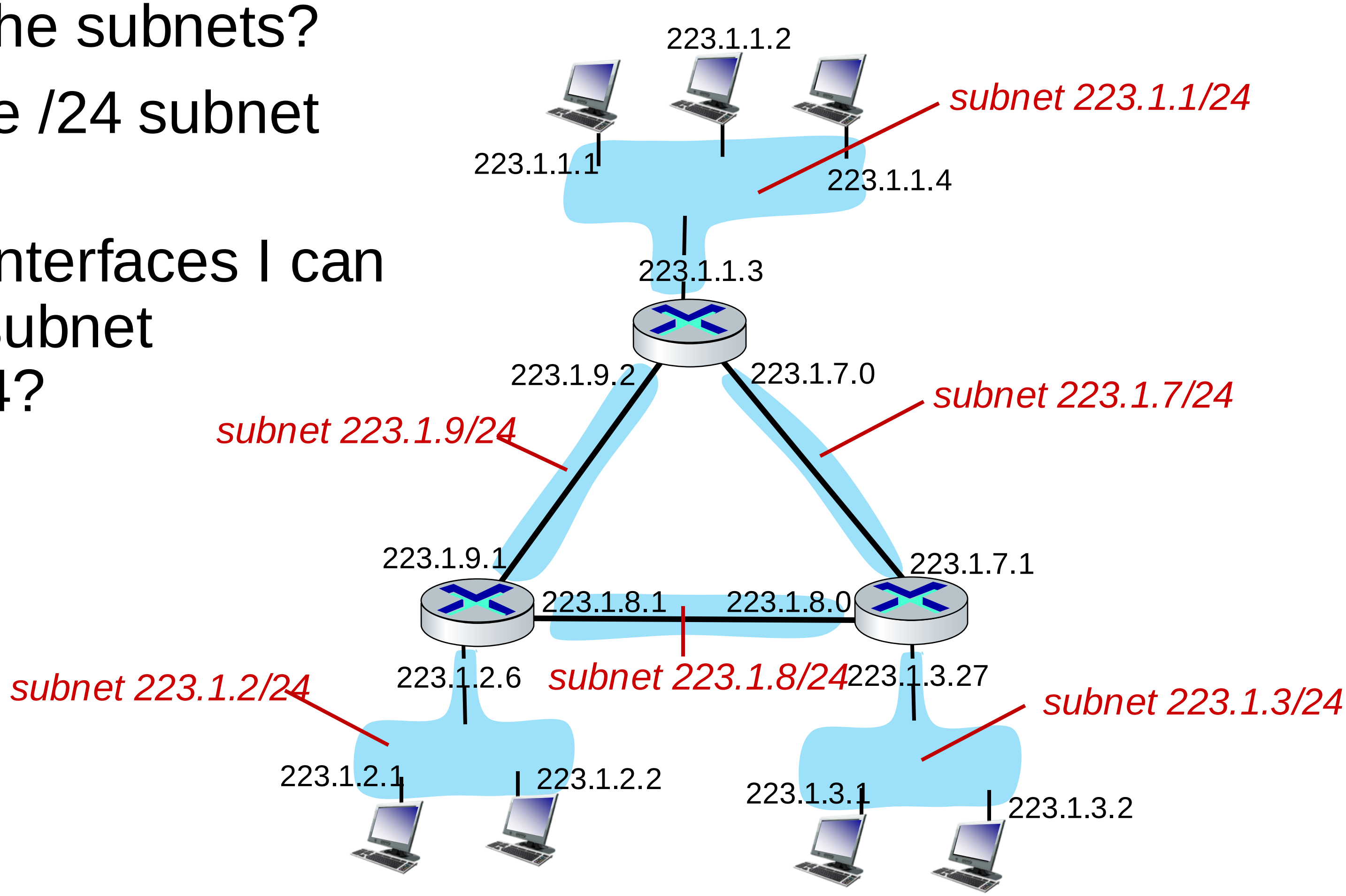
NANOG 92 Keynote: Whatever Happened to IPv6




https://www.slideshare.net/slideshow/what-happened-to-ipv6-presented-by-geoff-huston/273318949#1

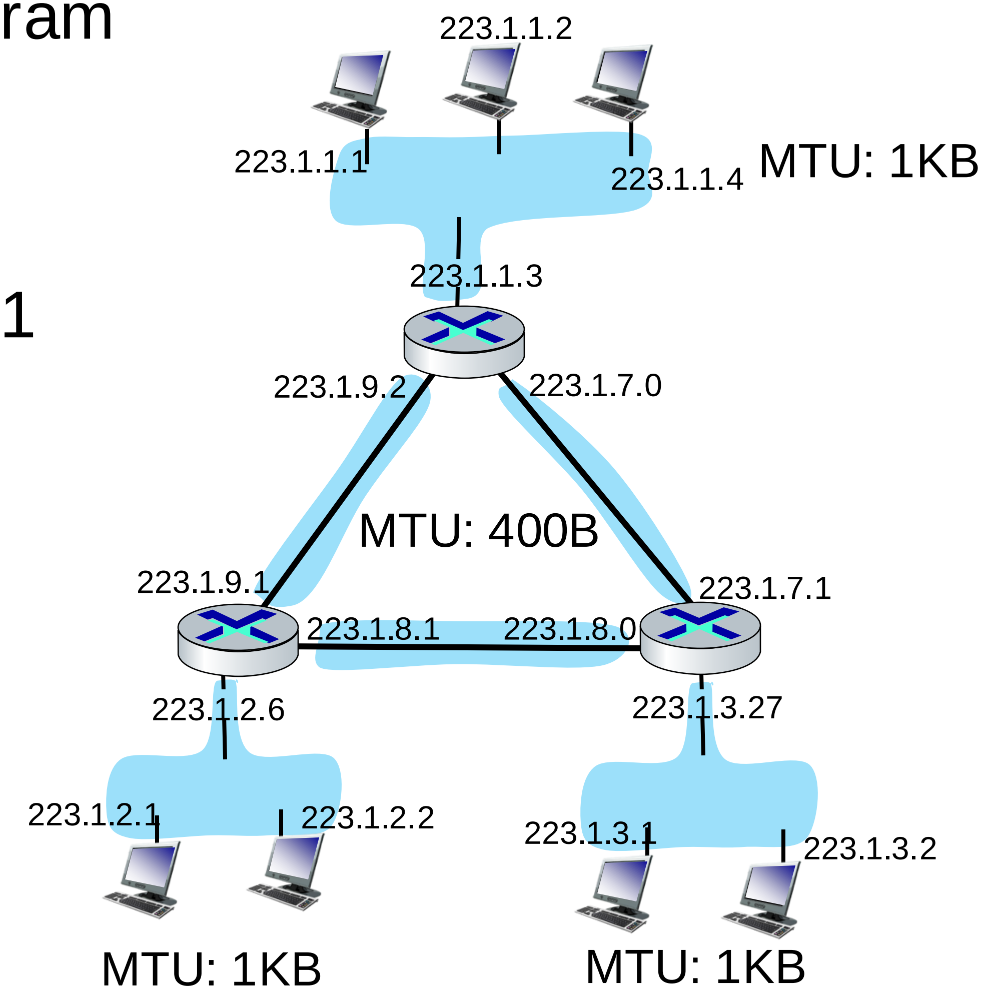
# Practice Question 1: subnets

- Where are the subnets?
- What are the /24 subnet addresses?
- How many interfaces I can address in subnet 223.1.2.1/24?



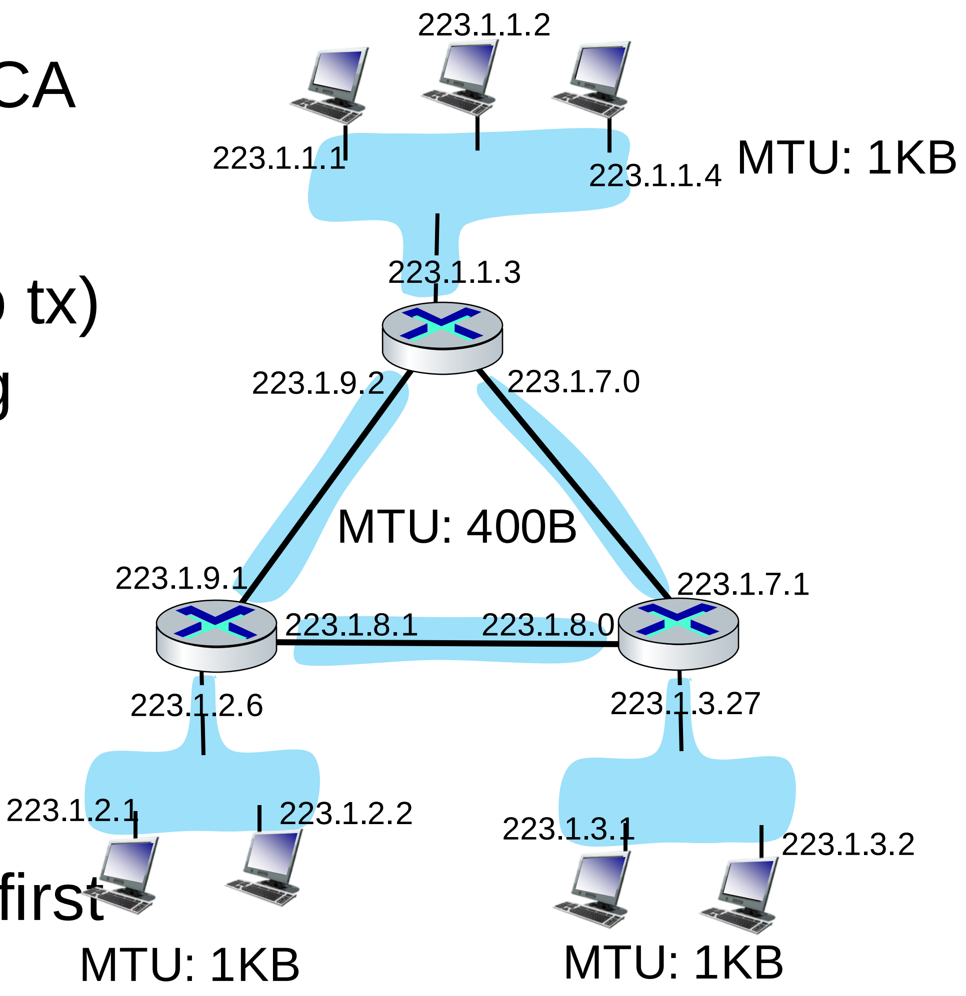
# Practice Question 2: fragmentation

- Assuming an IP datagram total size of 1KB
  - 223.1.2.1 to 223.1.1.1
- How many datagram fragments will 223.1.1.1 see?
- Try write the fragment details?



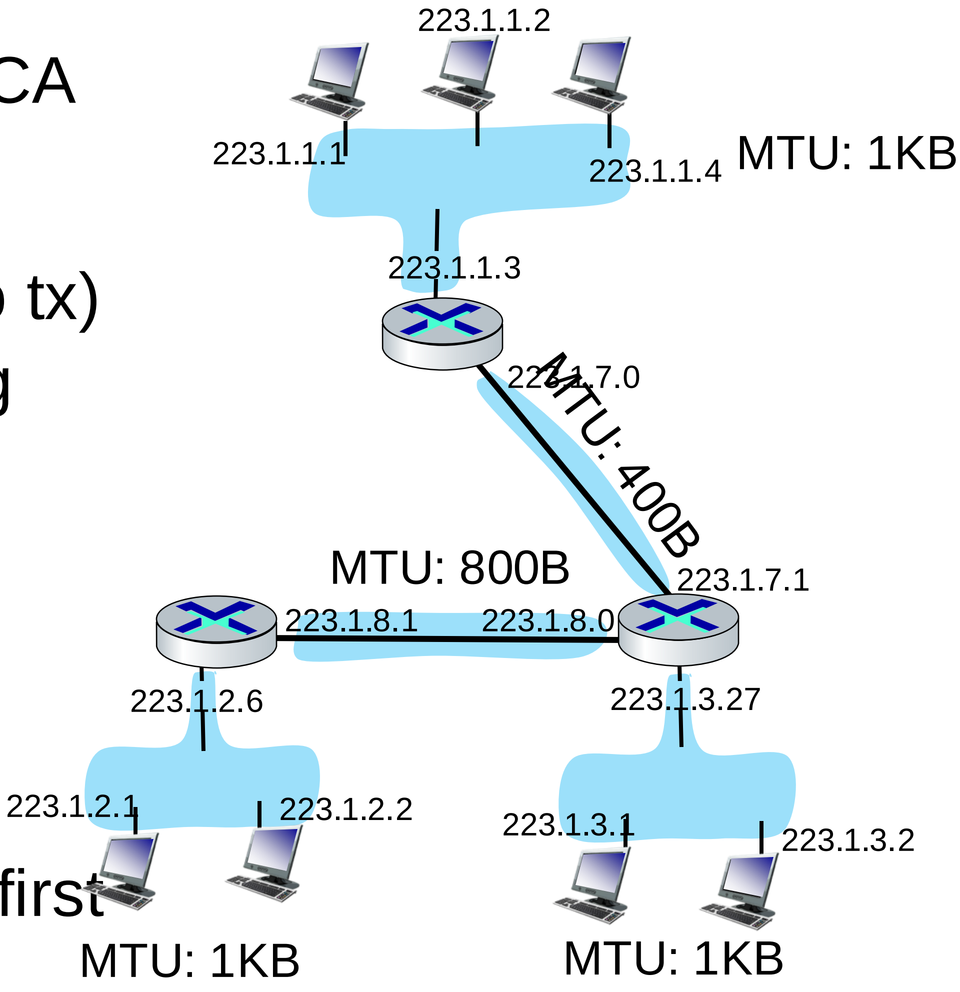
# Practice Question 3: coupled

- (Assuming TCP connection is in stable CA stage, cwnd = 4, each segment carries 960B payload, always data to tx)
- Consider 4 outstanding TCP segments
  - Src: 223.1.2.1: 80
  - Dst: 223.1.1.1: 1234
- The first segment lost, how many fragments 223.1.1.1 will receive before sender gets the first seg acked?



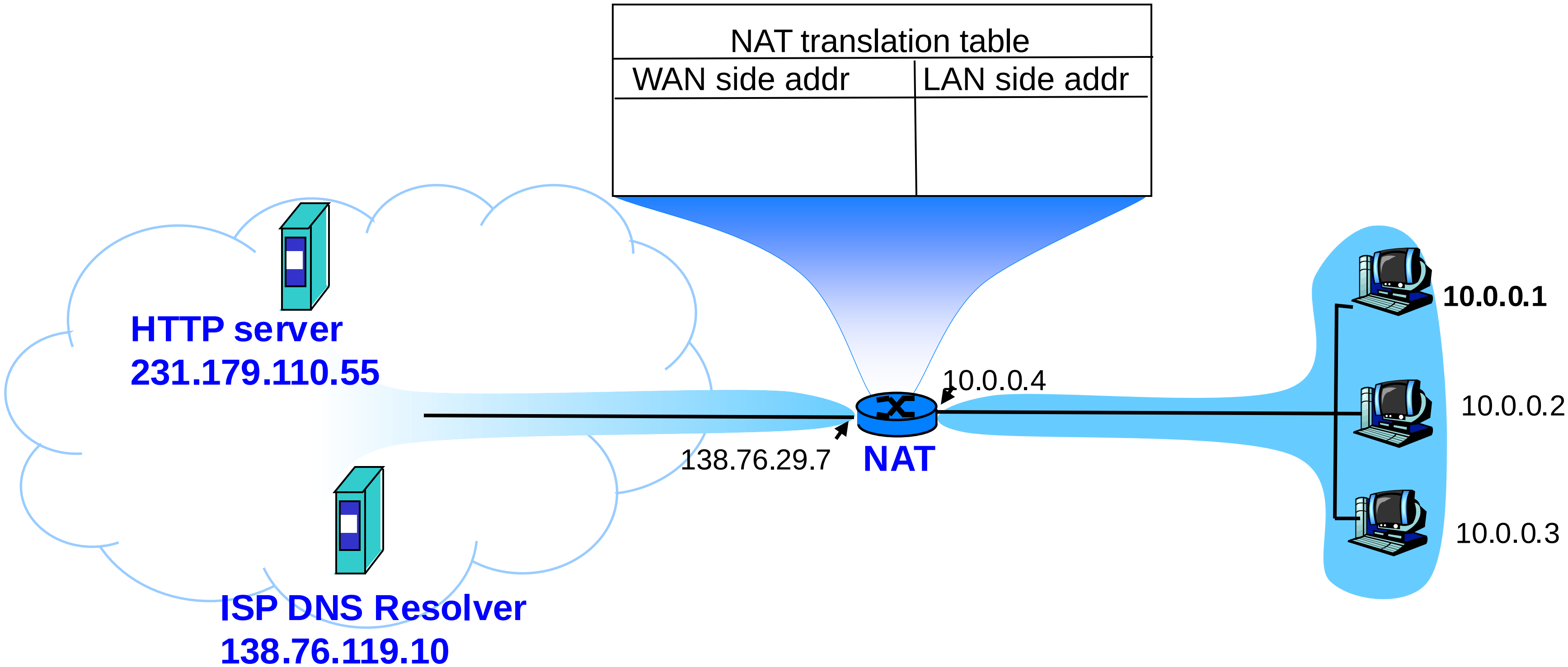
# Practice Question 4

- (Assuming TCP connection is in stable CA stage, cwnd = 4, each segment carries 960B payload, always data to tx)
- Consider 4 outstanding TCP segments
  - Src: 223.1.2.1: 80
  - Dst: 223.1.1.1: 1234
- The first segment lost, how many fragments 223.1.1.1 will receive before sender gets the first seg acked?



# Practice Question 5

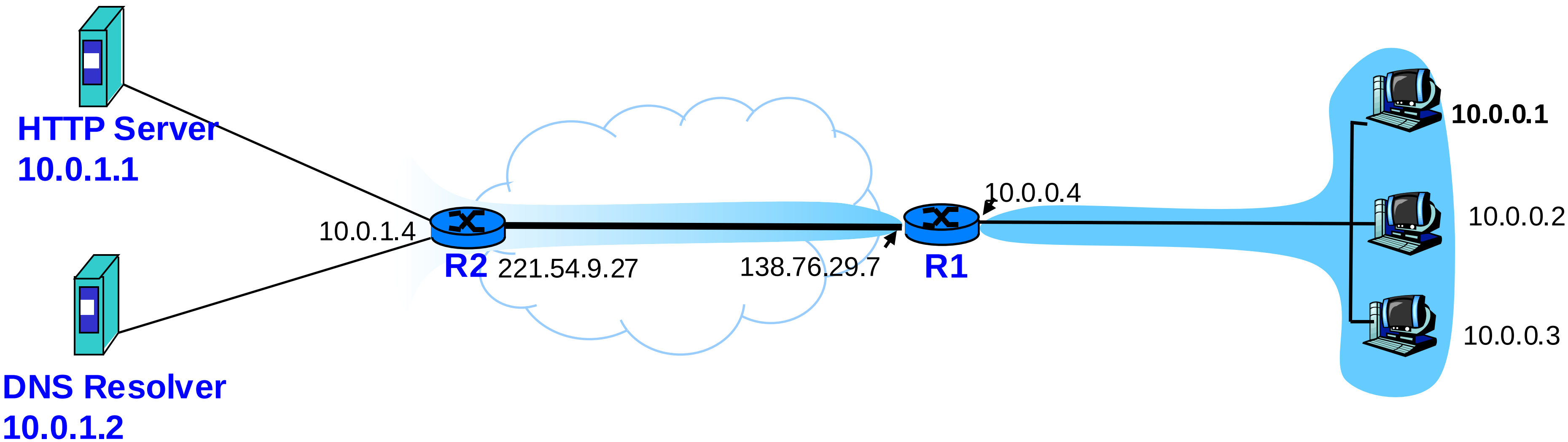
- NAT table after the host gets the page?




DNS runs on port 53, HTTP runs on port 80

# Practice Question 6

- Assuming there is magic to discover the bottleneck MTU
  - End host sends with the “just right” size to avoid fragmentation
  - FYI the magic exists and is called Path MTU Discovery
- Now you know the bottleneck is on the public Internet
  - Bottleneck MTU = 1420B
  - What is the “right” TCP Maximum Segment Size (i.e., payload)?




DNS runs on port 53, HTTP runs on port 80, no option fields in all headers