

# Chapter 4

## Network Layer:

## Data Plane

A note on the use of these PowerPoint slides:

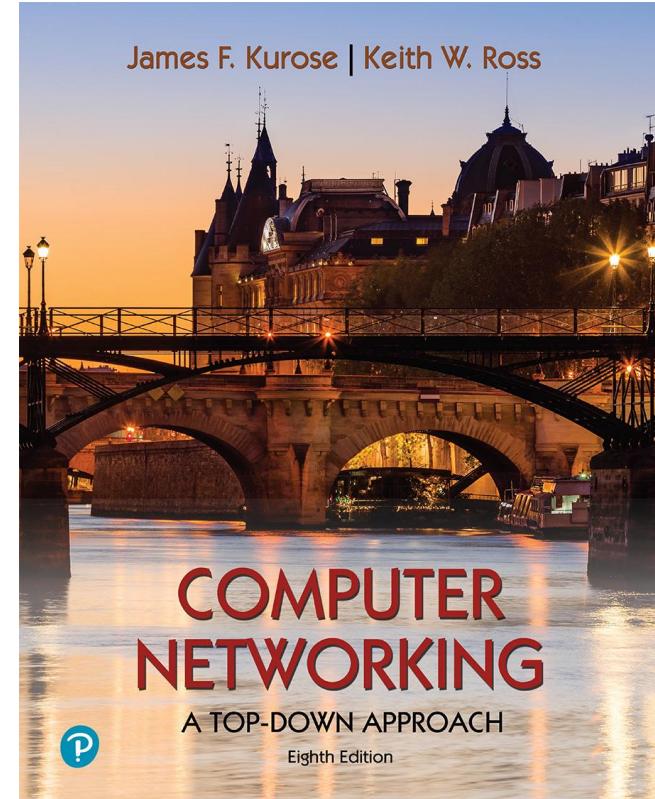
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020  
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A  
Top-Down Approach*  
8<sup>th</sup> edition  
Jim Kurose, Keith Ross  
Pearson, 2020

# Network layer: our goals

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - addressing
  - generalized forwarding
  - Internet architecture
- instantiation, implementation in the Internet
  - IP protocol
  - NAT, middleboxes

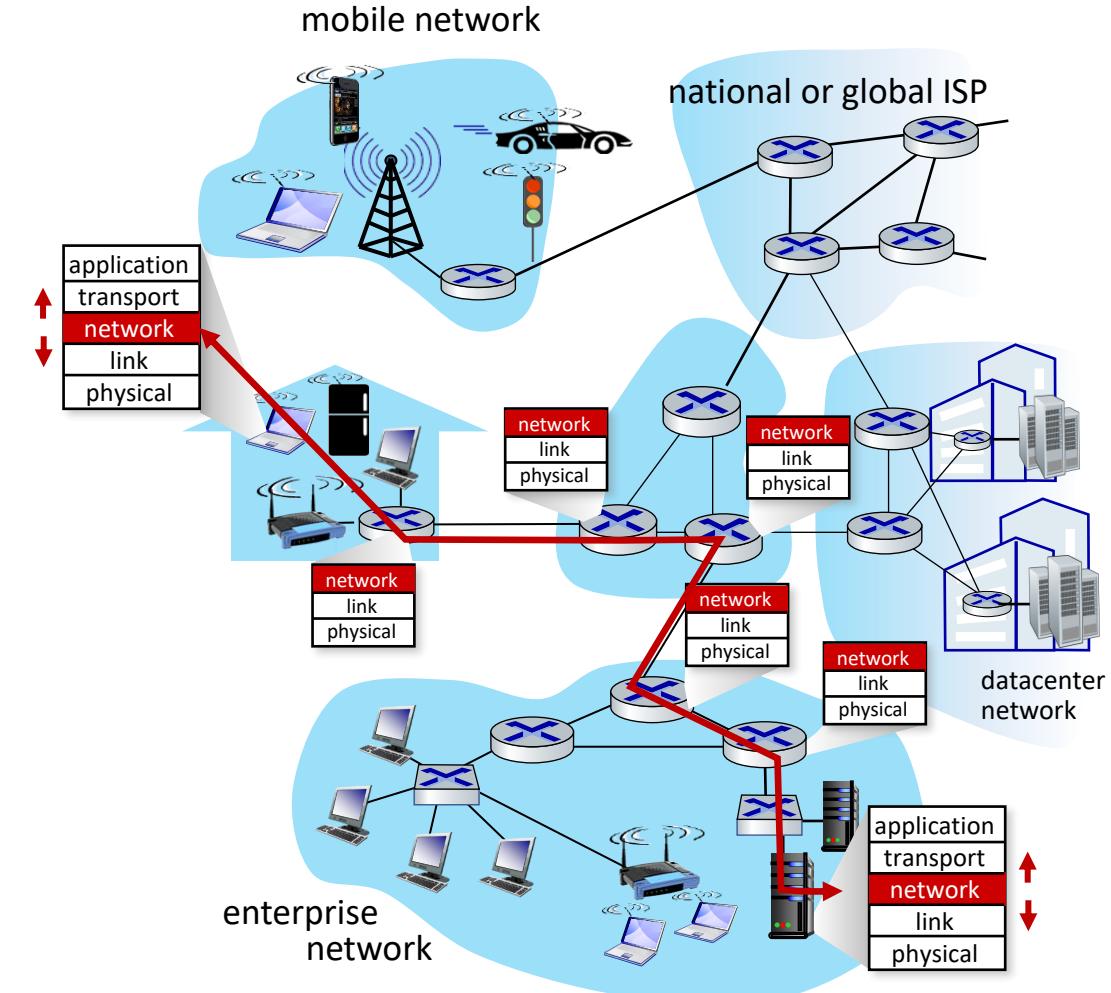
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes



# Network-layer services and protocols

- transport segment from sending to receiving host
  - **sender**: encapsulates segments into datagrams, passes to link layer
  - **receiver**: delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- **routers**:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path



# Two key network-layer functions

## network-layer functions:

- *forwarding*: move packets from a router's input link to appropriate router output link
- *routing*: determine route taken by packets from source to destination
  - *routing algorithms*

## analogy: taking a trip

- *forwarding*: process of getting through single interchange
- *routing*: process of planning trip from source to destination



forwarding

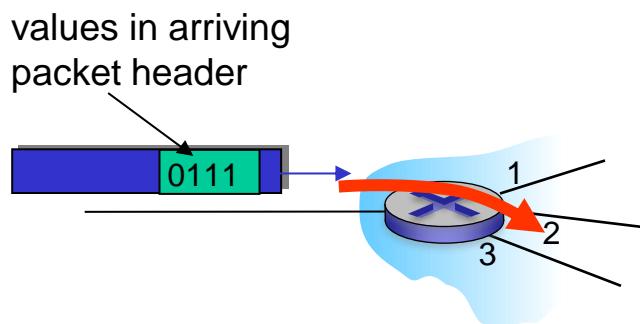


routing

# Network layer: data plane, control plane

## Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

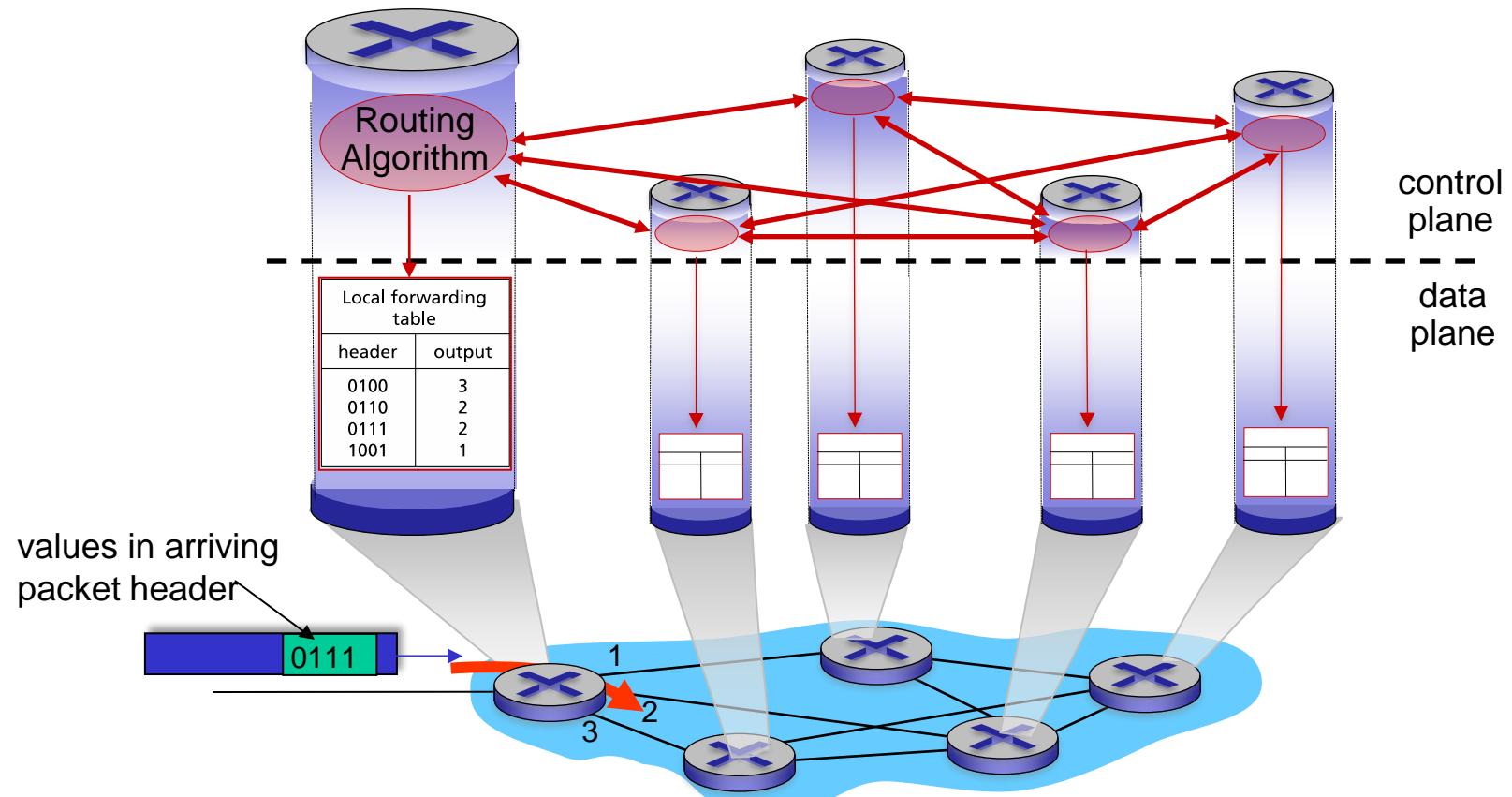


## Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms*: implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

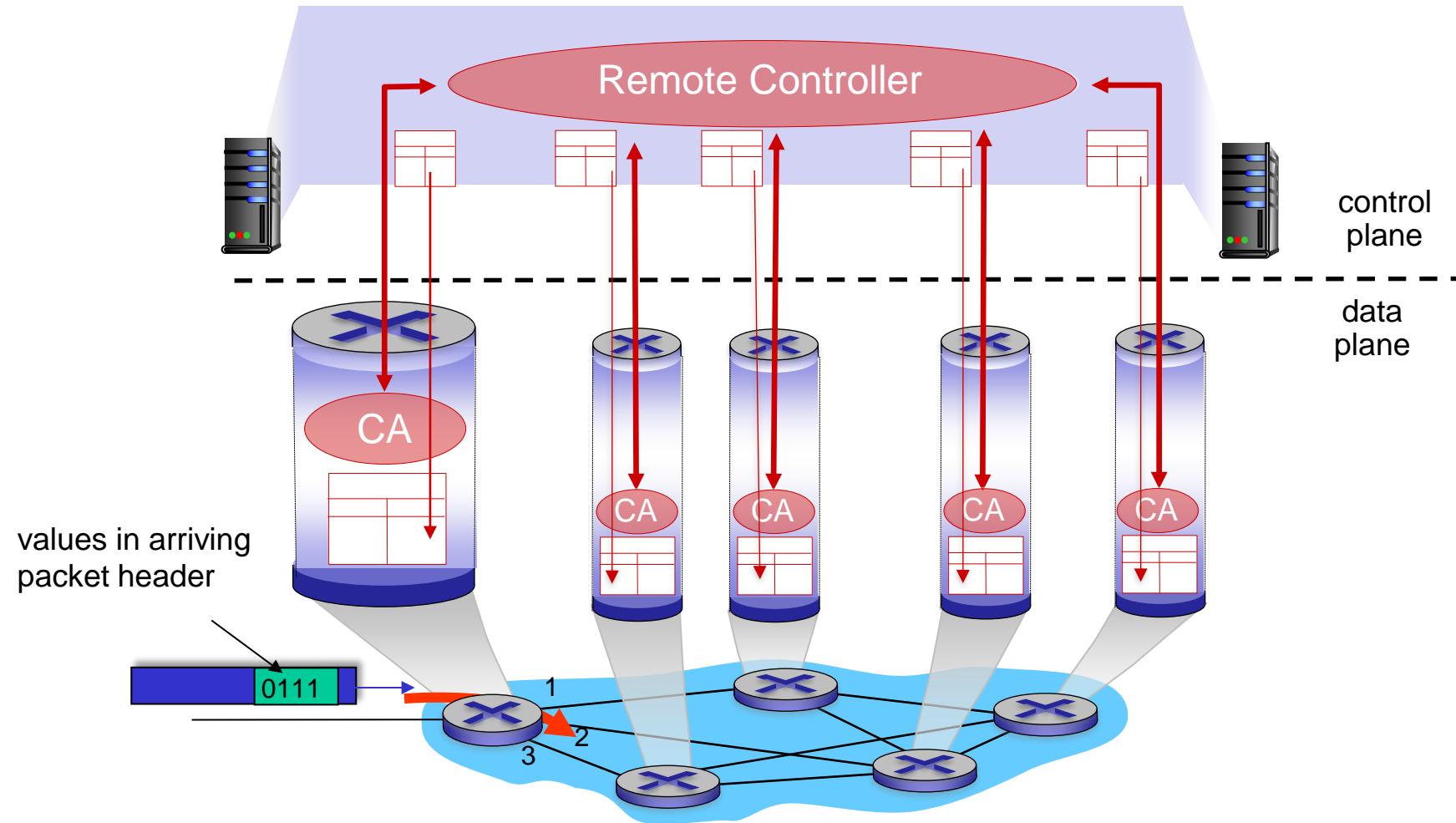
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Network service model

*Q:* What *service model* for “channel” transporting datagrams from sender to receiver?

example services for  
*individual* datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a *flow* of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

# Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no

Internet “best effort” service model

*No* guarantees on:

- i. successful datagram delivery to destination
- ii. timing or order of delivery
- iii. bandwidth available to end-end flow

# Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

# Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be “good enough” for “most of the time”
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients’ networks, allow services to be provided from multiple locations
- congestion control of “elastic” services helps

*It's hard to argue with success of best-effort service model*

# Destination-based forwarding

<i>forwarding table</i>	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	0
11001000 00010111 00010000 00000100 through	3
11001000 00010111 00010000 00000111	
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through	2
11001000 00010111 00011111 11111111	
otherwise	3

*Q:* but what happens if ranges don't divide up so nicely?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?  
11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*****	0
11001000 00010111 00011000 *****	1
11001000 1 00011*** *****	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range					Link interface
11001000	00010111	00010***	*****	*	0
11001000	00010111	00011000	*****	*	1
11001000	00010111	00011***	*****	*	2
otherwise					3

match!

examples:

11001000	00010111	00010110	10100001	which interface?
11001000	00010111	00011000	10101010	which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range					Link interface
11001000	00010111	00010***	*****	*	0
11001000	00010111	00011000	*****	*	1
11001000	00010111	00011***	*****	*	2
otherwise					3

match!

examples:

11001000	00010111	00010110	10100001	which interface?
11001000	00010111	00011000	10101010	which interface?

# Longest prefix matching

- we'll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable*: present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: ~1M routing table entries in TCAM

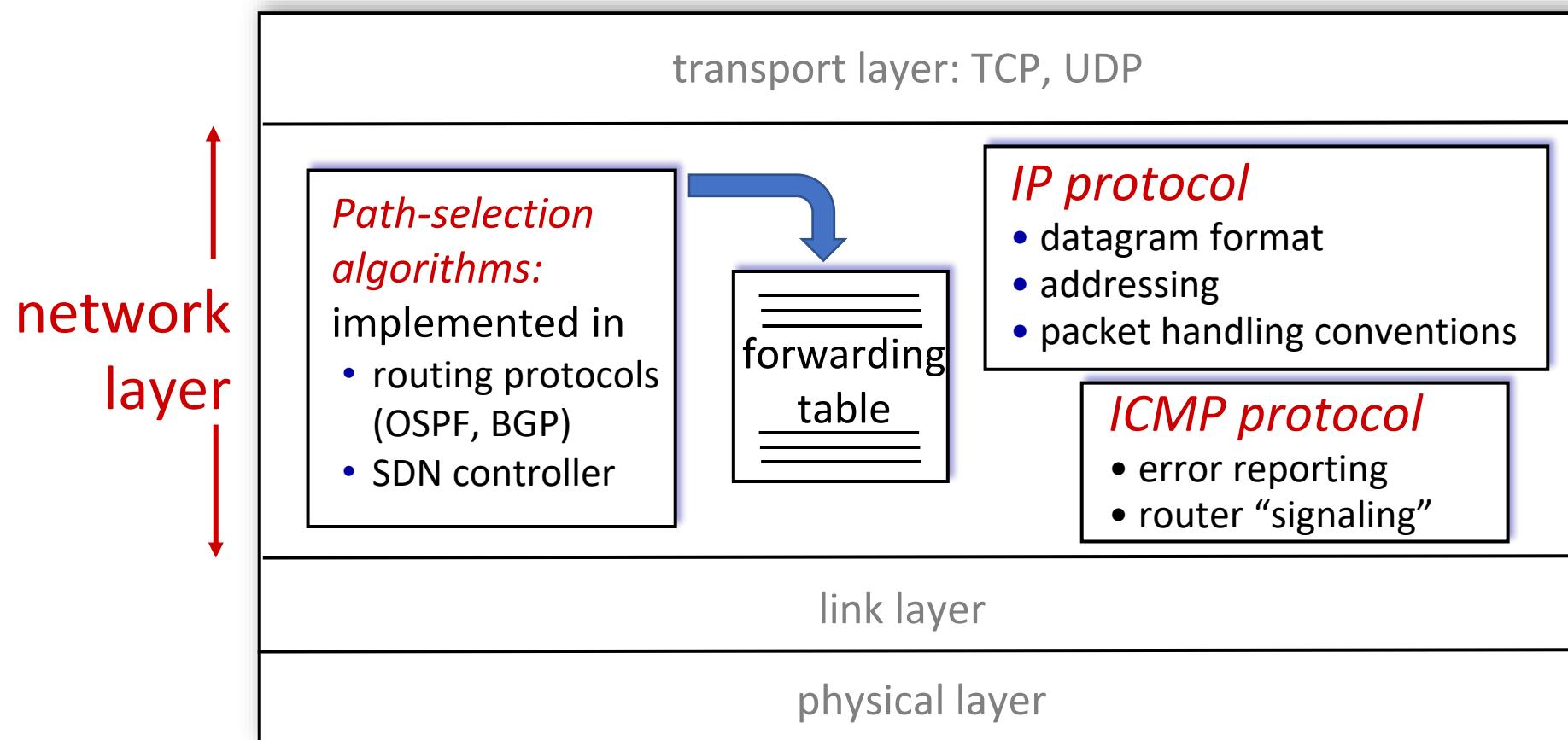
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - match+action
  - OpenFlow: match+action in action
- Middleboxes

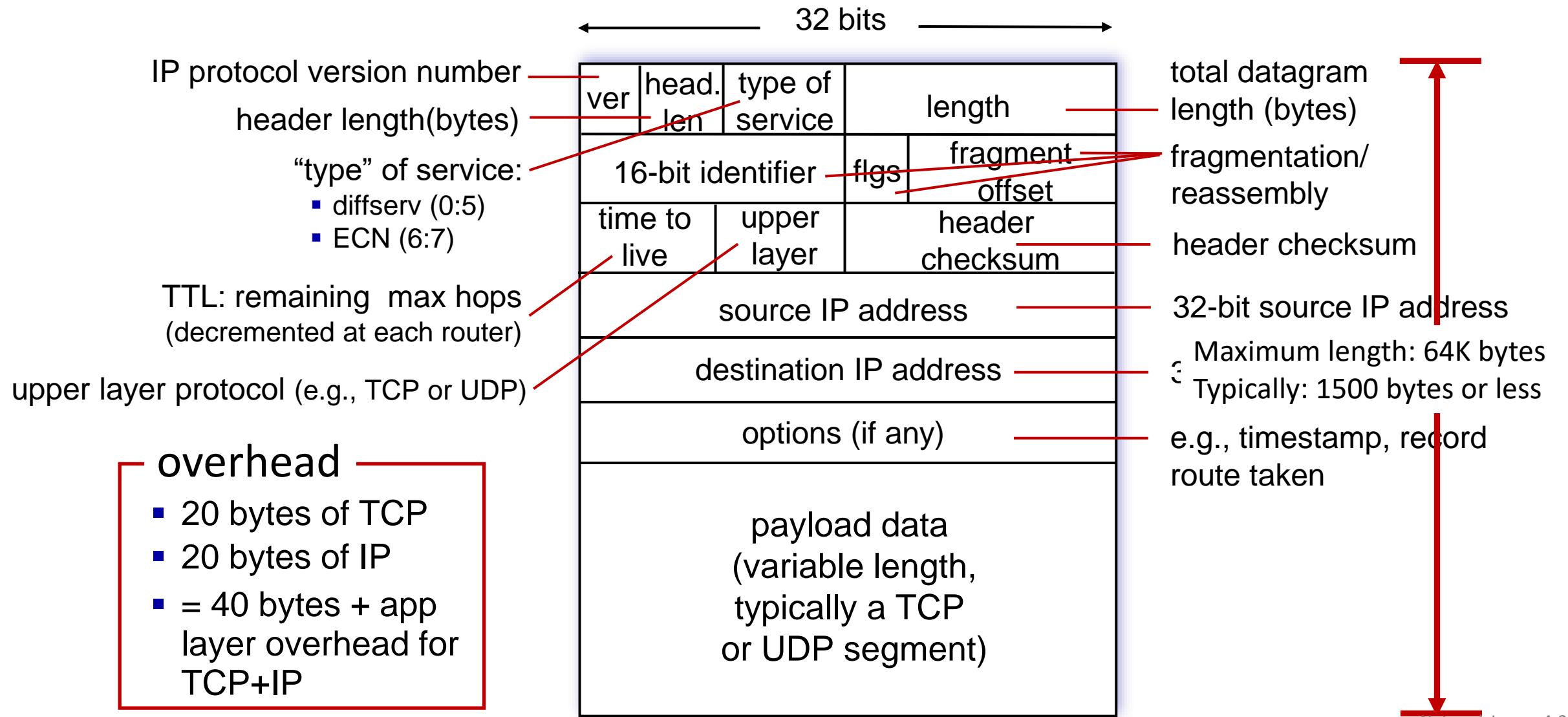


# Network Layer: Internet

host, router network layer functions:

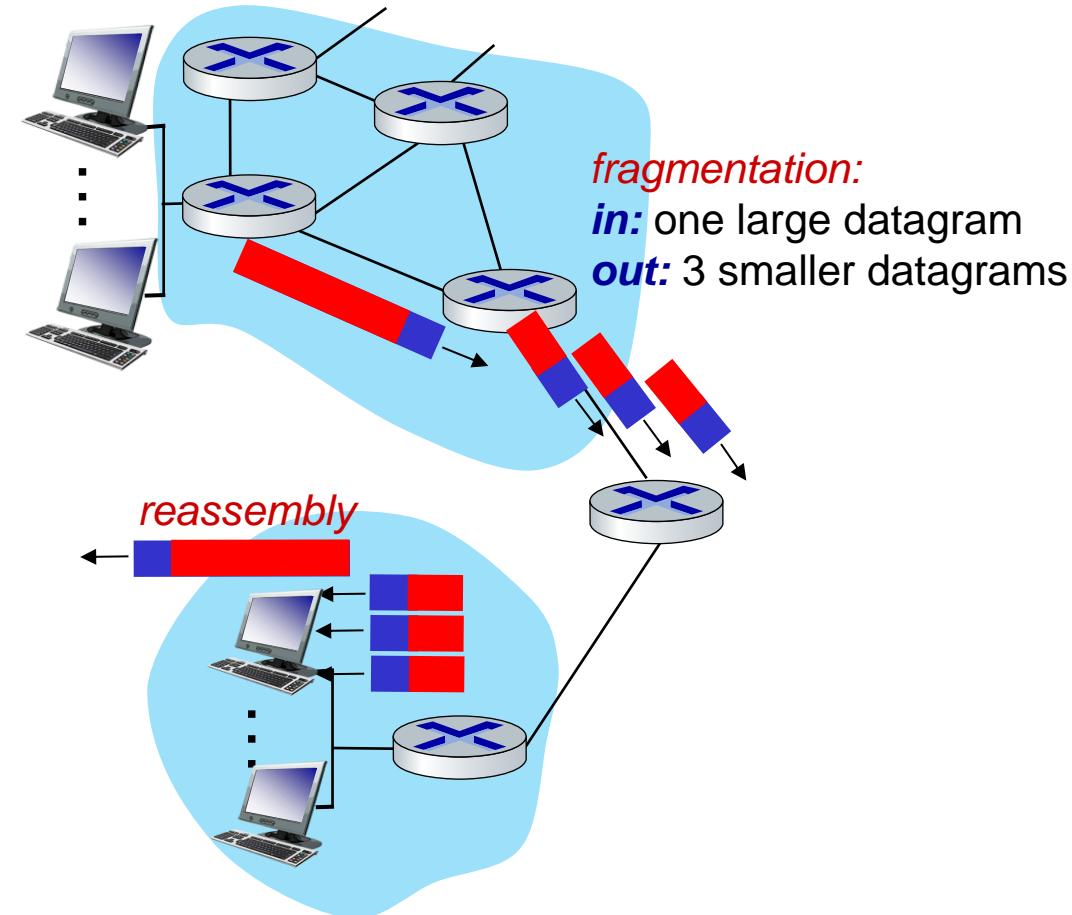


# IP Datagram format



# IP fragmentation/reassembly

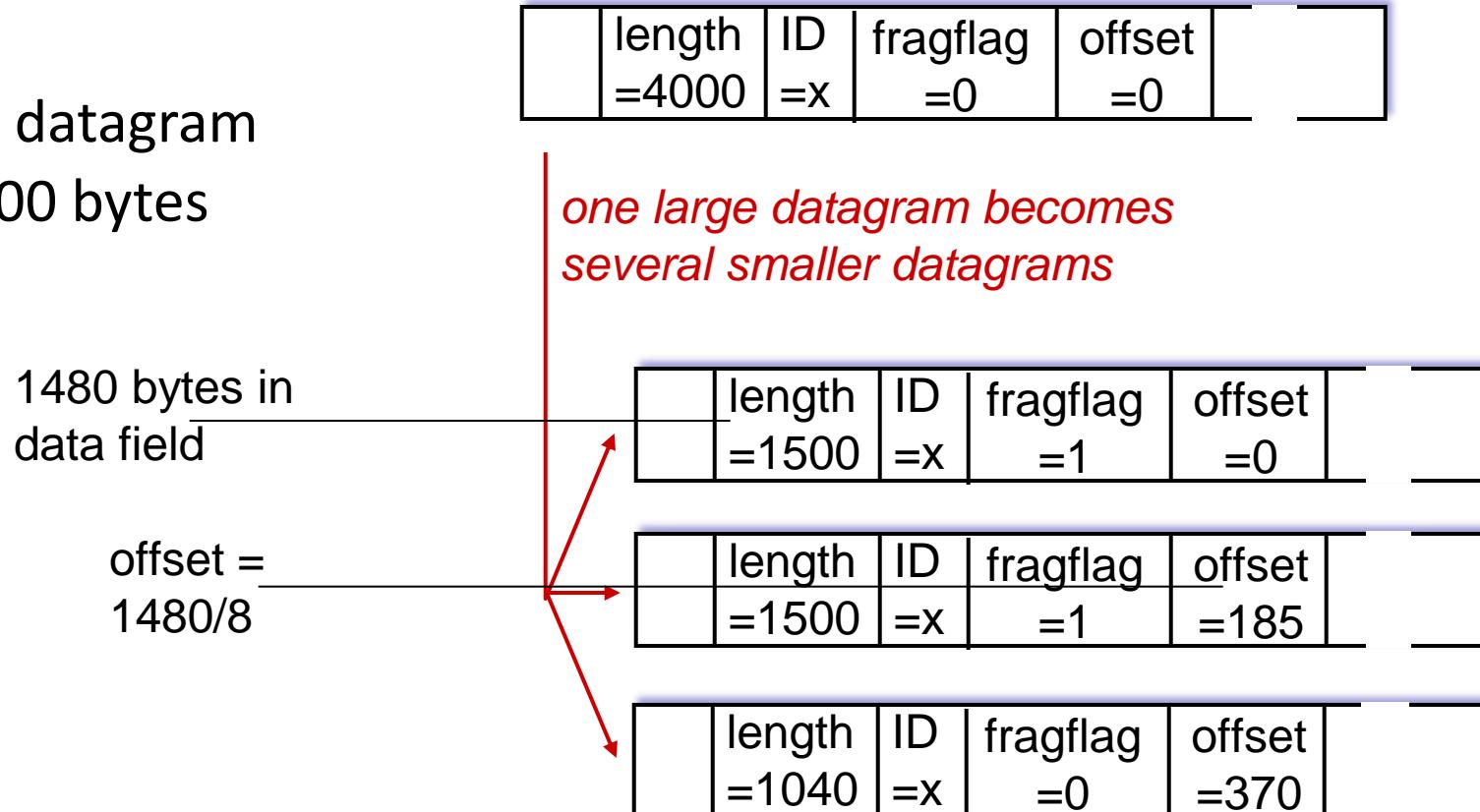
- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided (“fragmented”) within net
  - one datagram becomes several datagrams
  - “reassembled” only at *destination*
  - IP header bits used to identify, order related fragments



# IP fragmentation/reassembly

## example:

- 4000 byte datagram
- MTU = 1500 bytes



# IP fragmentation, reassembly

## example:

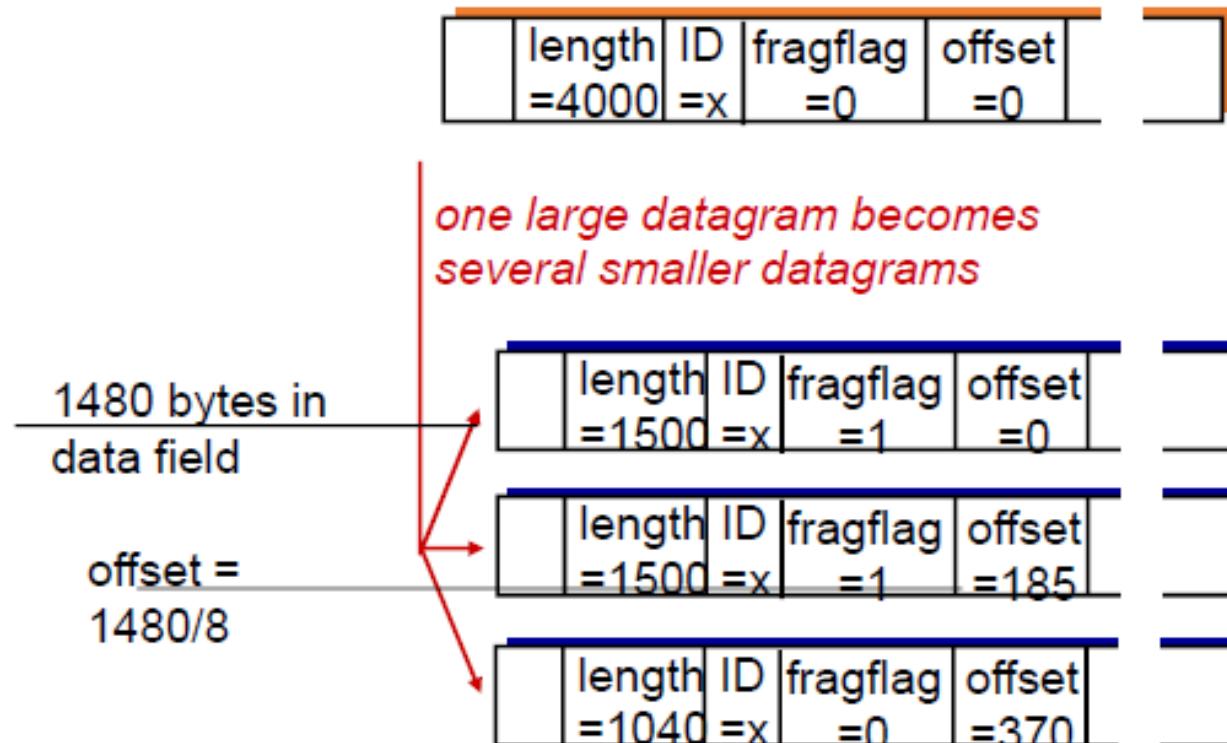
- 4000 byte datagram
- MTU = 1500 bytes

Figure shows a datagram with a data size of 4000 bytes fragmented into three fragments. The bytes in the original datagram are numbered 0 to 3999.

The first fragment carries bytes 0 to 1479. The offset for this datagram is  $0/8 = 0$ .

The second fragment carries bytes 1480 to 2959; the offset value for this fragment is  $1480/8 = 185$ .

Finally, the third fragment carries bytes 2960 to 3999. The offset value for this fragment is  $2960/8 = 370$ .



- All fragments have the same identification number, which is also the same as the original datagram.
- The offset is measured in units of 8 bytes.
- The maximum size of each fragment is the MTU minus the IP header size (Minimum 20 bytes and Maximum 60 bytes)
- If its value of flag field is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment.

## IP Fragmentation Example

- There is a host A present in network X having MTU = 520 bytes.
- There is a host B present in network Y having MTU = 200 bytes.
- There exists a network Z having MTU = 110 bytes.
- Host A wants to send a message to host B.

Consider Router-1 receives a datagram from host A having-

Header length = 20 bytes

Payload length = 500 bytes

Total length = 520 bytes

Consider Router-1 divides the datagram into 3 fragments:

Maximum amount of data that can be sent in one fragment =  $200 - 20 = 180$  bytes.

Amount of data sent in a fragment must be a multiple of 8.

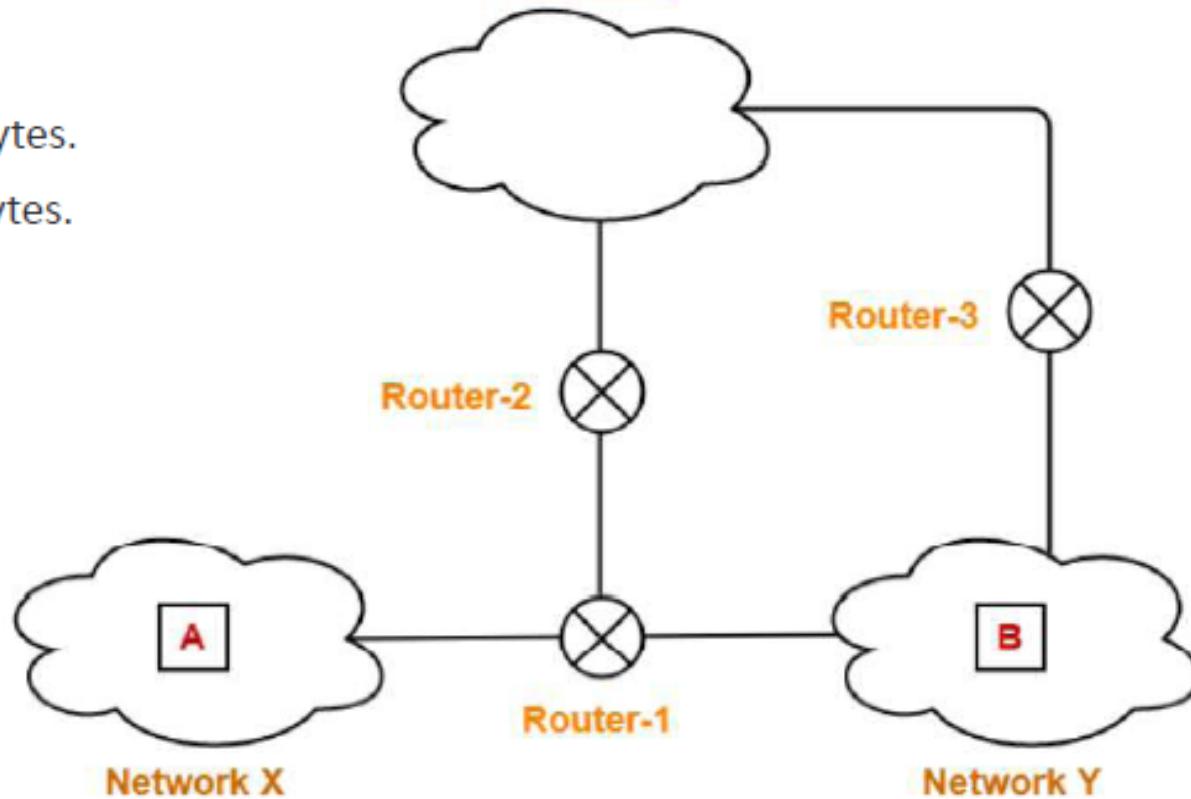
So, maximum data sent that can be in one fragment = 176 bytes.

Then,

First fragment contains the data = 176 bytes

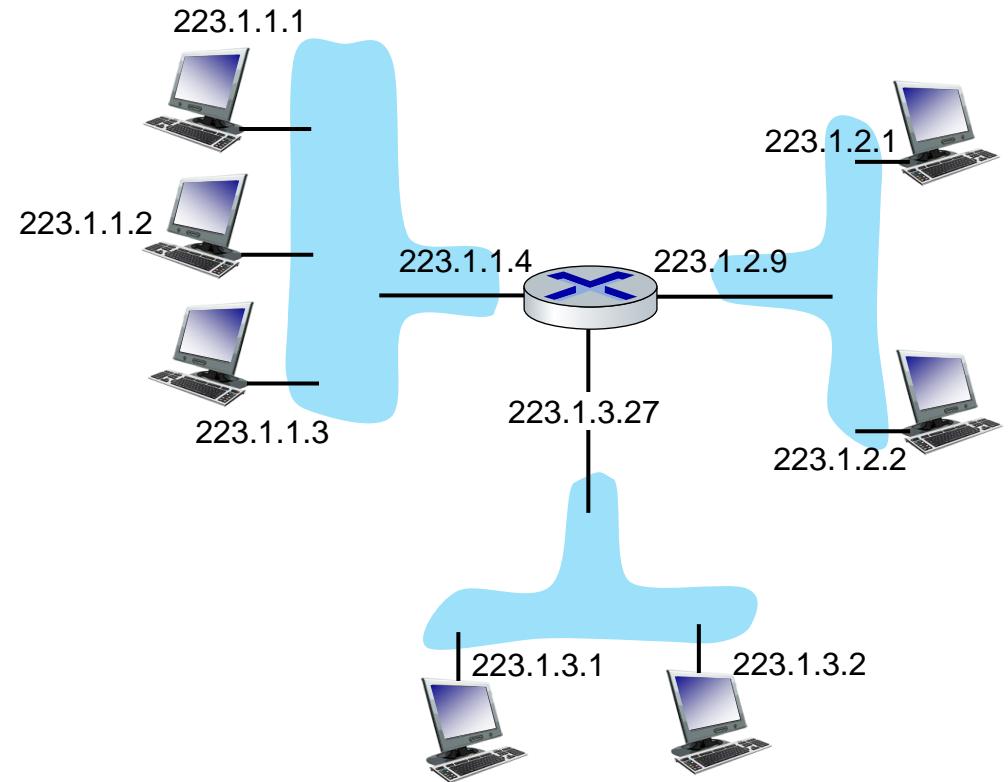
Second fragment contains the data = 176 bytes

Third fragment contains the data = 148 bytes



# IP addressing: introduction

- **IP address:** 32-bit identifier associated with each host or router *interface*
- **interface:** connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

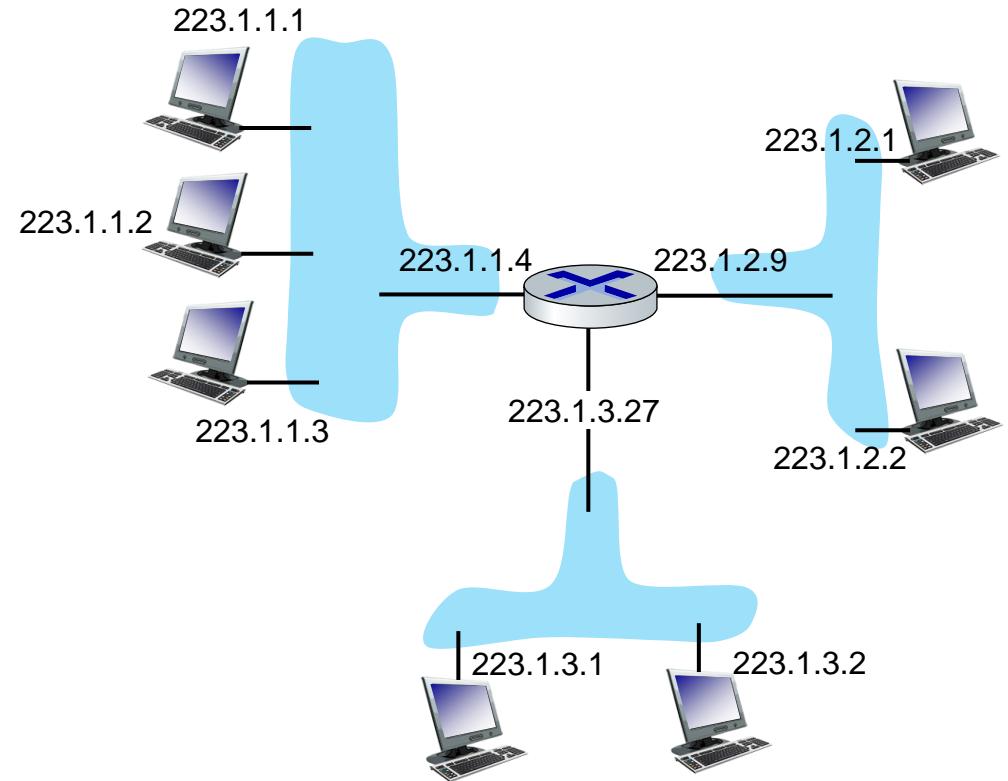


dotted-decimal IP address notation:

223.1.1.1 =  $\begin{array}{cccc} 11011111 & 00000001 & 00000001 & 00000001 \\ \hline 223 & 1 & 1 & 1 \end{array}$

# IP addressing: introduction

- **IP address:** 32-bit identifier associated with each host or router *interface*
- **interface:** connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)



dotted-decimal IP address notation:

223.1.1.1 =  $\begin{array}{cccc} 11011111 & 00000001 & 00000001 & 00000001 \end{array}$

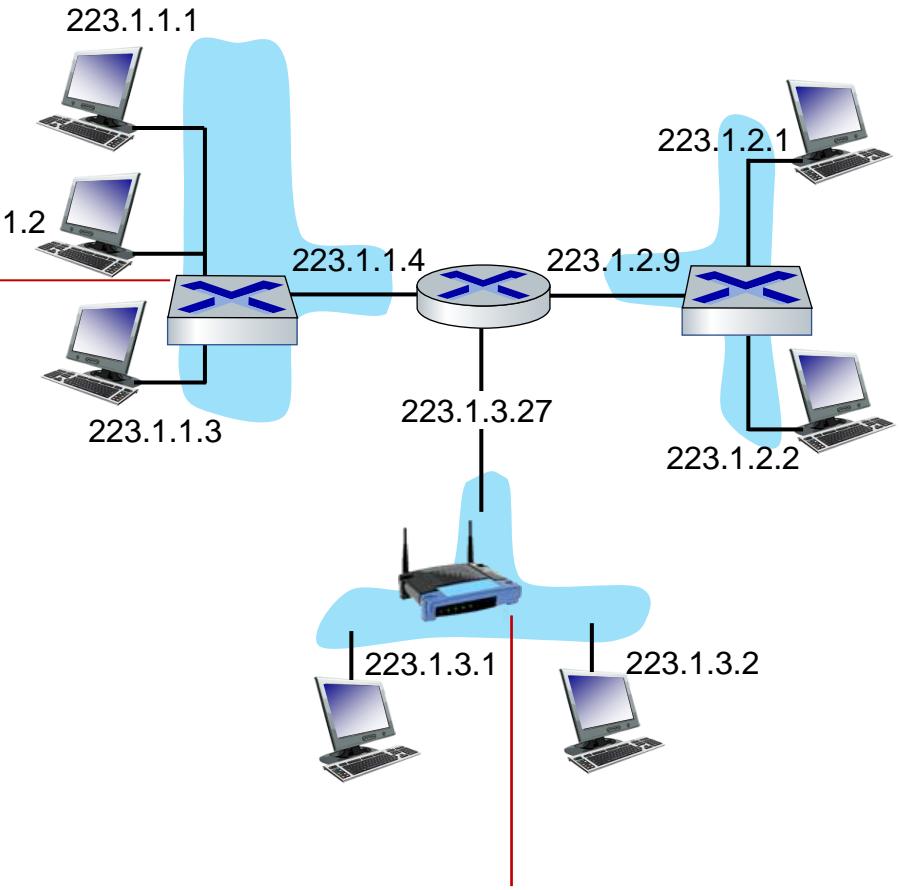
# IP addressing: introduction

**Q:** how are interfaces actually connected?

**A:** we'll learn about that in chapters 6, 7

*For now:* don't need to worry about how one interface is connected to another (with no intervening router)

**A:** wired Ethernet interfaces connected by Ethernet switches



**A:** wireless WiFi interfaces connected by WiFi base station

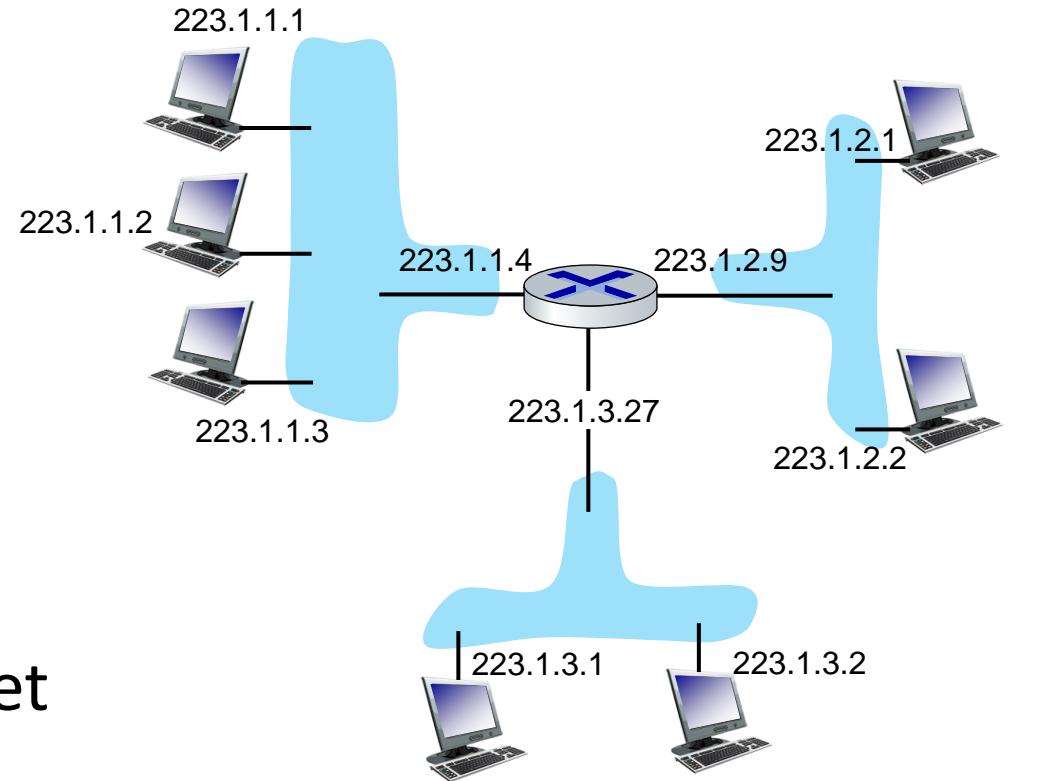
# Subnets

- *What's a subnet ?*

- device interfaces that can physically reach each other **without passing through an intervening router**

- IP addresses have structure:

- **subnet part:** devices in same subnet have common high order bits
- **host part:** remaining low order bits

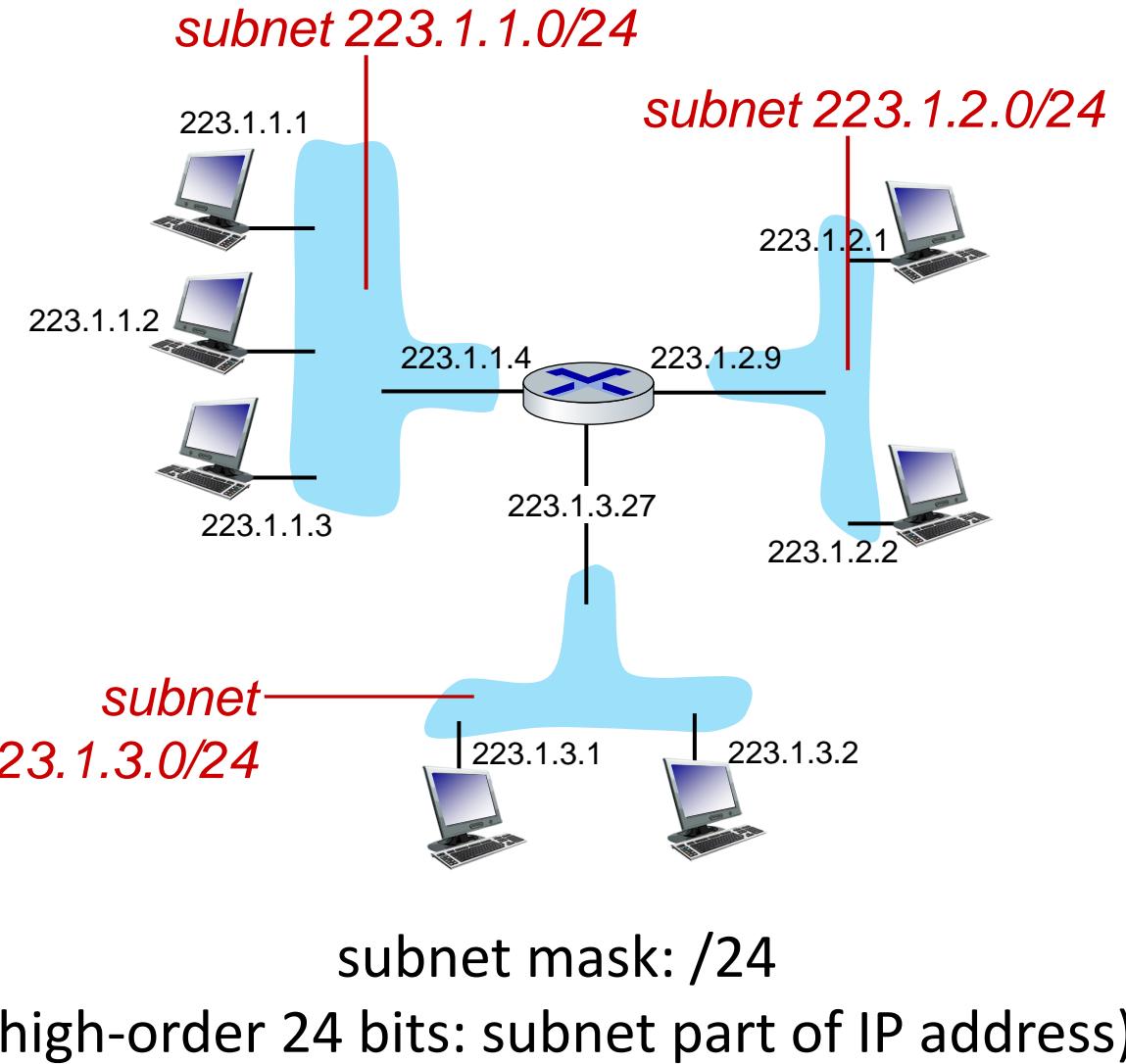


network consisting of 3 subnets

# Subnets

*Recipe for defining subnets:*

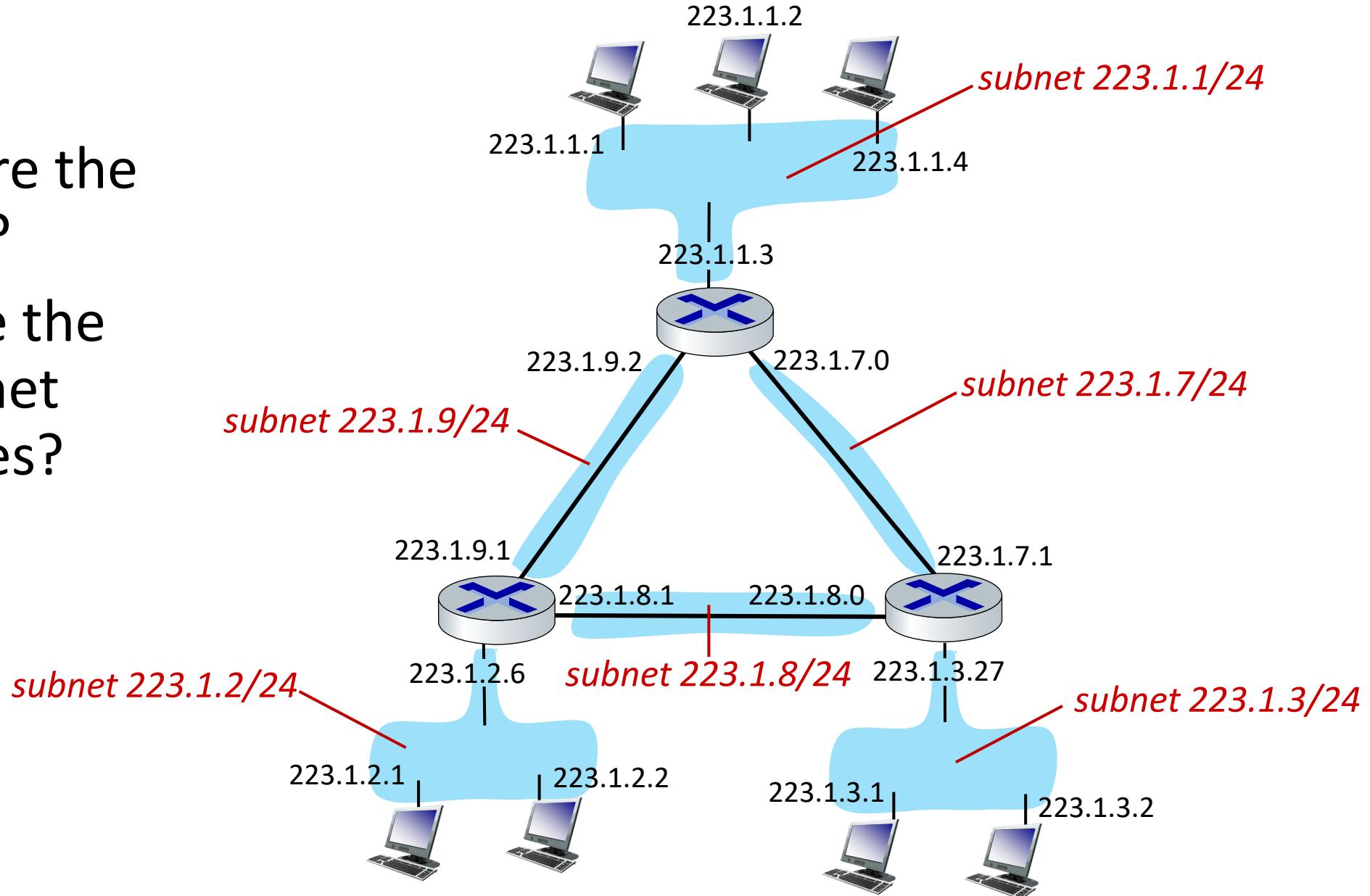
- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*



subnet mask: /24  
(high-order 24 bits: subnet part of IP address)

# Subnets

- where are the subnets?
- what are the /24 subnet addresses?



## Classful Addressing

The IP address is divided into five sub-classes. These are:

- Class A
- Class B
- Class C
- Class D
- Class E

Each of these classes has a valid range of IP addresses. Classes D and E are reserved for multicast and experimental purposes respectively. The order of bits in the first octet determine the classes of IP address.

IPv4 address is divided into two parts:

- Network ID
- Host ID

**The classification is shown with the help of the below table:**

<b>Class</b>	<b>Ist octet Decimal Range</b>	<b>Network/Host ID</b>	<b>Default subnet mask</b>
A	1 to 126	N.H.H.H	255.0.0.0
B	128 to 191	N.N.H.H	255.255.0.0
C	192 to 223	N.N.N.H	255.255.255.0
D	224 to 239	Reserved for Multicasting	
E	240 to 254	Experimental	

# IP addressing: CIDR

CIDR: Classless InterDomain Routing (pronounced “cider”)

- subnet portion of address of arbitrary length
- address format:  $a.b.c.d/x$ , where  $x$  is # bits in subnet portion of address



CIDR notation	Netmask notation	$2^{(32-n)}$	Available hosts	Note
/0	0.0.0.0	4,294,967,296	4,294,967,294	The whole Internet
/8	255.0.0.0	16,777,216	16,777,214	Class A
/16	255.255.0.0	65,535	65,533	Class B
/24	255.255.255.0	256	254	Class C
/25	255.255.255.128	128	126	
/26	255.255.255.192	64	62	
/27	255.255.255.224	32	30	
/28	255.255.255.240	16	14	
/29	255.255.255.248	8	6	
/30	255.255.255.252	4	2	
/31	255.255.255.254	2	2*	
/32	255.255.255.255	0	1*	

# IP addresses: how to get one?

That's actually **two** questions:

1. Q: How does a *host* get IP address within its network (host part of address)?
2. Q: How does a *network* get IP address for itself (network part of address)

How does *host* get IP address?

- hard-coded by sysadmin in config file (e.g., `/etc/rc.config` in UNIX)
- **DHCP: Dynamic Host Configuration Protocol:** dynamically get address from server
  - “plug-and-play”

# DHCP: Dynamic Host Configuration Protocol

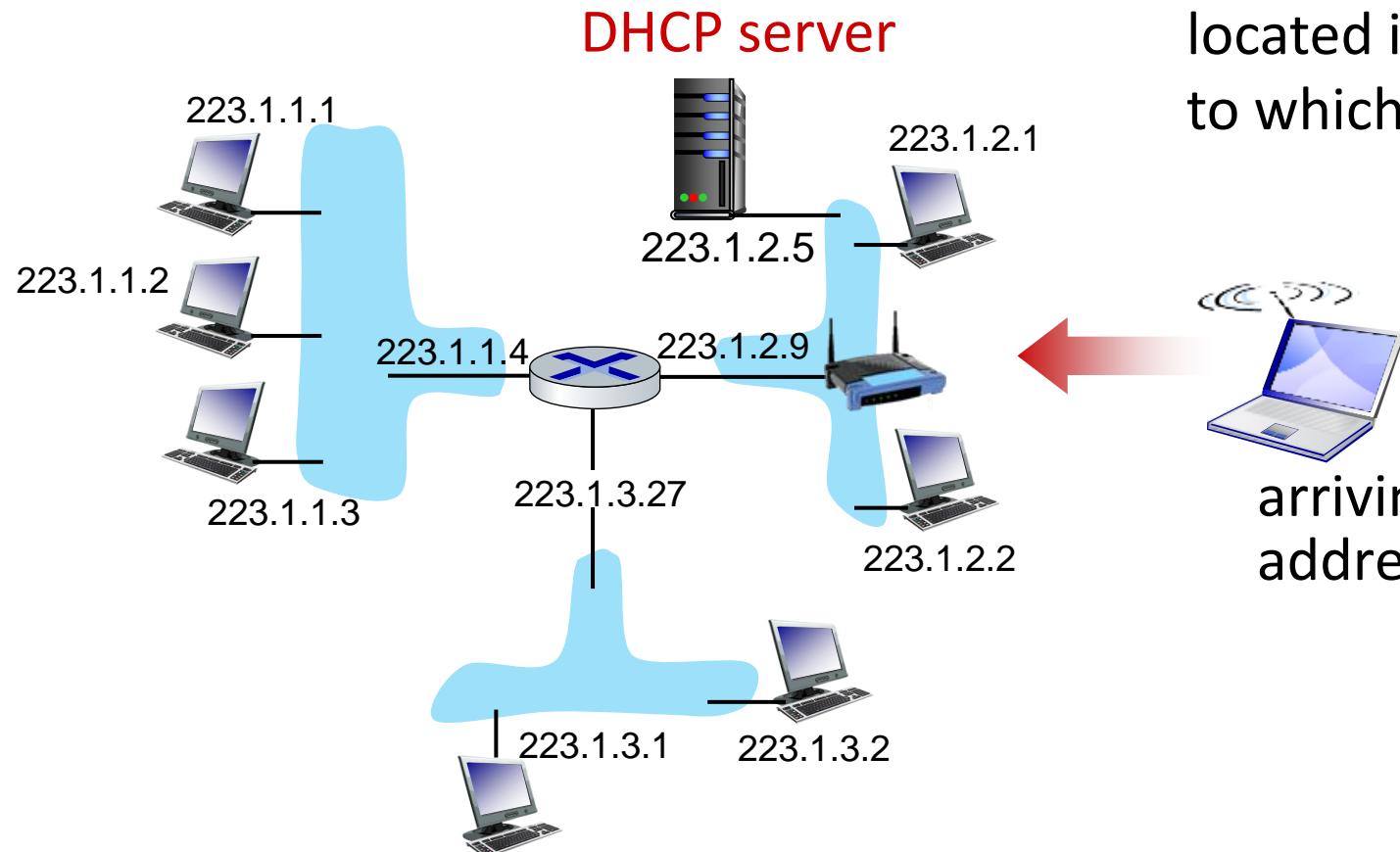
**goal:** host *dynamically* obtains IP address from network server when it “joins” network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

## DHCP overview:

- host broadcasts **DHCP discover** msg [optional]
- DHCP server responds with **DHCP offer** msg [optional]
- host requests IP address: **DHCP request** msg
- DHCP server sends address: **DHCP ack** msg

# DHCP client-server scenario

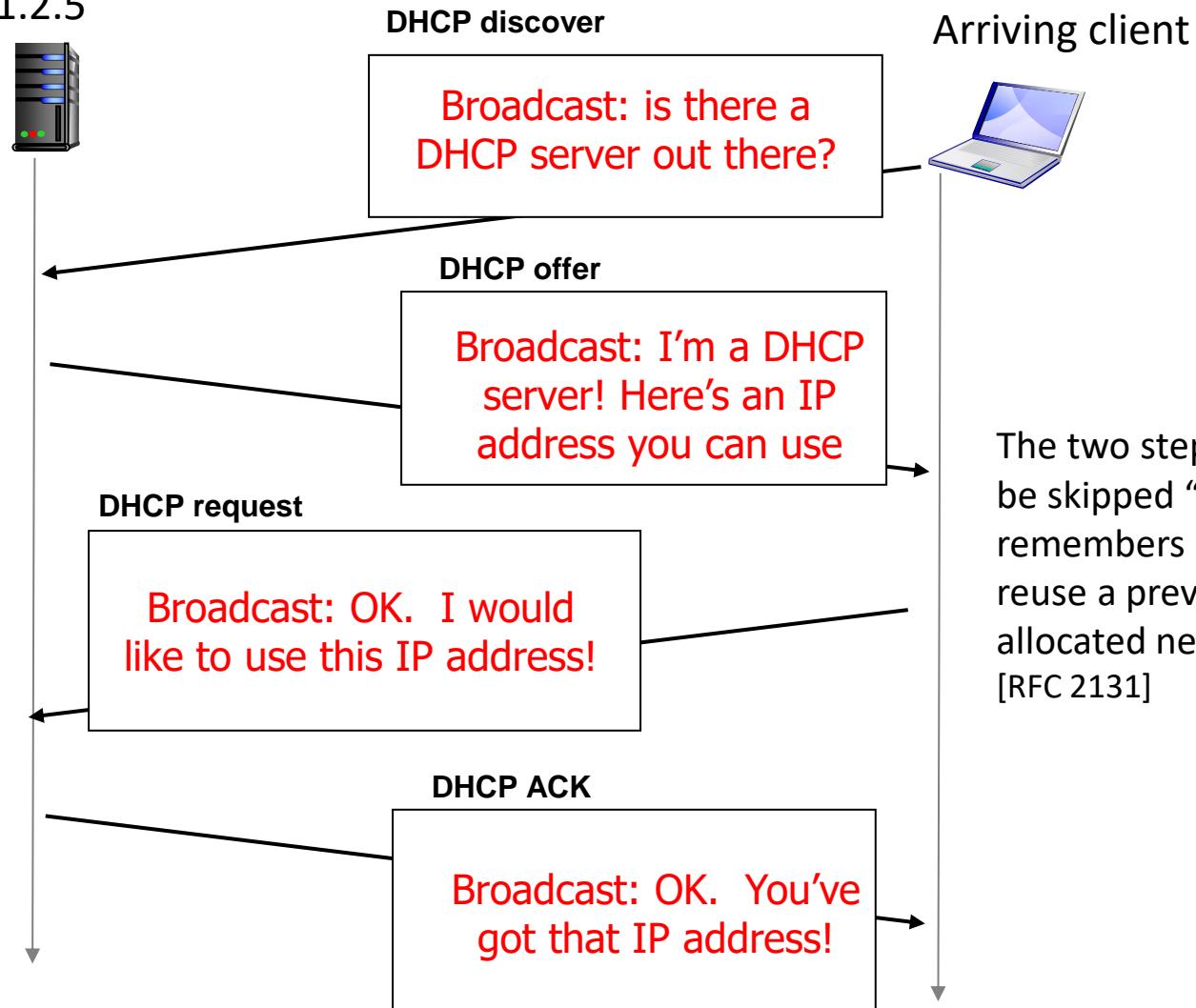


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached

arriving **DHCP client** needs address in this network

# DHCP client-server scenario

DHCP server: 223.1.2.5

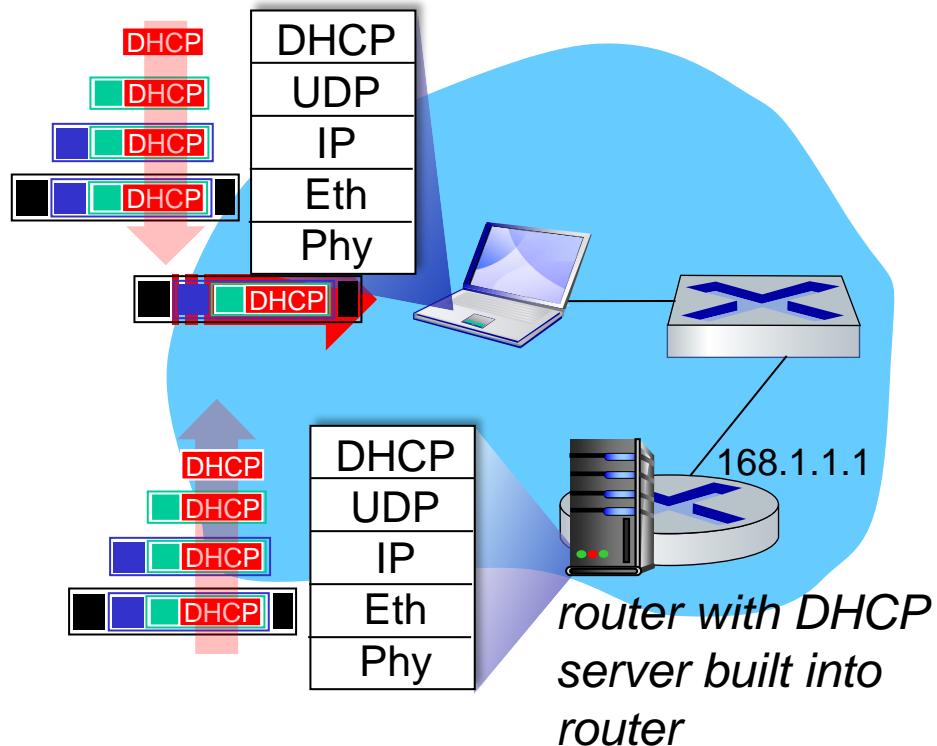


# DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

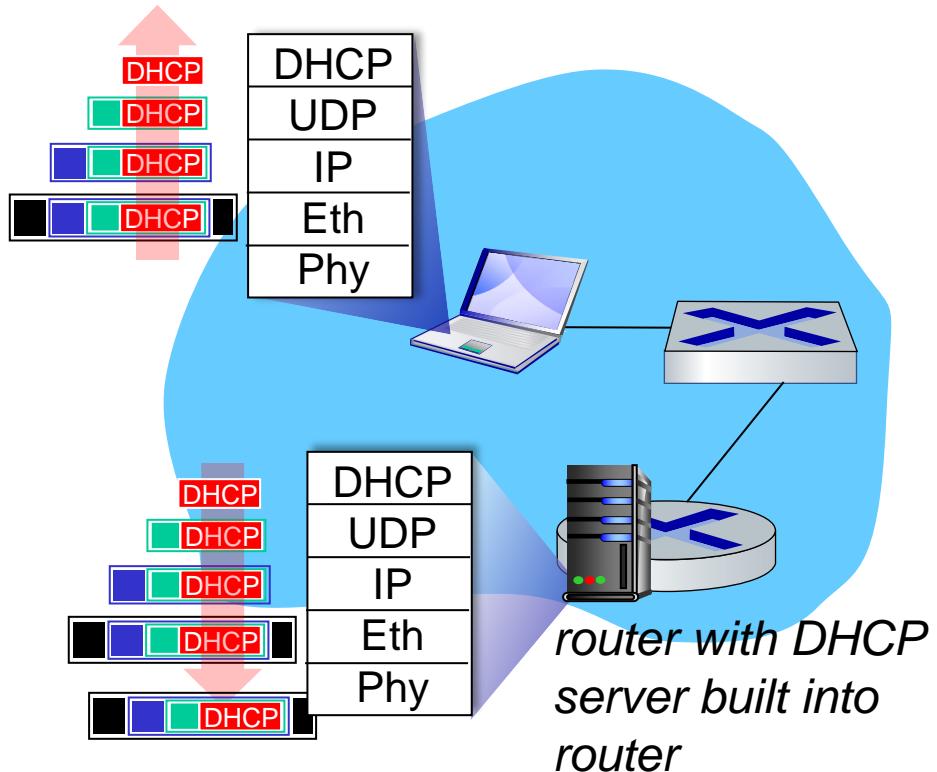
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example



- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet de-mux'ed to IP de-mux'ed, UDP de-mux'ed to DHCP

# DHCP: example



- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulated DHCP server reply forwarded to client, de-muxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

# IP addresses: how to get one?

**Q:** how does *network* get subnet part of IP address?

**A:** gets allocated portion of its provider ISP's address space

ISP's block	<u>11001000</u> <u>00010111</u> <u>00010000</u> <u>00000000</u>	200.23.16.0/20
-------------	---	----------------

ISP can then allocate out its address space in 8 blocks:

Organization 0	<u>11001000</u> <u>00010111</u> <u>00010000</u> <u>00000000</u>	200.23.16.0/23
----------------	---	----------------

Organization 1	<u>11001000</u> <u>00010111</u> <u>00010010</u> <u>00000000</u>	200.23.18.0/23
----------------	---	----------------

Organization 2	<u>11001000</u> <u>00010111</u> <u>00010100</u> <u>00000000</u>	200.23.20.0/23
----------------	---	----------------

...

.....

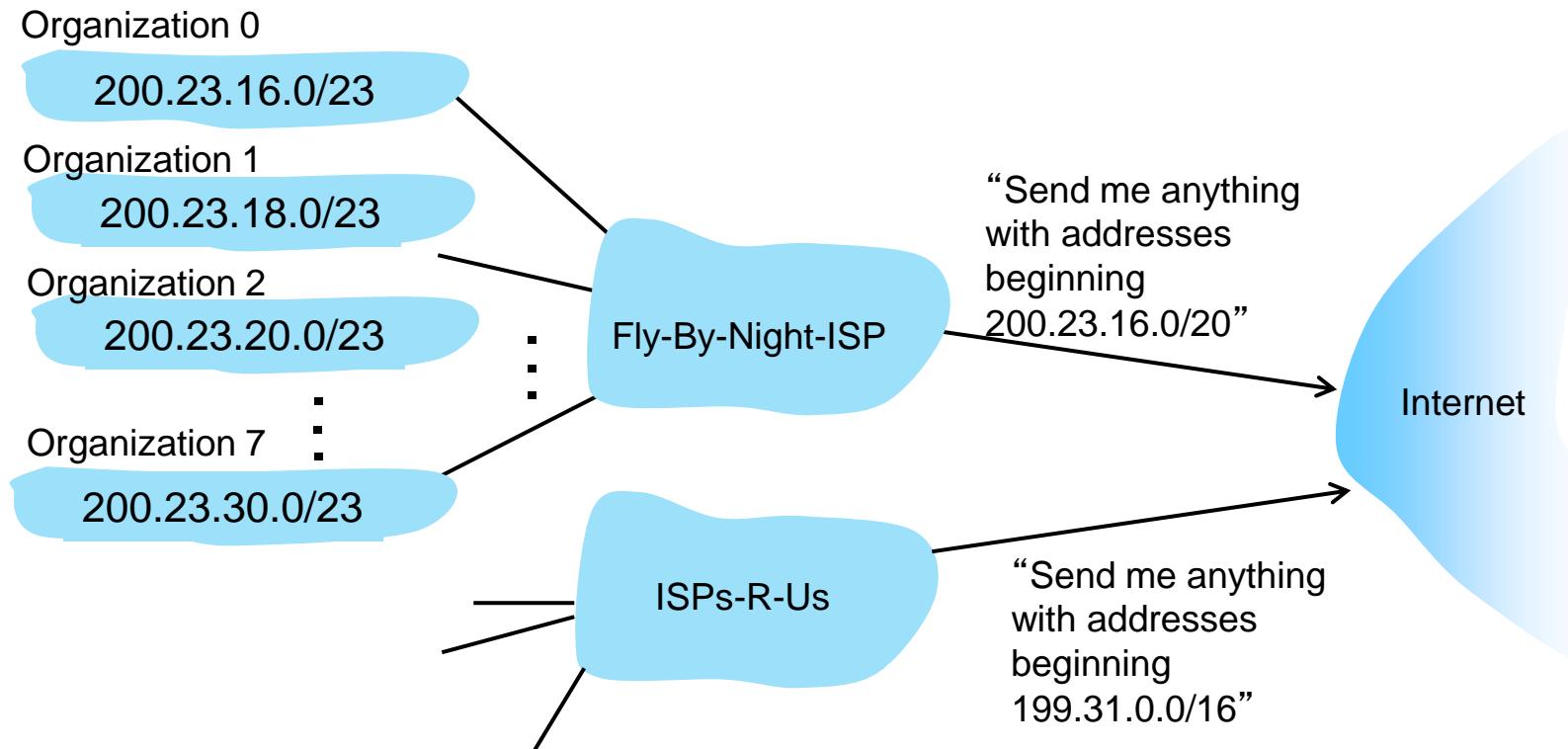
.....

.....

Organization 7	<u>11001000</u> <u>00010111</u> <u>00011110</u> <u>00000000</u>	200.23.30.0/23
----------------	---	----------------

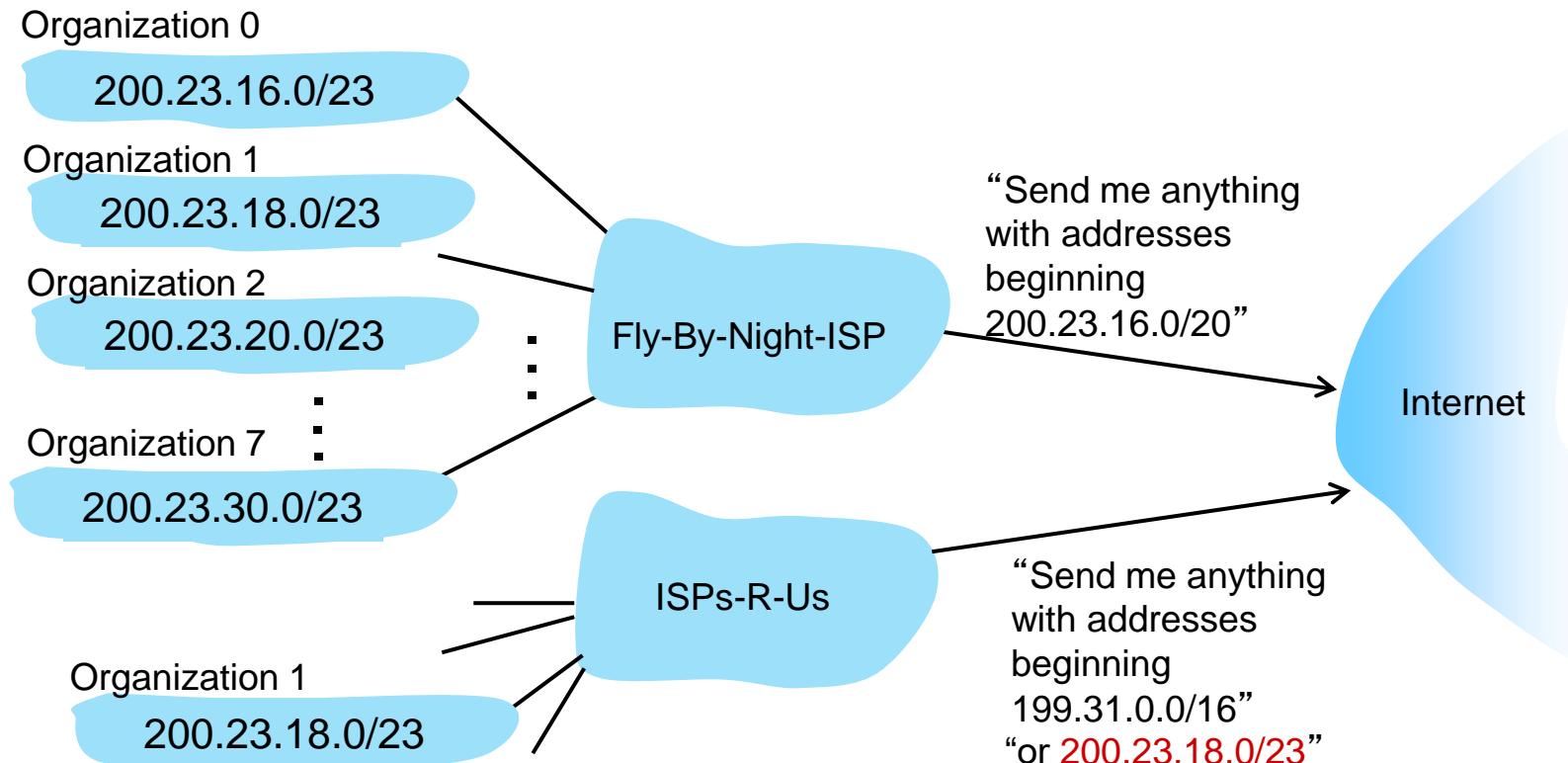
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



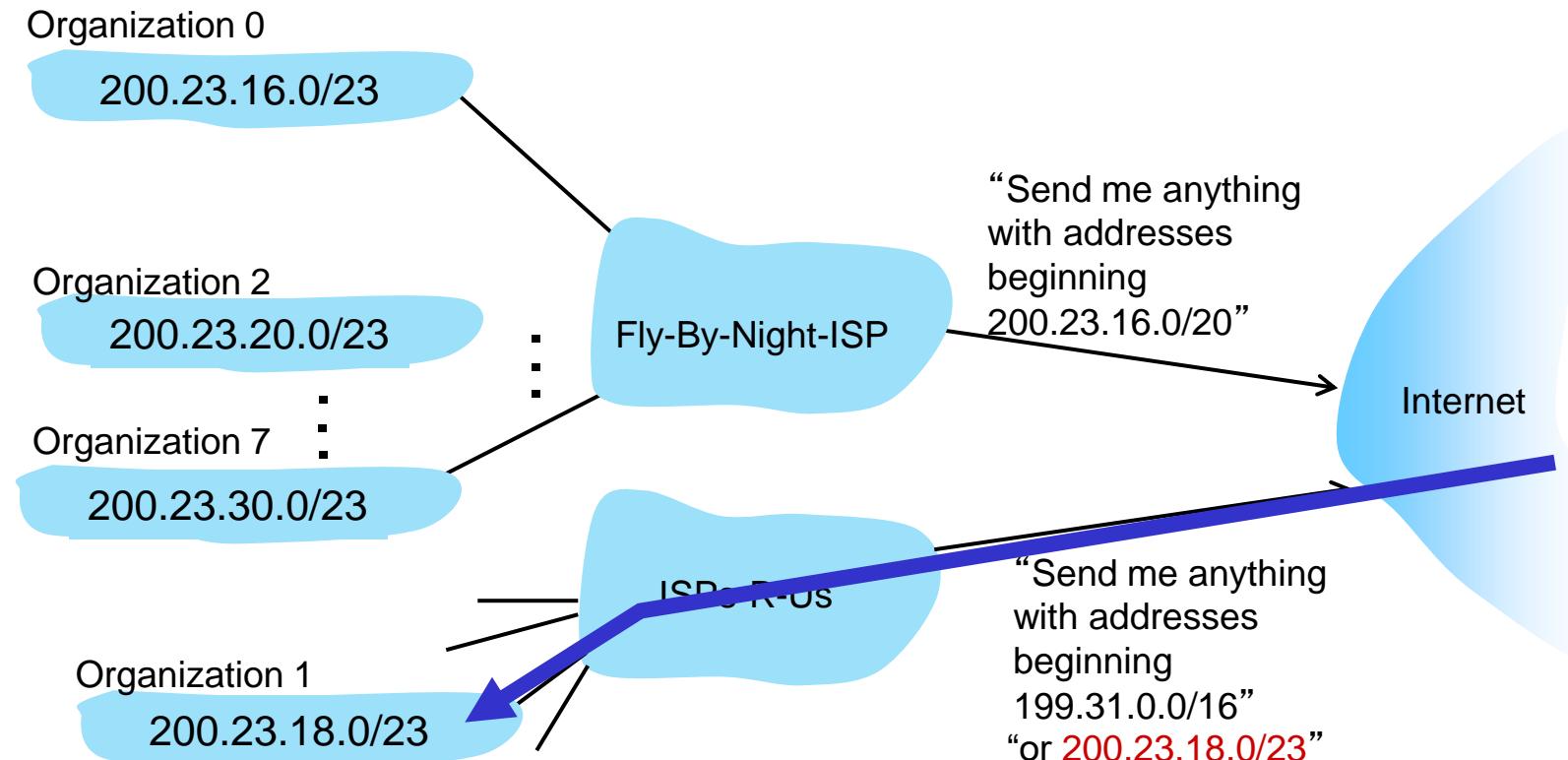
# Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



# Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



# IP addressing: last words ...

**Q:** how does an ISP get block of addresses?

**A:** ICANN: Internet Corporation for Assigned Names and Numbers

<http://www.icann.org/>

- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management

**Q:** are there enough 32-bit IP addresses?

- ICANN allocated last chunk of IPv4 addresses to RRs in 2011
- NAT (next) helps IPv4 address space exhaustion
- IPv6 has 128-bit address space

"Who the hell knew how much address space we needed?" Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)

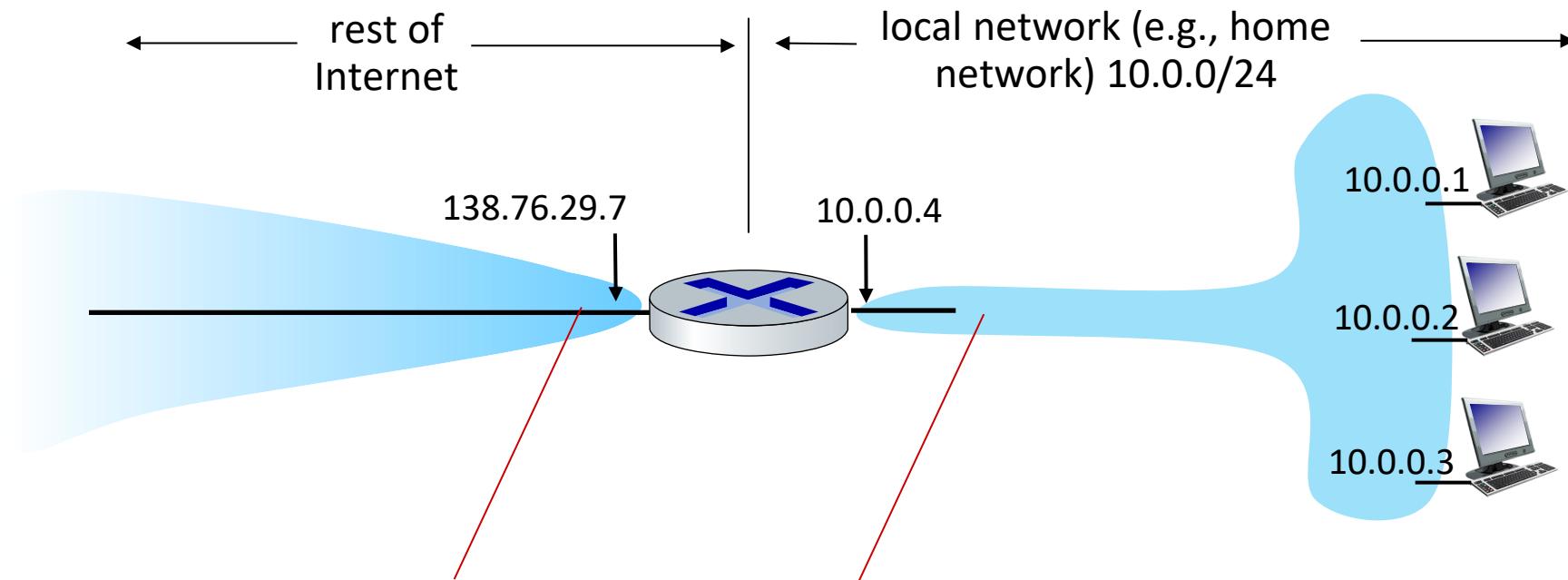
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - match+action
  - OpenFlow: match+action in action
- Middleboxes



# NAT: network address translation

**NAT:** all devices in local network share just **one** IPv4 address as far as outside world is concerned



*all* datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

# NAT: network address translation

- all devices in local network have 32-bit addresses in a “private” IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just **one** IP address needed from provider ISP for ***all*** devices
  - can change addresses of host in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable, visible by outside world

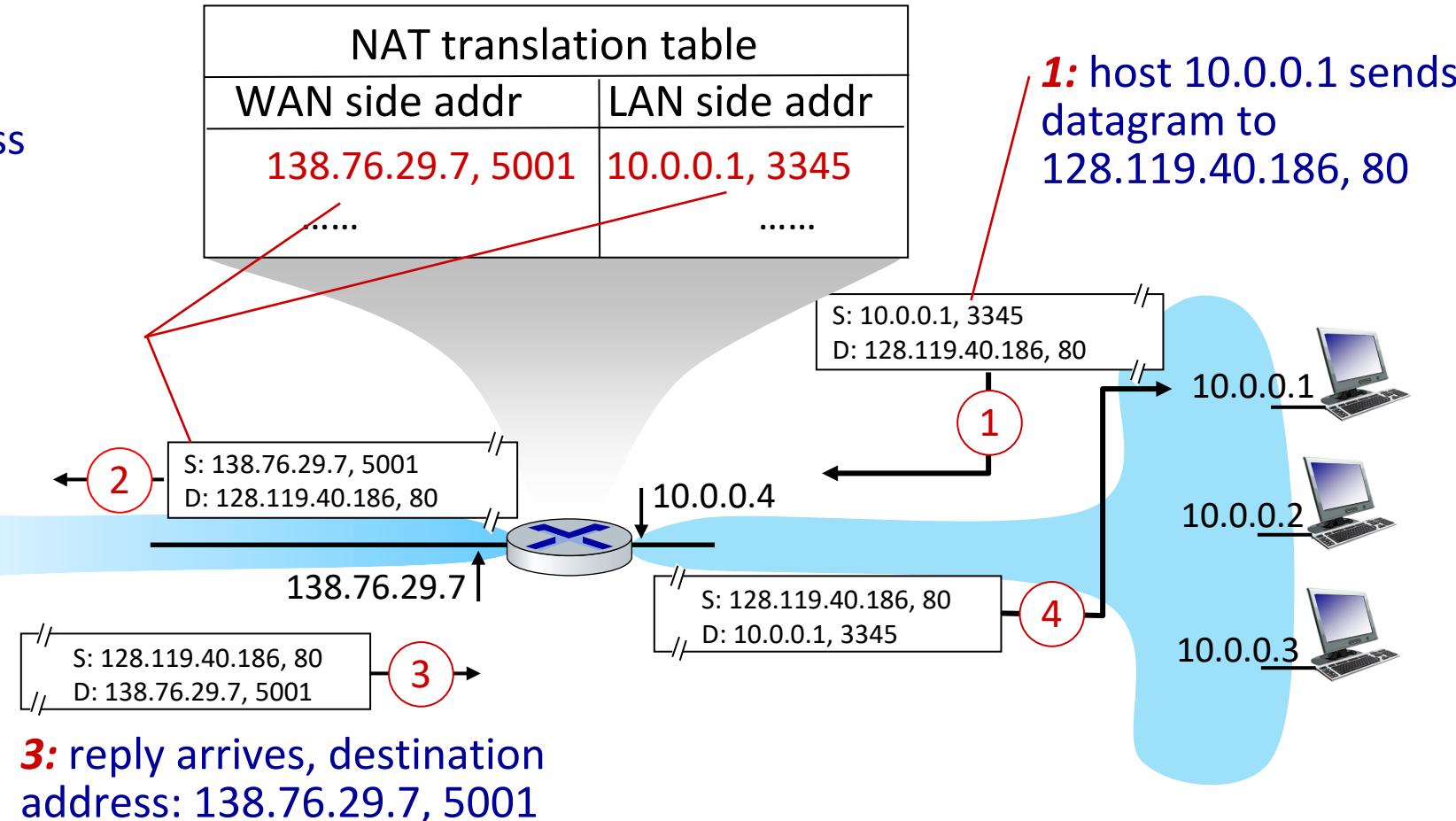
# NAT: network address translation

implementation: NAT router must (transparently):

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram 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 datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

# NAT: network address translation

**2:** NAT router changes datagram source address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table



**3:** reply arrives, destination address: 138.76.29.7, 5001

# NAT: network address translation

- NAT has been controversial:
  - routers “should” only process up to layer 3
  - address “shortage” should be solved by IPv6
  - violates end-to-end argument (port # manipulation by network-layer device)
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home and institutional nets, 4G/5G cellular nets

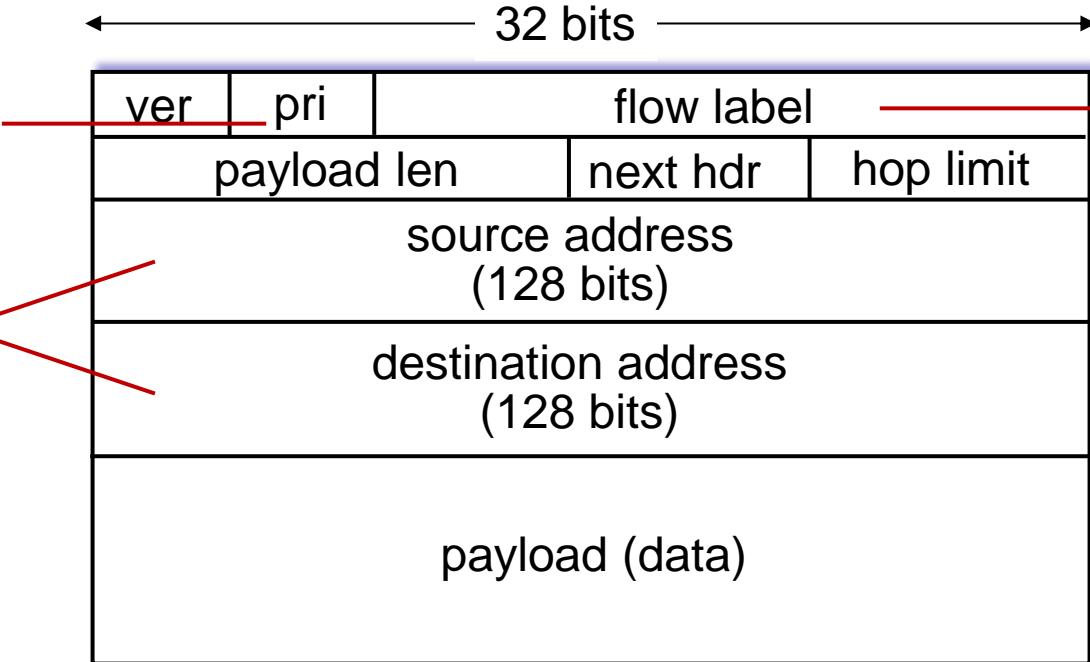
# IPv6: motivation

- **initial motivation:** 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”

# IPv6 datagram format

**priority:** identify priority among datagrams in flow

**128-bit IPv6 addresses**



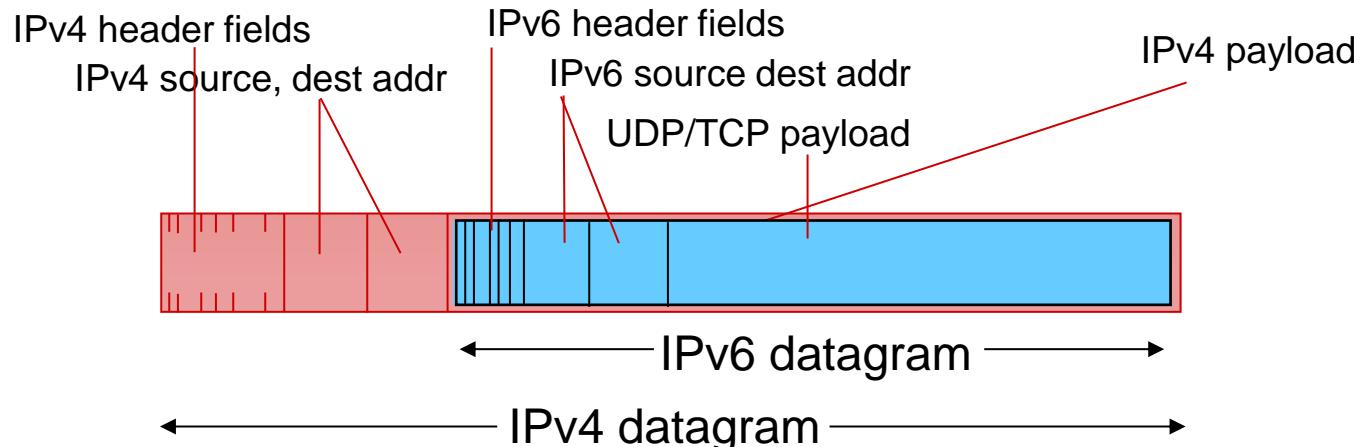
**flow label:** identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

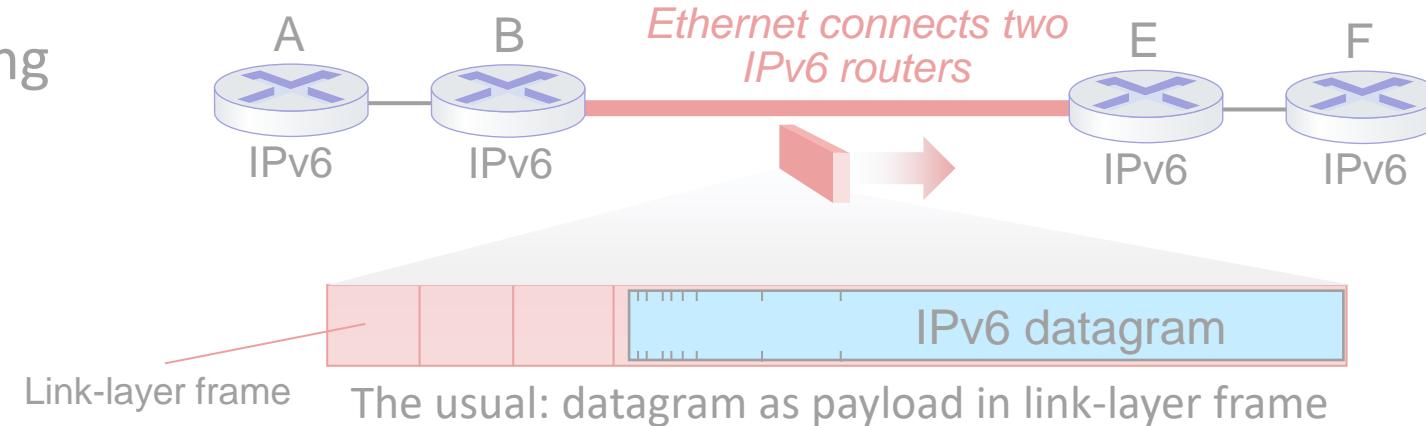
# Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?
- **tunneling:** IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers (“packet within a packet”)
  - tunneling used extensively in other contexts (4G/5G)

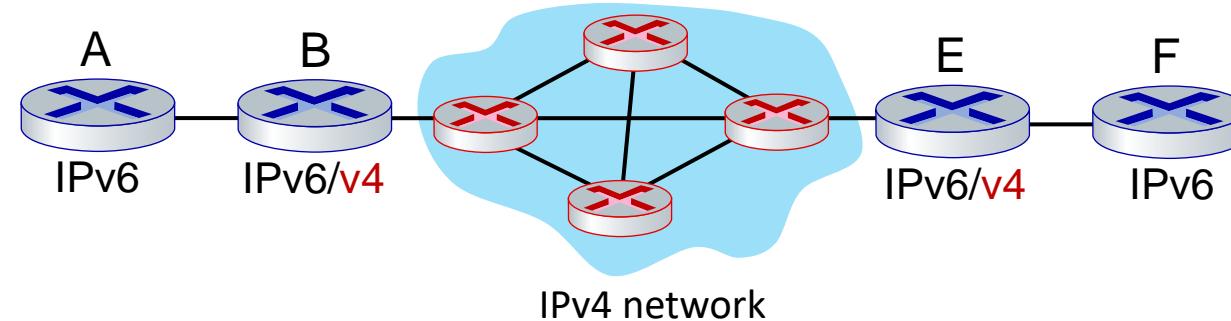


# Tunneling and encapsulation

Ethernet connecting  
two IPv6 routers:

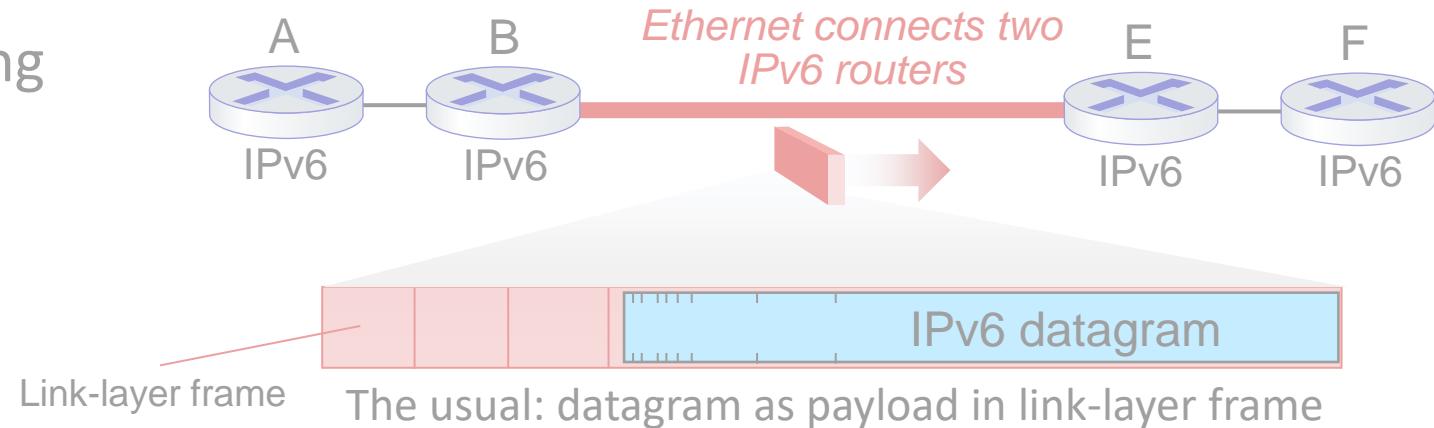


IPv4 network  
connecting two  
IPv6 routers

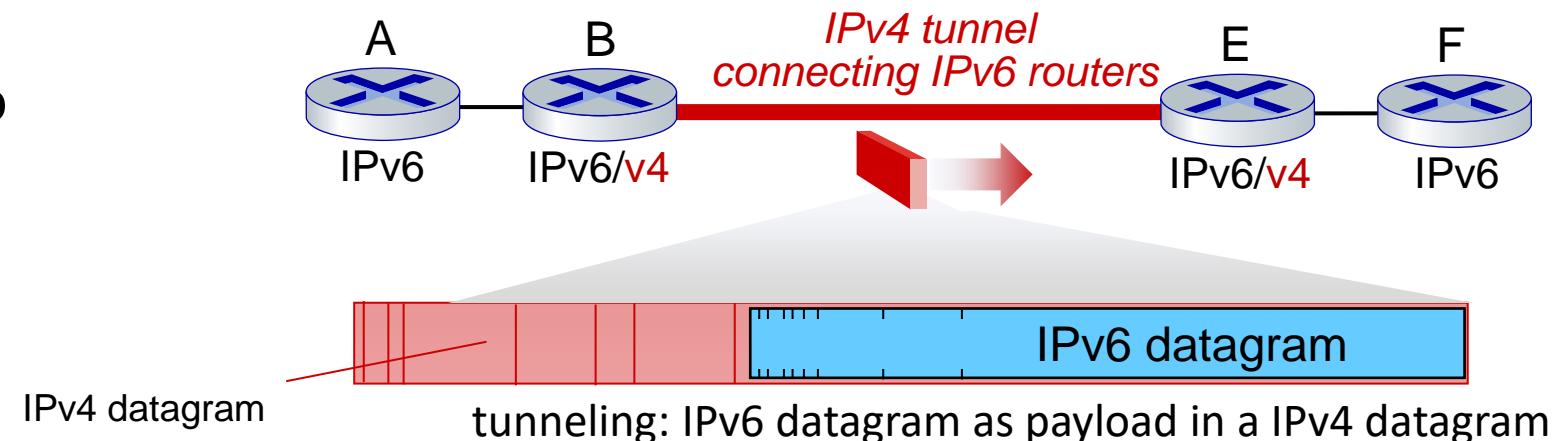


# Tunneling and encapsulation

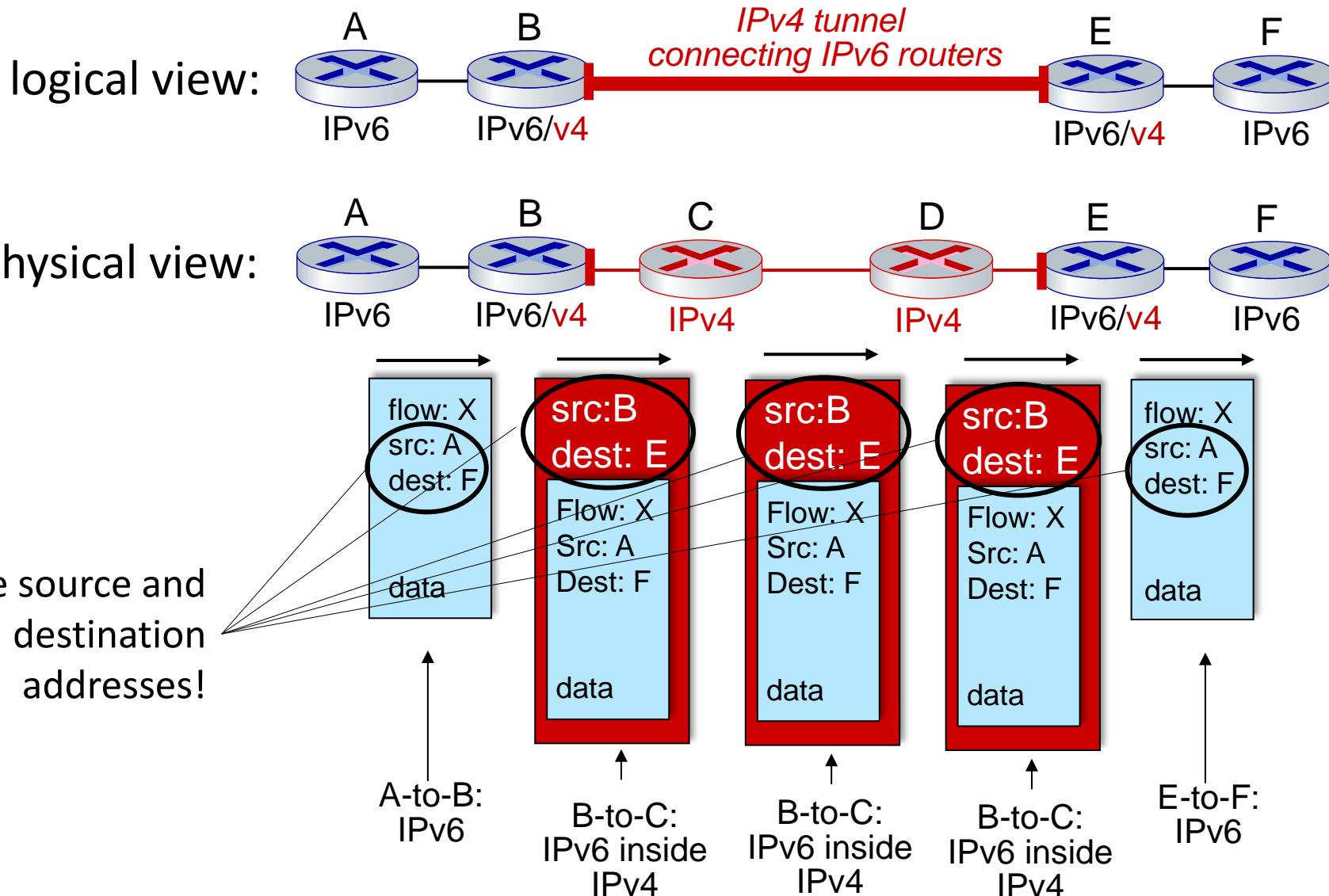
Ethernet connecting  
two IPv6 routers:



IPv4 tunnel  
connecting two  
IPv6 routers



# Tunneling

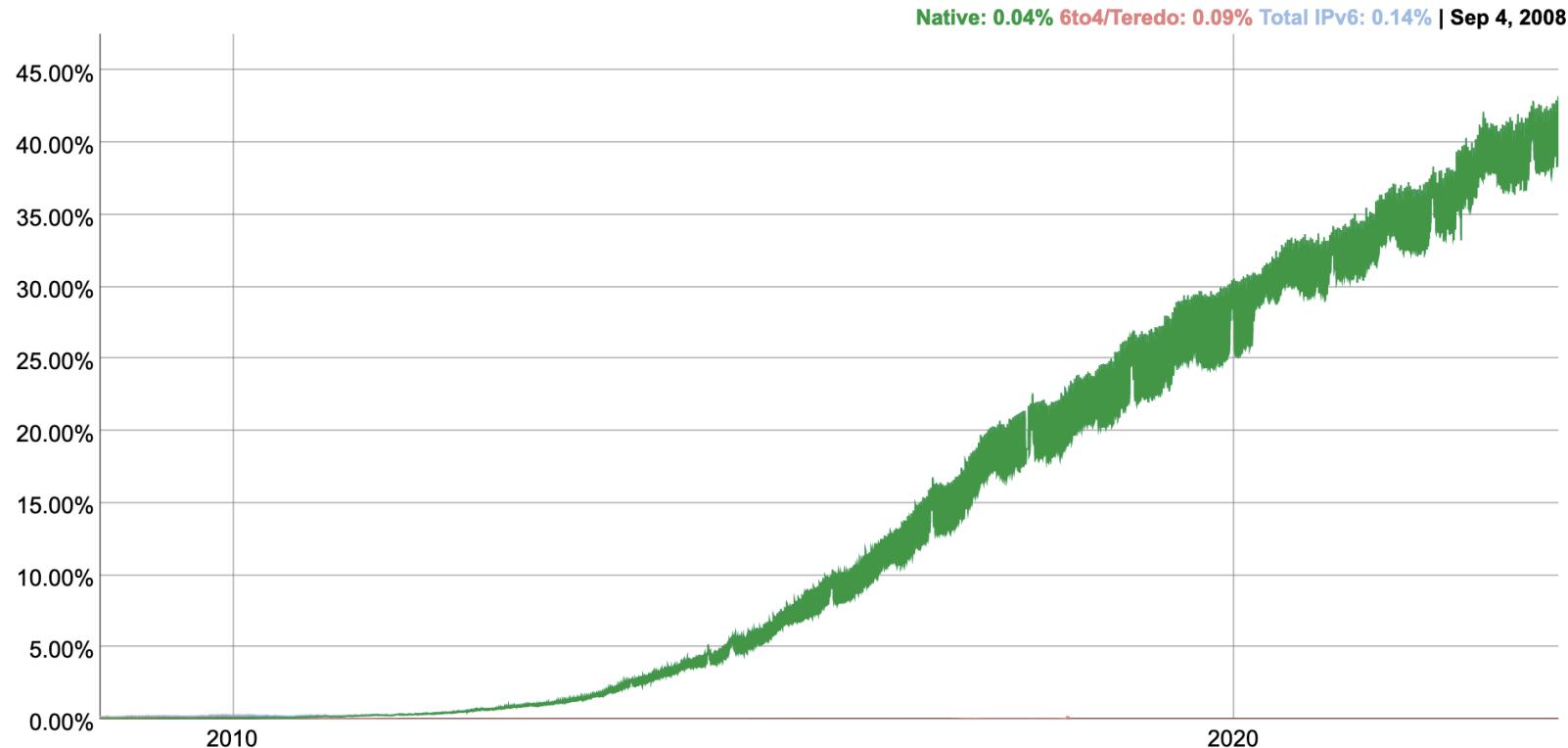


# IPv6: adoption

- Google<sup>1</sup>: ~ 40% of clients access services via IPv6 (2023)
- NIST: 1/3 of all US government domains are IPv6 capable

IPv6 Adoption

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



# IPv6: adoption

- Google<sup>1</sup>: ~ 40% of clients access services via IPv6 (2023)
- NIST: 1/3 of all US government domains are IPv6 capable
- Long (long!) time for deployment, use
  - 25 years and counting!
  - think of application-level changes in last 25 years: WWW, social media, streaming media, gaming, telepresence, ...
  - *Why?*

<sup>1</sup> <https://www.google.com/intl/en/ipv6/statistics.html>

# Network layer: “data plane” roadmap

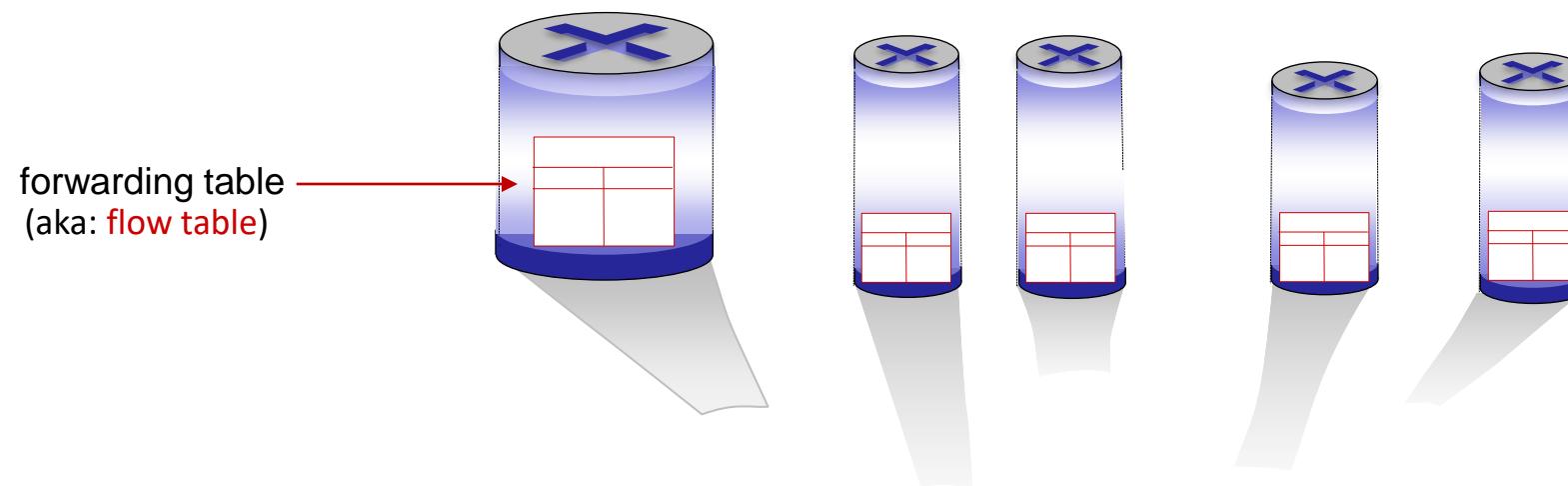
- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes



# Generalized forwarding: match plus action

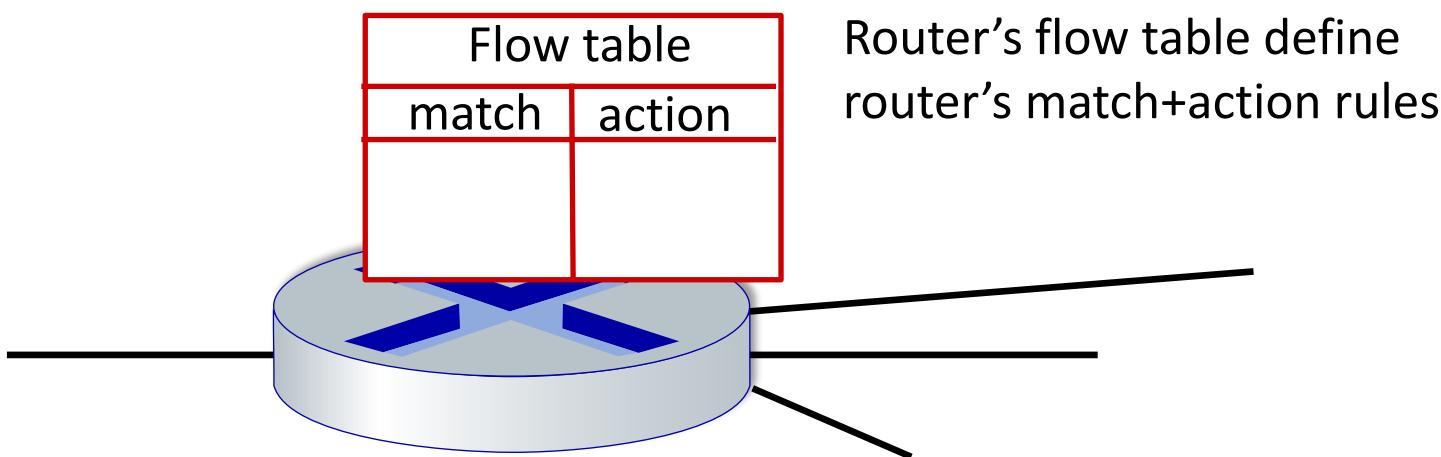
*Review:* each router contains a **forwarding table** (aka: **flow table**)

- “**match plus action**” abstraction: match bits in arriving packet, take action
  - *destination-based forwarding*: forward based on dest. IP address
  - *generalized forwarding*:
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet



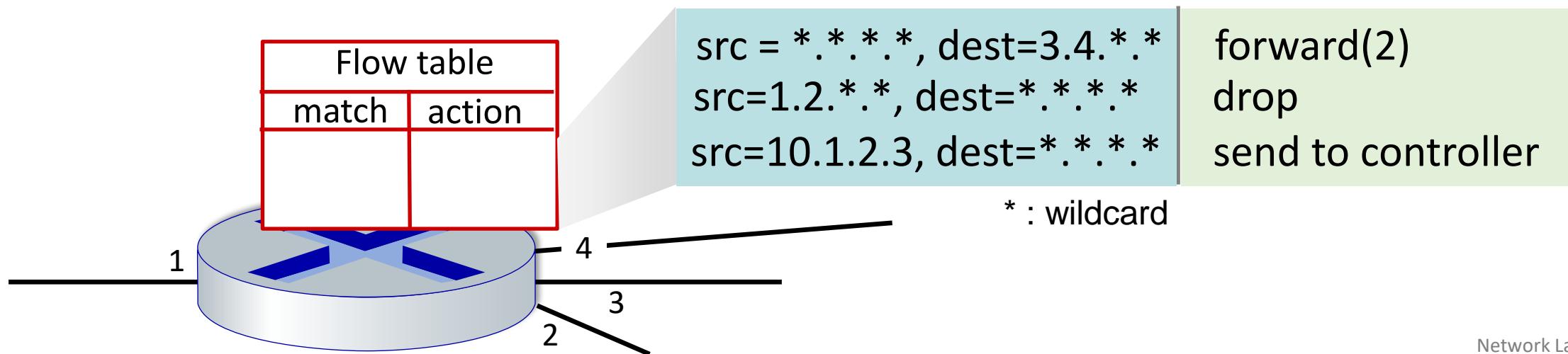
# Flow table abstraction

- **flow**: defined by header field values (in link-, network-, transport-layer fields)
- **generalized forwarding**: simple packet-handling rules
  - **match**: pattern values in packet header fields
  - **actions**: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - **priority**: disambiguate overlapping patterns
  - **counters**: #bytes and #packets

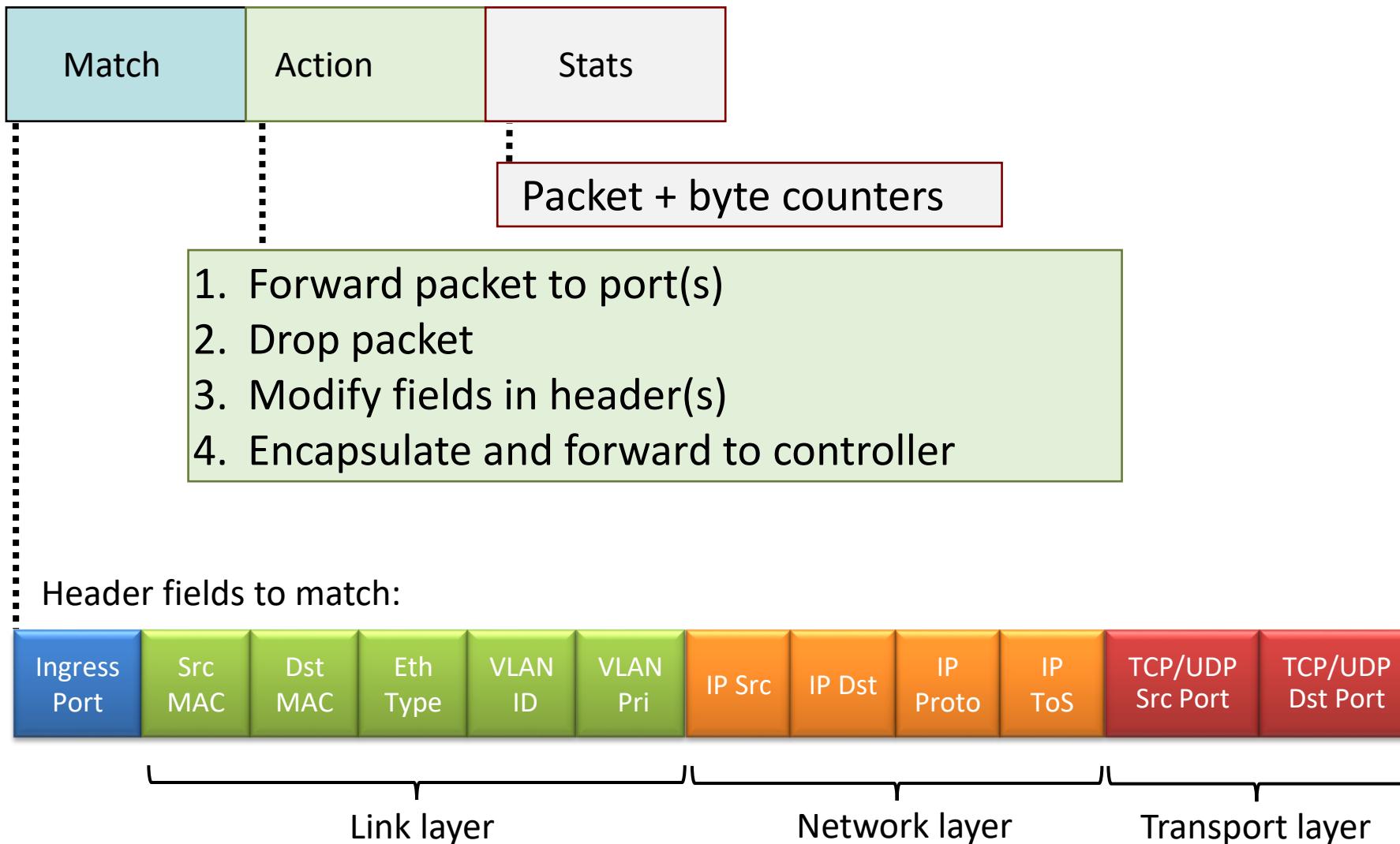


# Flow table abstraction

- **flow:** defined by header fields
- **generalized forwarding:** simple packet-handling rules
  - **match:** pattern values in packet header fields
  - **actions:** for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - **priority:** disambiguate overlapping patterns
  - **counters:** #bytes and #packets



# OpenFlow: flow table entries



# OpenFlow: examples

## Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

## Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

# OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

# OpenFlow abstraction

- **match+action:** abstraction unifies different kinds of devices

## Router

- *match:* longest destination IP prefix
- *action:* forward out a link

## Switch

- *match:* destination MAC address
- *action:* forward or flood

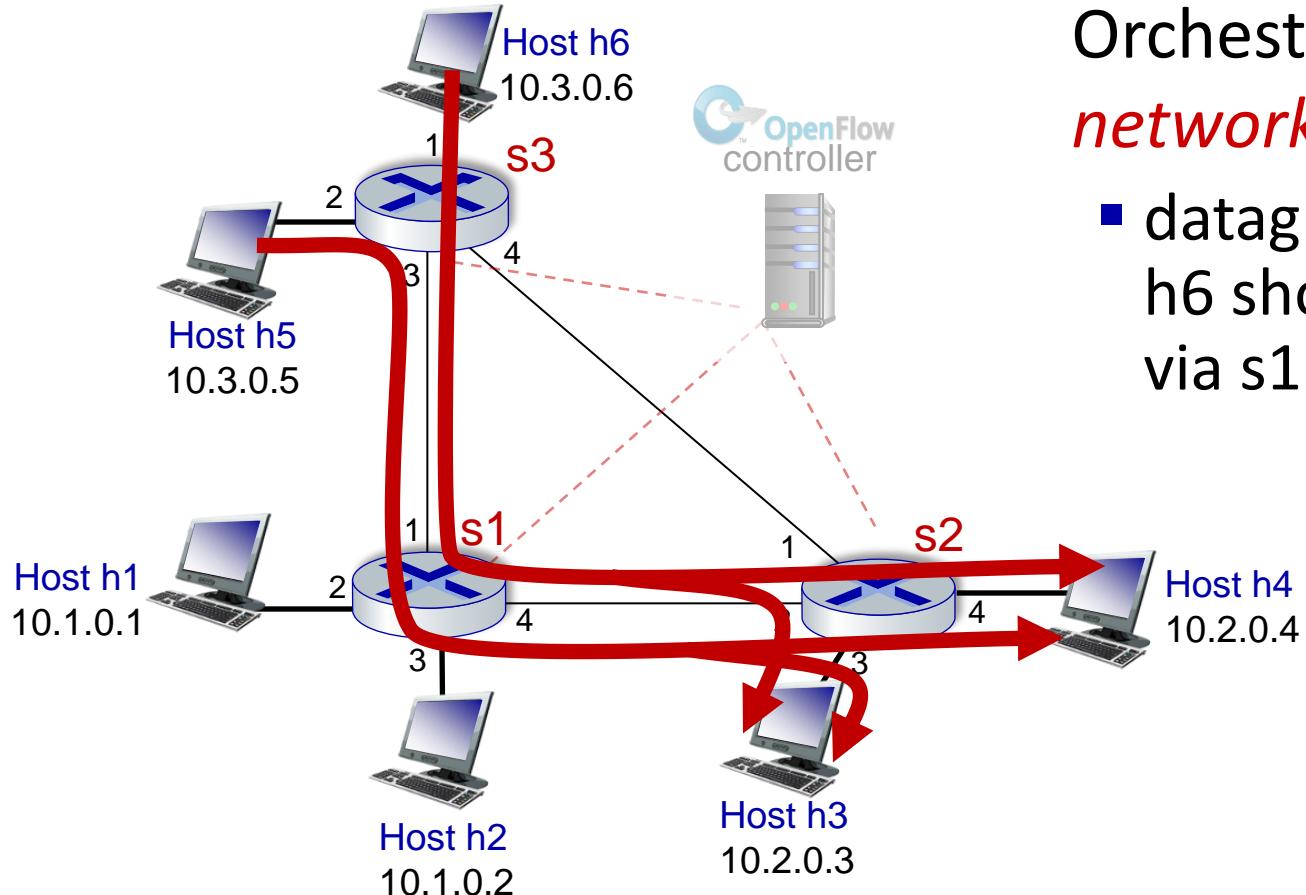
## Firewall

- *match:* IP addresses and TCP/UDP port numbers
- *action:* permit or deny

## NAT

- *match:* IP address and port
- *action:* rewrite address and port

# OpenFlow example

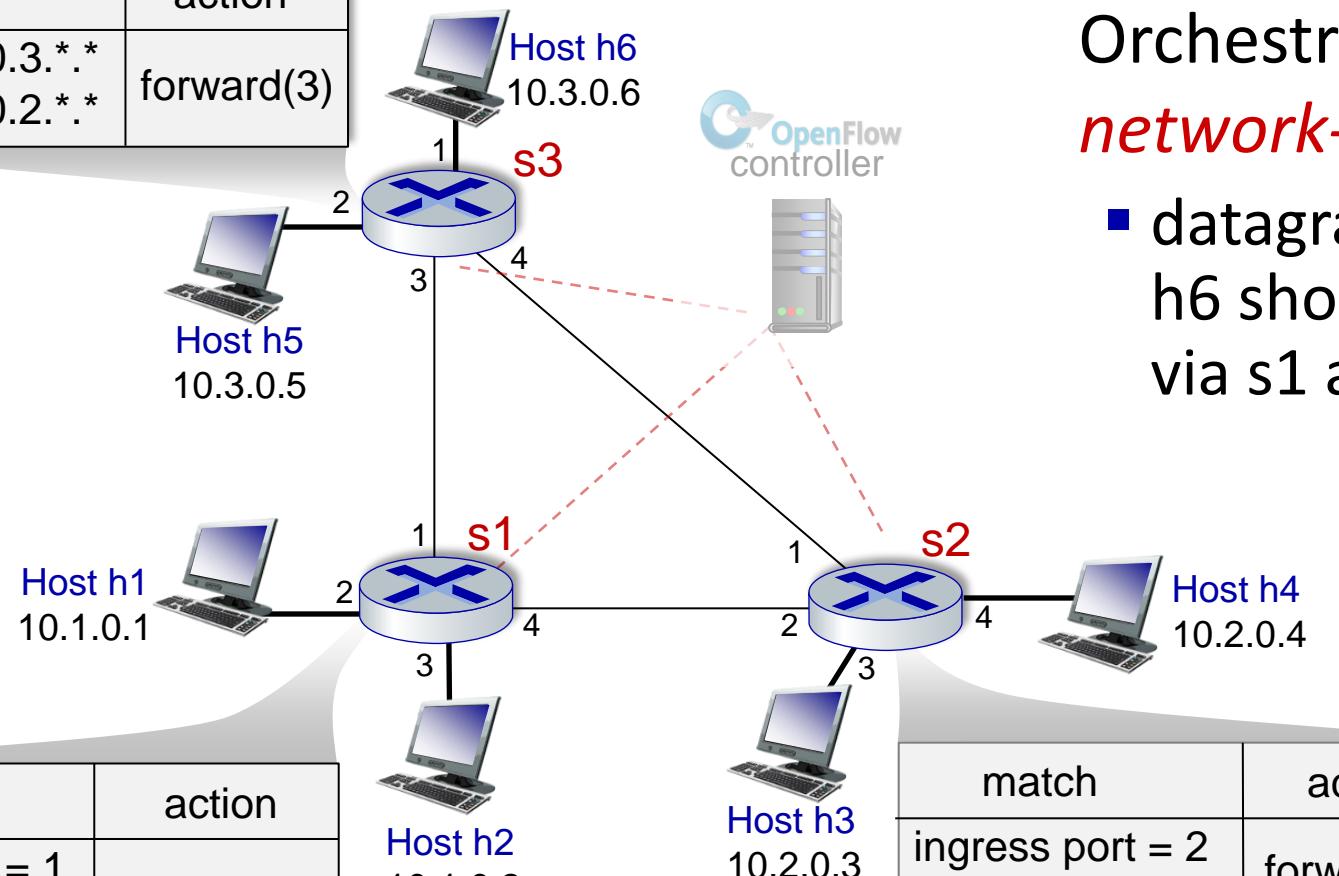


Orchestrated tables can create ***network-wide*** behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

# OpenFlow example

match	action
IP Src = 10.3.*.*	
IP Dst = 10.2.*.*	forward(3)



match	action
ingress port = 1	
IP Src = 10.3.*.*	forward(4)
IP Dst = 10.2.*.*	

Orchestrated tables can create *network-wide* behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

match	action
ingress port = 2	
IP Dst = 10.2.0.3	forward(3)
ingress port = 2	
IP Dst = 10.2.0.4	forward(4)

# Generalized forwarding: summary

- “**match plus action**” abstraction: match bits in arriving packet header(s) in any layers, take action
  - matching over many fields (link-, network-, transport-layer)
  - local actions: drop, forward, modify, or send matched packet to controller
  - “program” *network-wide* behaviors
- simple form of “network programmability”
  - programmable, per-packet “processing”
  - *historical roots*: active networking
  - *today*: more generalized programming:  
P4 (see p4.org).

# Network layer: “data plane” roadmap

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding
- Middleboxes
  - middlebox functions
  - evolution, architectural principles of the Internet



# Middleboxes

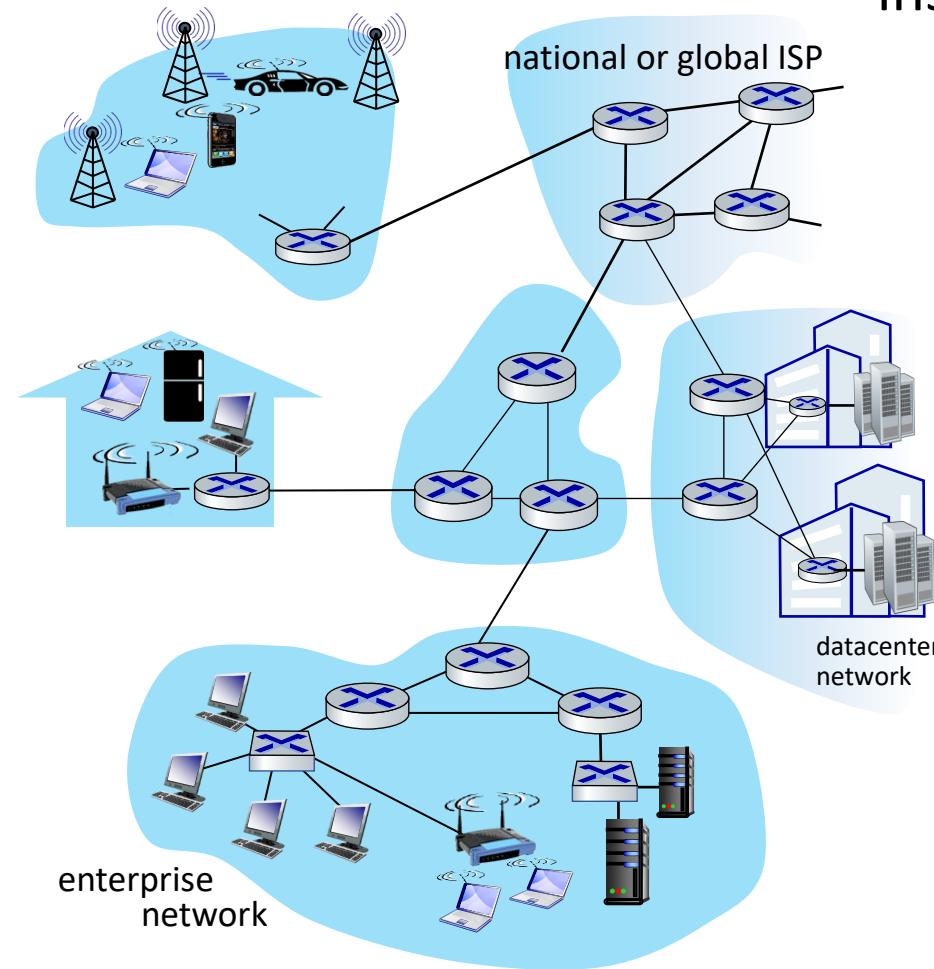
Middlebox (RFC 3234)

“any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host”

# Middleboxes everywhere!

NAT: home,  
cellular,  
institutional

Application-specific: service providers, institutional, CDN



Firewalls, IDS: corporate, institutional, service providers, ISPs

Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs

# Middleboxes

- initially: proprietary (closed) hardware solutions
- move towards “whitebox” hardware implementing open API
  - move away from proprietary hardware solutions
  - programmable local actions via match+action
  - move towards innovation/differentiation in software
- SDN: (logically) centralized control and configuration management often in private/public cloud
- network functions virtualization (NFV): programmable services over white box networking, computation, storage

# Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN
- Middleboxes



*Question:* how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

*Answer:* by the control plane (next chapter)