

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.

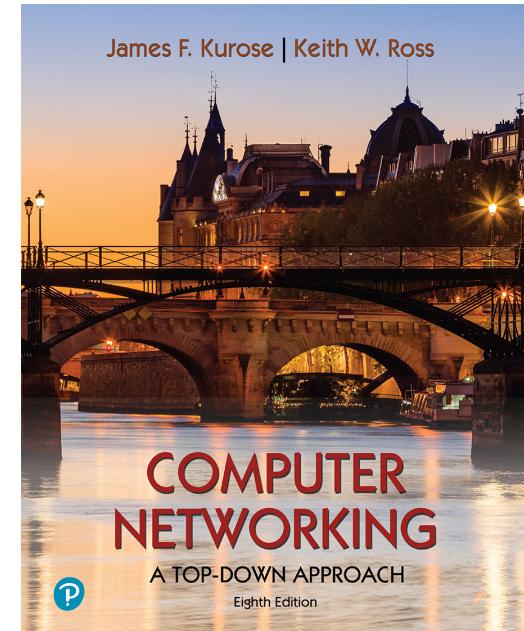
In return for use, we only ask the following:

- 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.

Thanks and enjoy! JFK/KWR

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

(Adapted, v1, 2024, pmc)



*Computer Networking: A
Top-Down Approach*

8th 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

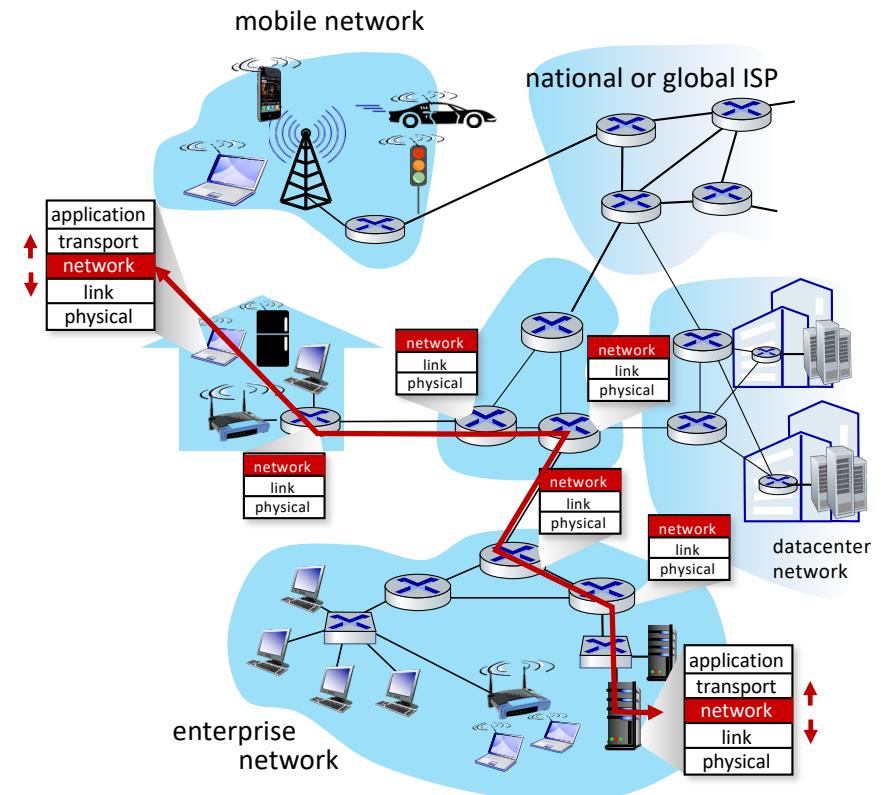
Network layer: “data plane” roadmap

- 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:**
 - examine header fields in all IP datagrams passing through it
 - move 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

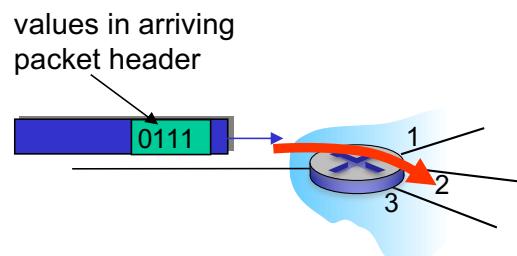


routing

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

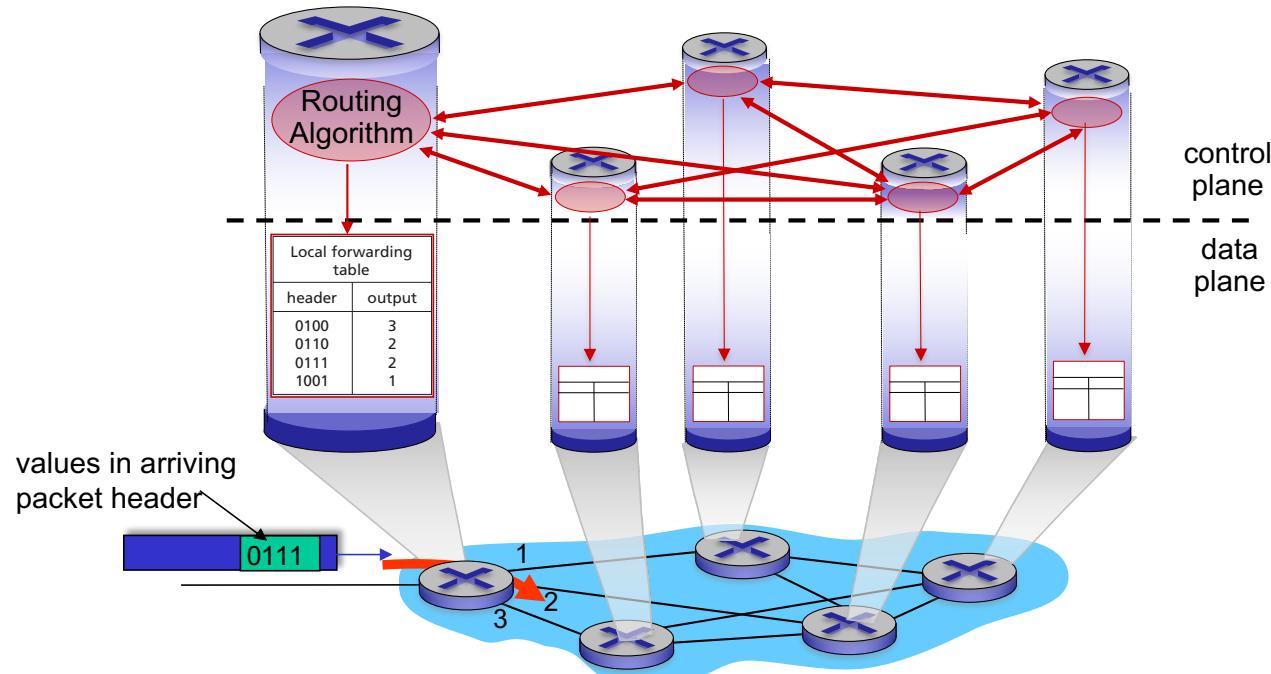


Control plane

- *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

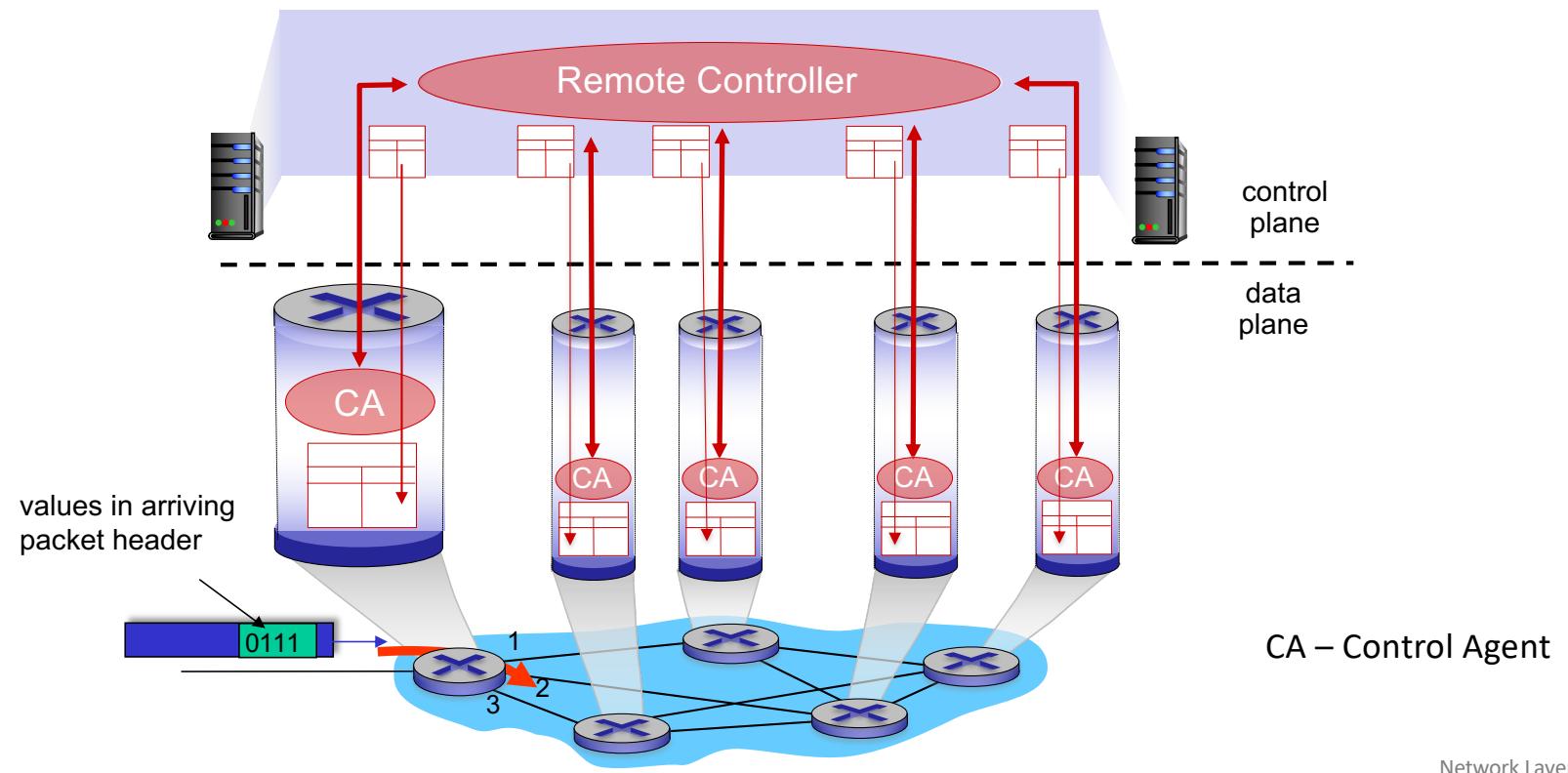
Per-router control plane

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

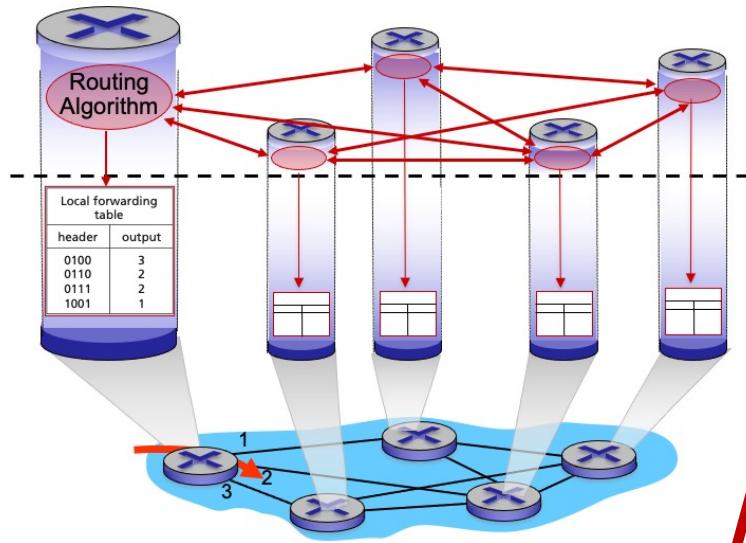


Software-Defined Networking - control plane

Remote controller computes, installs forwarding tables in routers

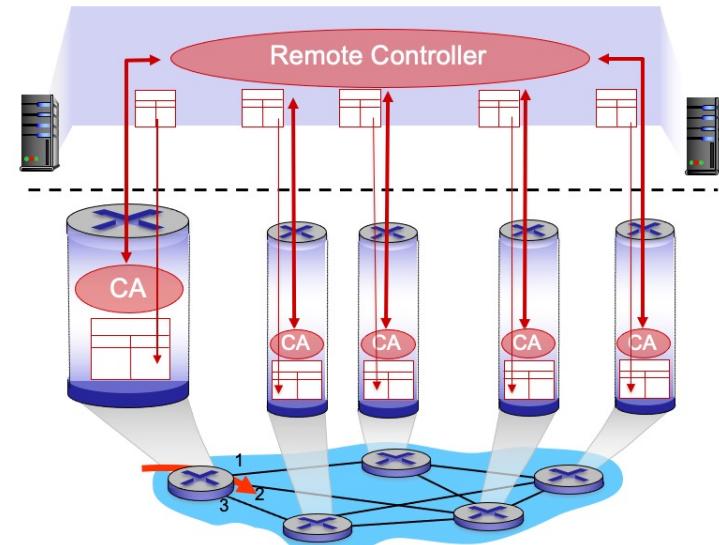


Per-router control plane



traditional approach, configuration per node, less flexible

SDN control plane



allows network management tasks automation, more flexible and customizable network configurations.

Network service model

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

example of services for *individual* datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay
- reality: no guarantees, **best effort** (default)

example of services for a *flow* of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing (delay variation aka jitter)
- reality: no notion of a flow (in IPv4)

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

global trend

Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed and 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

Network layer: “data plane” roadmap

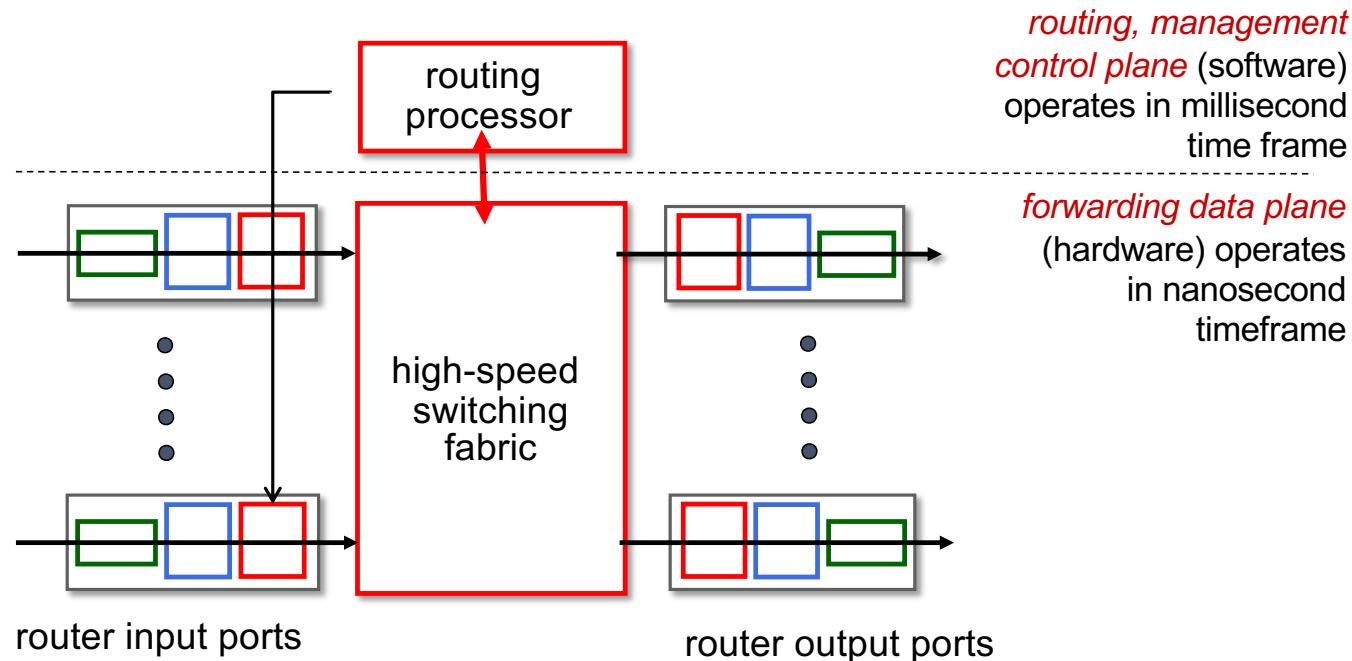
- Network layer: overview
 - data plane
 - control plane
- What's inside a router
 - input ports, switching, output ports
 - buffer management, scheduling
- IP: the Internet Protocol
 - datagram format
 - addressing
 - network address translation
 - IPv6



- Generalized Forwarding, SDN
 - Match+action
 - OpenFlow: match+action in action
- Middleboxes

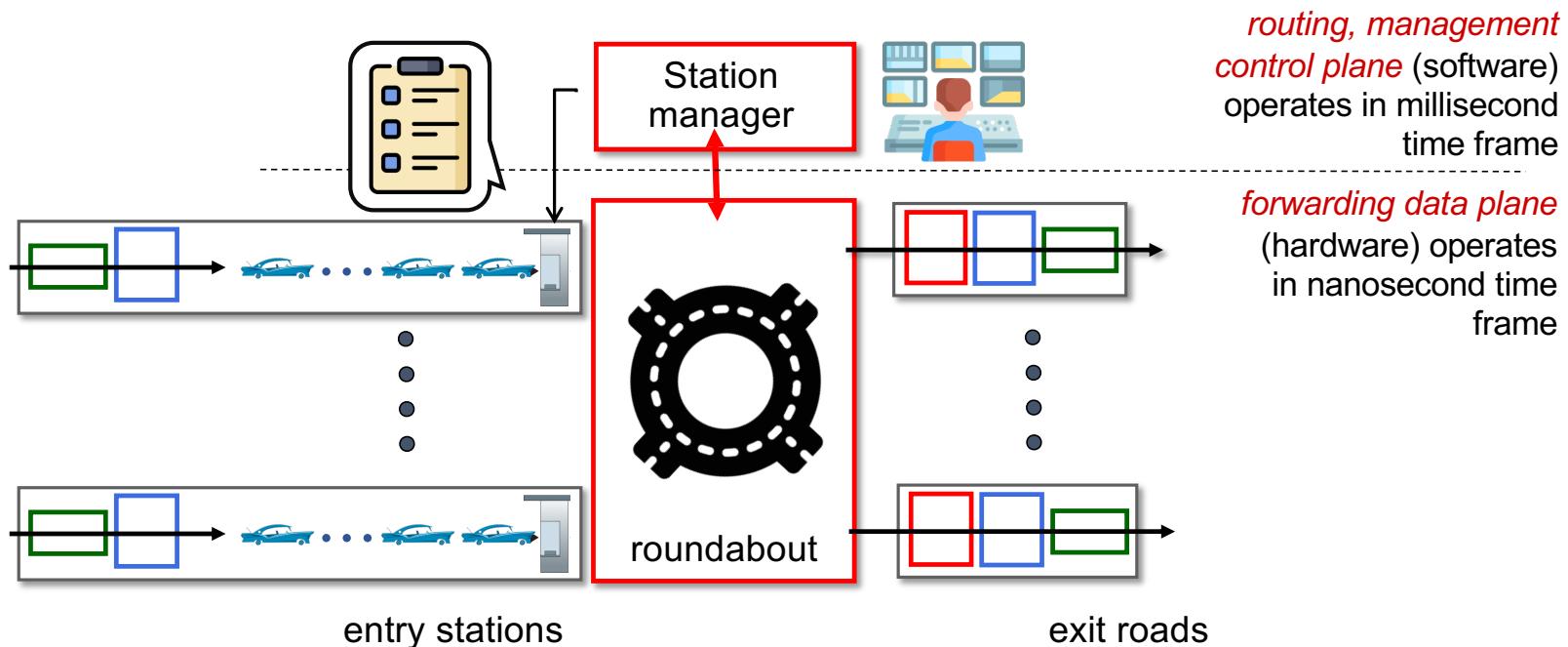
Router architecture overview

high-level view of generic router architecture:

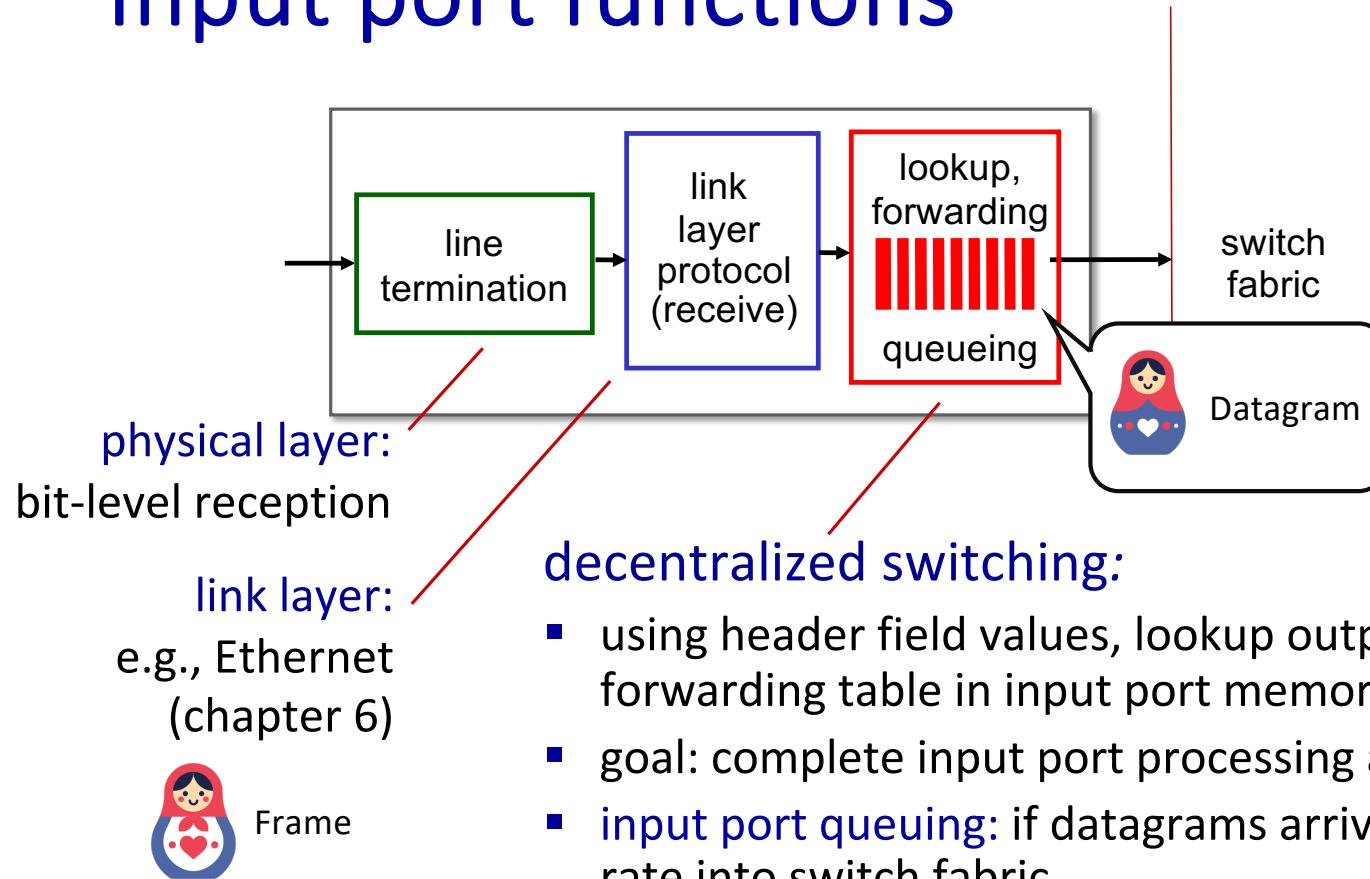


Router architecture overview

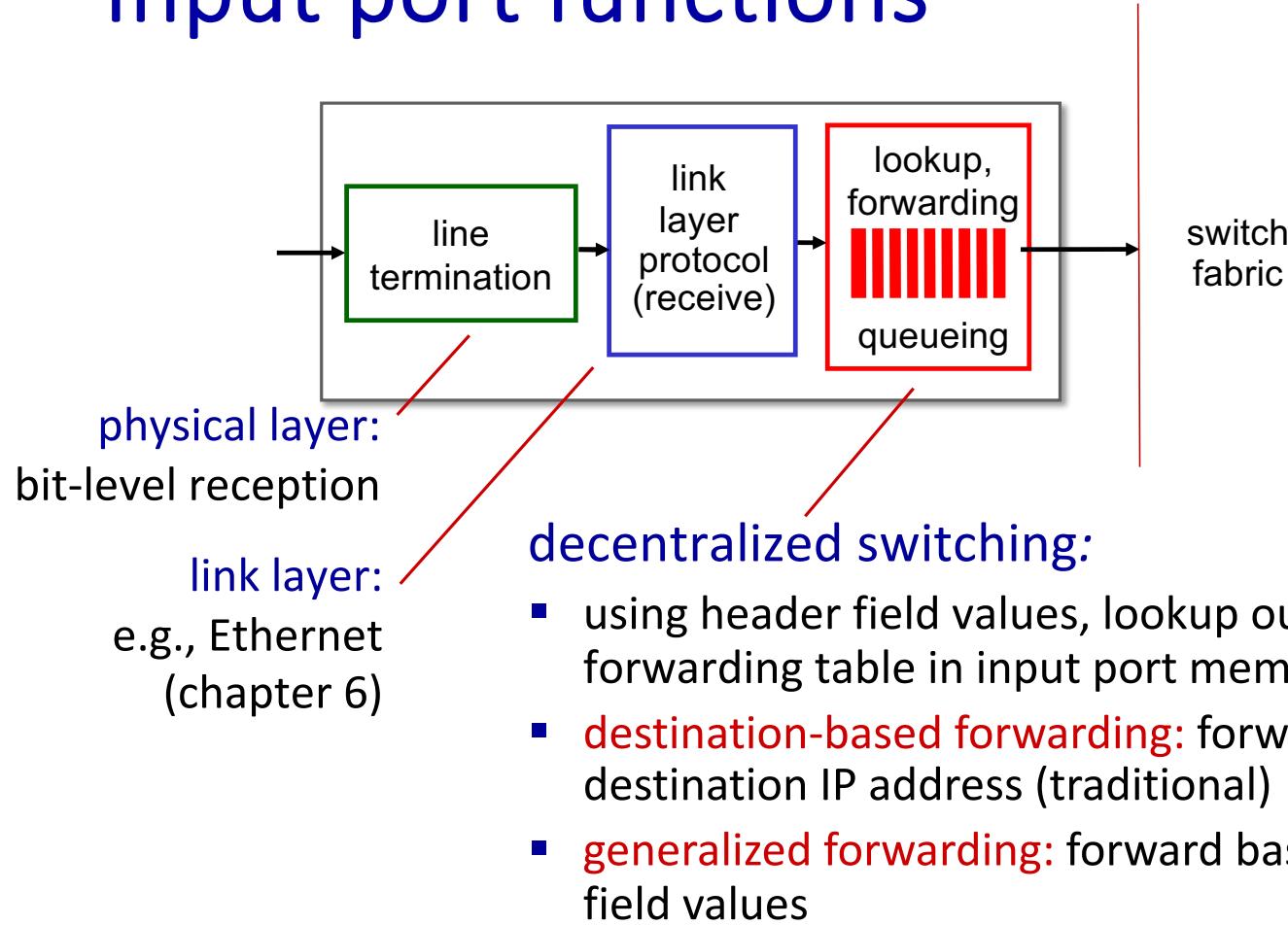
analogy view of generic router architecture:



Input port functions



Input port functions



- using header field values, lookup output port using forwarding table in input port memory (*"match plus action"*)
- **destination-based forwarding:** forward based only on destination IP address (traditional)
- **generalized forwarding:** forward based on any set of header field values

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

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

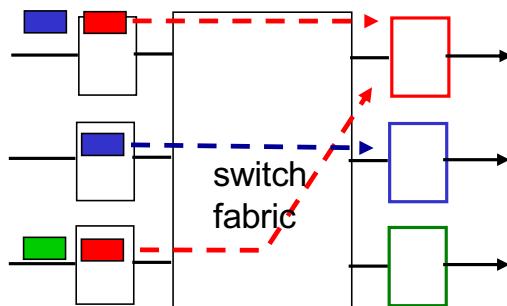
examples:

match!

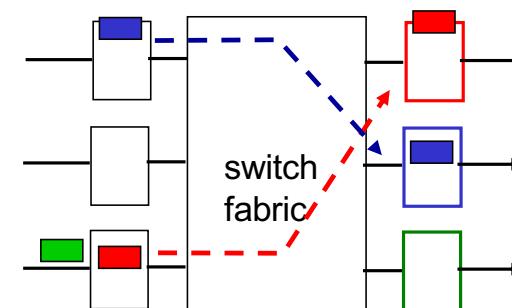
11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
 - queueing delay and loss due to input buffer overflow!
- **Head-of-the-Line (HOL) blocking:** queued datagram at front of queue prevents others in queue from moving forward



output port contention: only one red datagram can be transferred. lower red packet is *blocked*

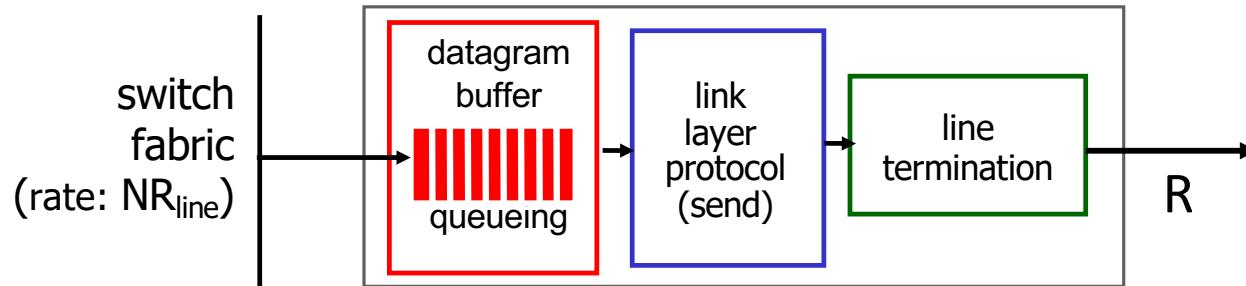


one packet time later: green packet experiences HOL blocking

Output port queuing



This is a really important slide



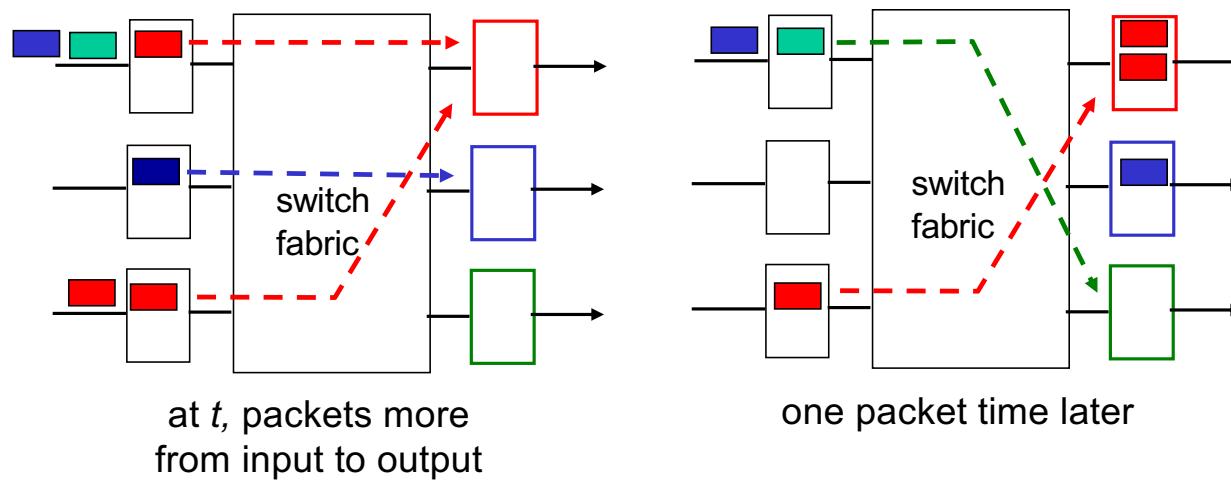
- **Buffering** required when datagrams arrive from fabric faster than link transmission rate. **Drop policy:** which datagrams to drop if no free buffers?

Datagrams can be lost due to congestion, lack of buffers

- **Scheduling discipline** chooses among queued datagrams for transmission

Priority scheduling – who gets best performance

Output port queuing



- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C

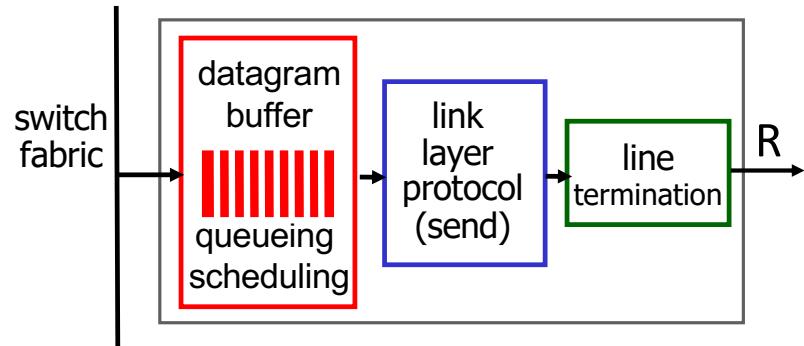
- e.g., $C = 10 \text{ Gbps}$ link $\rightarrow 2.5 \text{ Gbit buffer}$

- more recent recommendation: with N flows, buffering equal to

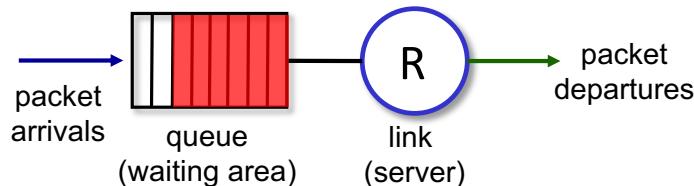
$$\frac{\text{RTT} \cdot C}{\sqrt{N}}$$

- but *too* much buffering can increase delays (particularly in home routers)
 - long RTTs: poor performance for real-time apps, sluggish TCP response
 - recall delay-based congestion control: “keep bottleneck link just full enough (busy) but no fuller”

Buffer Management



Abstraction: queue



buffer management:

- **drop**: which packet to add, drop when buffers are full
 - **tail drop**: drop arriving packet
 - **priority**: drop/remove on priority basis
- **marking**: which packets to mark to signal congestion (ECN, RED)

Packet Scheduling: FCFS

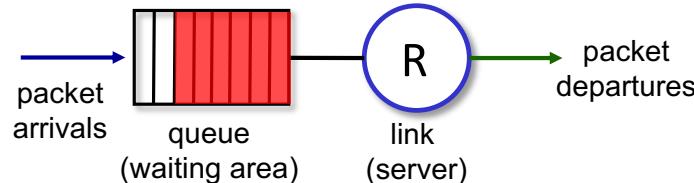
packet scheduling: deciding which packet to send next on link

- first come, first served
- priority
- round robin
- weighted fair queueing

FCFS: packets transmitted in order of arrival to output port

- also known as: First-in-first-out (FIFO)
- real world examples?

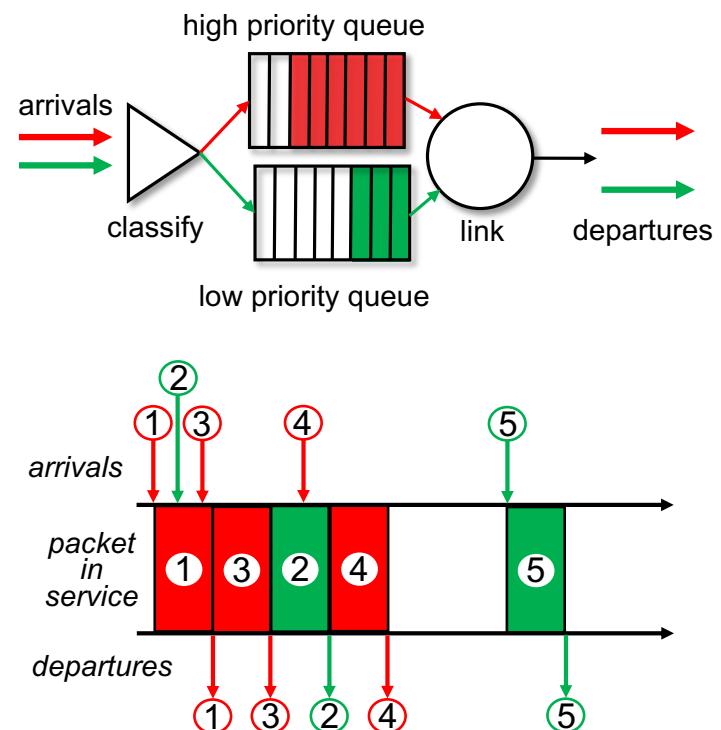
Abstraction: queue



Scheduling policies: priority

Priority scheduling:

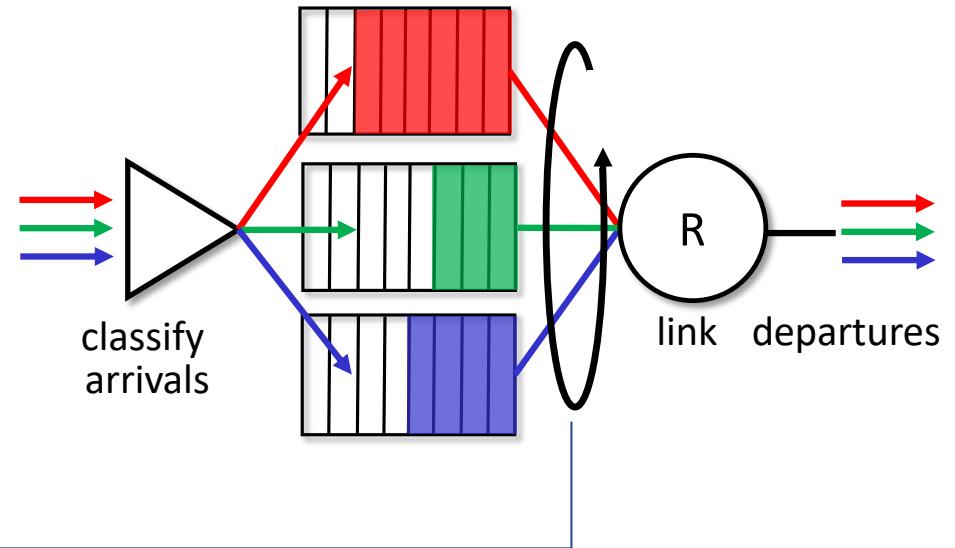
- arriving traffic classified, queued by class
 - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
 - FCFS within priority class



Scheduling policies: round robin

Round Robin (RR) scheduling:

- arriving traffic classified, queued by class
 - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



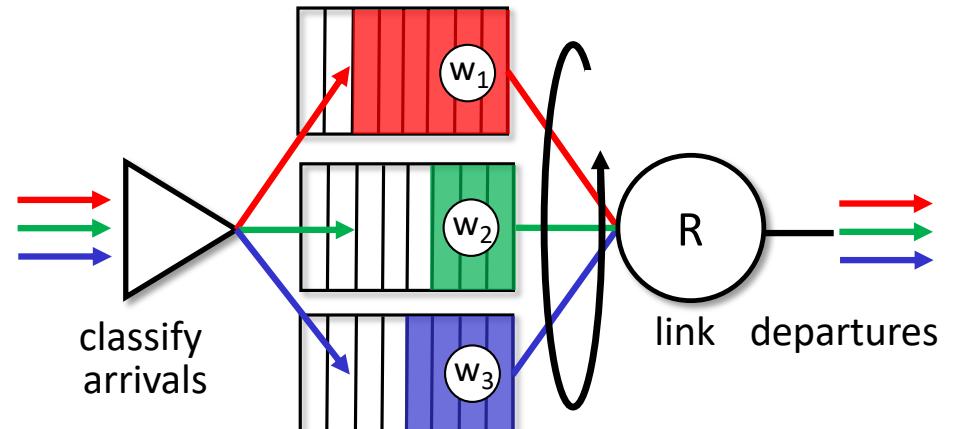
Scheduling policies: weighted fair queueing

Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class, i , has weight, w_i , and gets weighted amount of service in each cycle:

$$\frac{w_i}{\sum_j w_j}$$

- minimum bandwidth guarantee (per-traffic-class)



Network layer: “data plane” roadmap

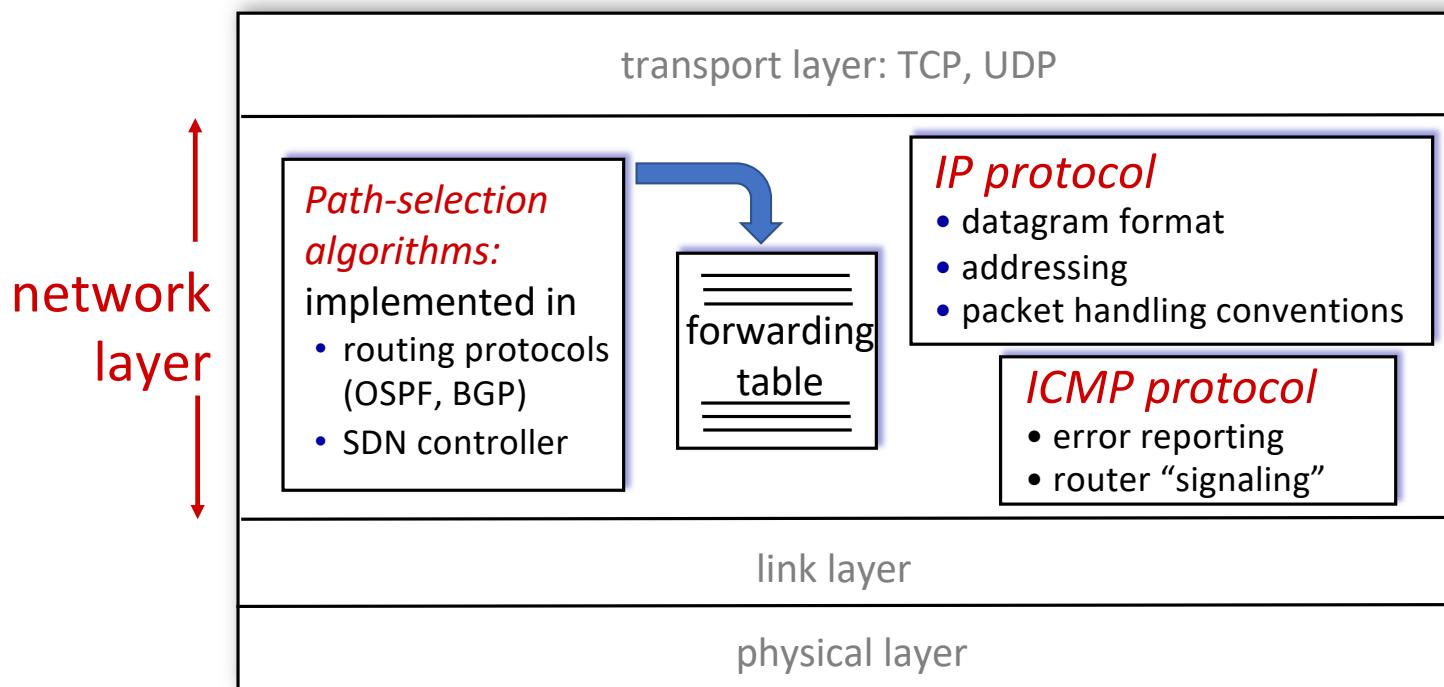
- Network layer: overview
 - data plane
 - control plane
- What's inside a router
 - input ports, switching, output ports
 - buffer management, scheduling
- IP: the Internet Protocol
 - datagram format
 - addressing
 - network address translation
 - IPv6



- Generalized Forwarding, SDN
 - match+action
 - OpenFlow: match+action in action
- Middleboxes

Network Layer: Internet

host, router network layer functions:



Network Layer: ICMP

Internet Control Message Protocol

- ❖ used by hosts & routers to communicate network-level information
 - error reporting: unreachable host, network, port, protocol
 - echo request/reply (used by ping)
- ❖ network-layer “above” IP:
 - ICMP msgs carried in IP datagrams
- ❖ **ICMP message:** type, code, checksum, description (packet hdr, occurrence or error)
- ❖ ICMPv4 – RFC792
- ❖ ICMPv6 – RFC4443

Type(8)	Code(8)	Description (32 bits)
0	0	echo reply (ping)
3	0	dest network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header
13	0	timestamp
14	0	timestamp reply
(...)		

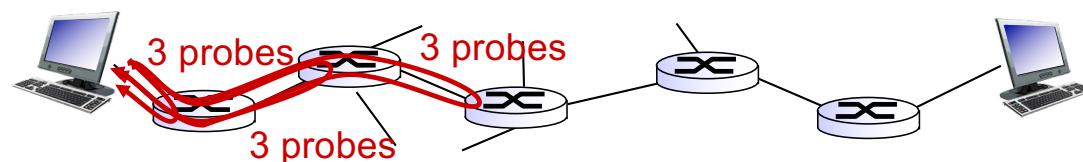
(partial list)

Network Layer: traceroute and ICMP

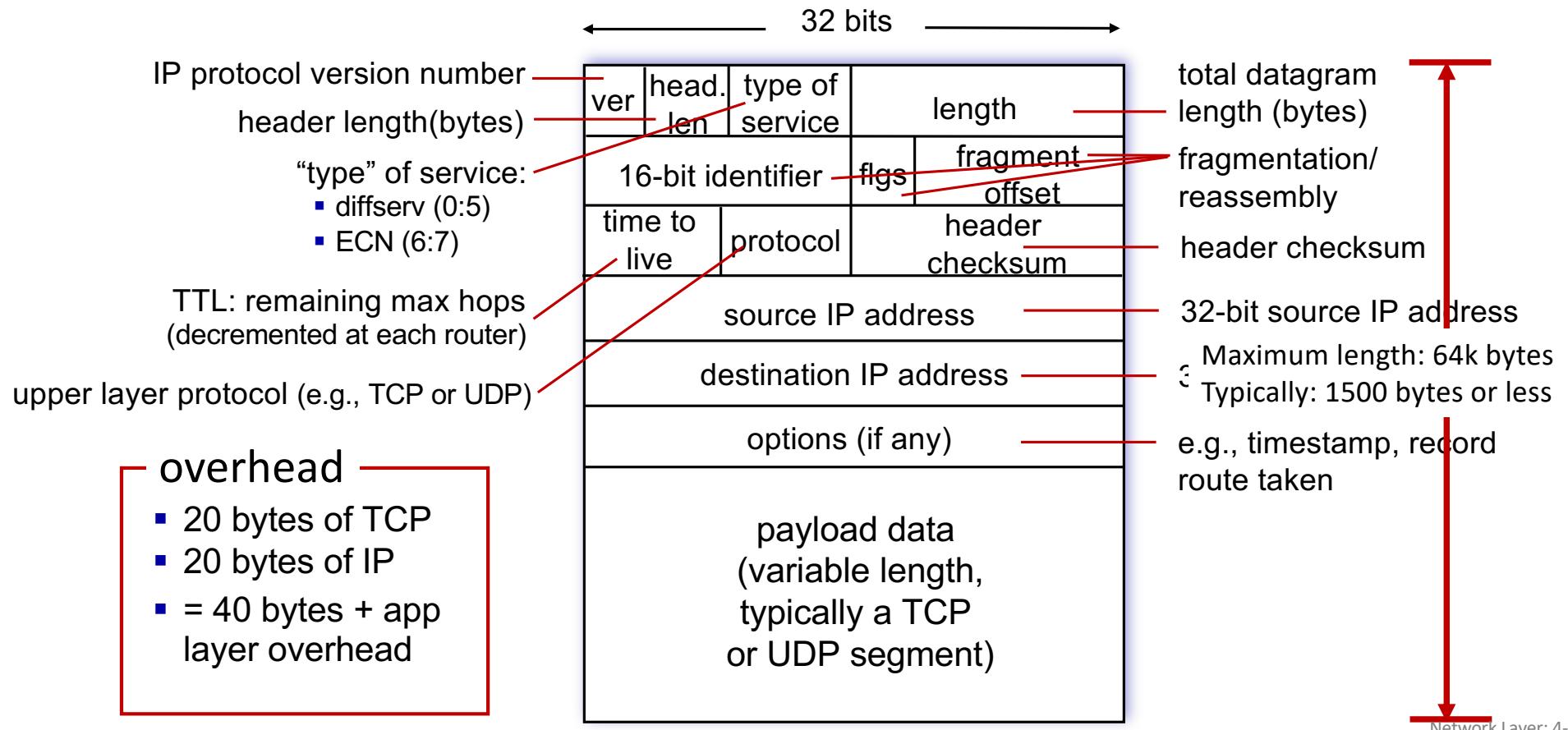
- ❖ source sends series of UDP segments (or ICMP with flag -I) to dest
 - first set has TTL = 1
 - second set has TTL=2, etc.
 - unlikely port number
- ❖ when n^{th} set of datagrams arrives to n^{th} router:
 - router discards datagrams
 - and sends to source ICMP messages (type 11, code 0)
 - ICMP messages includes name of router & IP address
- ❖ when ICMP messages arrive, source records RTTs

stopping criteria:

- ❖ UDP segment eventually arrives at destination host
- ❖ destination returns ICMP “port unreachable” message (type 3, code 3)
- ❖ source stops

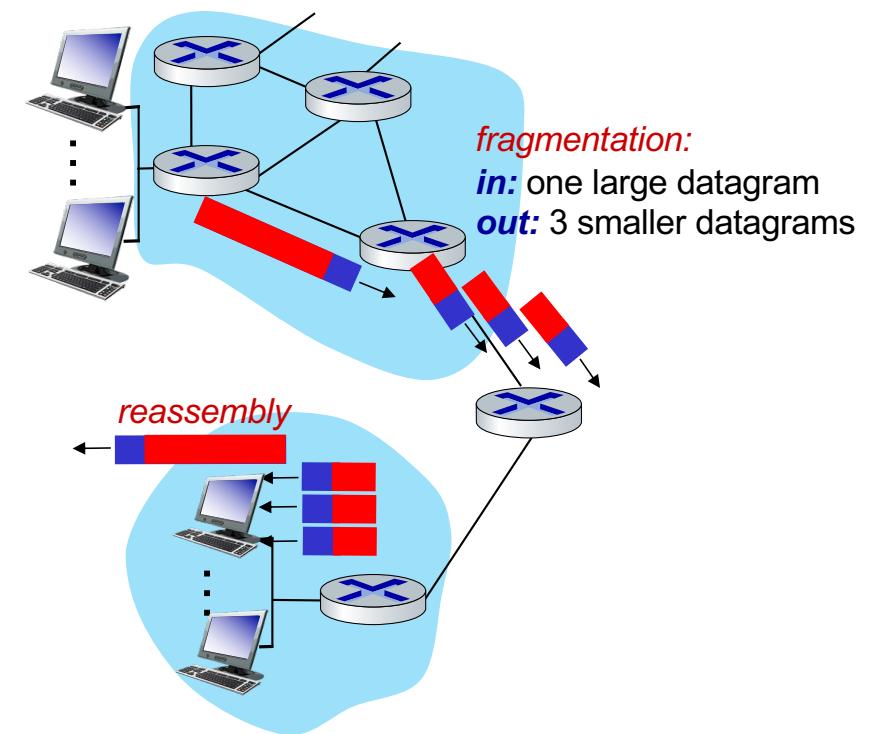


IP Datagram format



IP fragmentation/reassembly

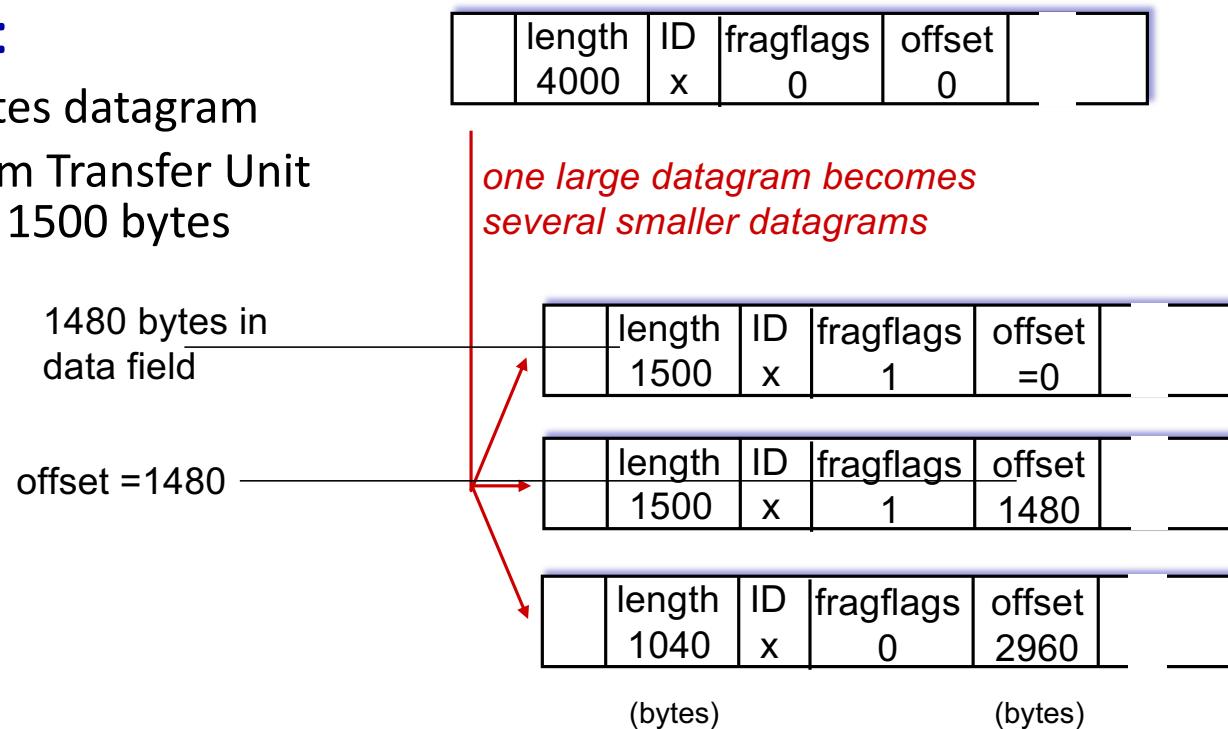
- network links have MTU (max. transfer unit/size) - largest possible link-level frame
 - different link types -> different MTUs
- large IP datagram divided (“fragmented”) within network
 - one datagram becomes several datagrams, whenever needed
 - “reassembled” only at *destination*
 - IP header fields (identifier, flags, fragment offset) used to identify and order related fragments



IP fragmentation/reassembly

example:

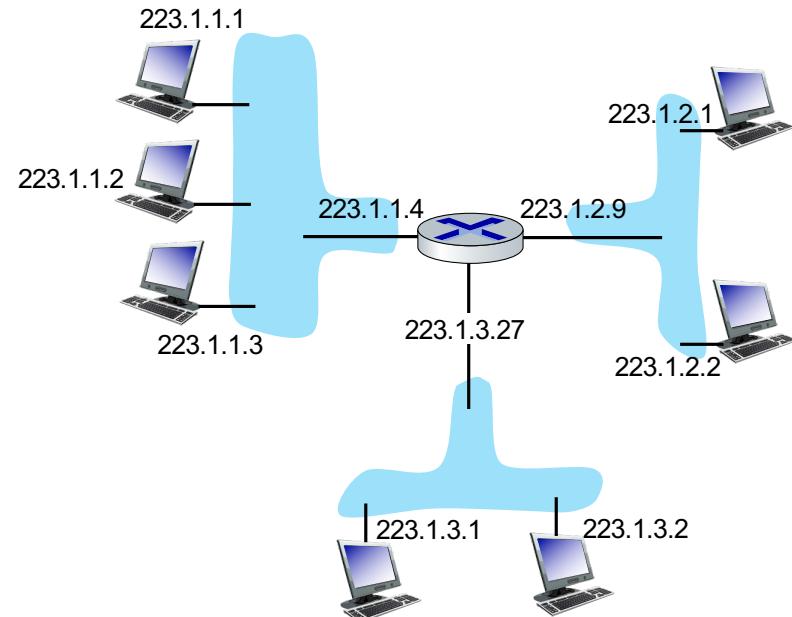
- 4000 bytes datagram
- Maximum Transfer Unit (MTU) = 1500 bytes



(fragment offset: starting position of the data in the fragment in relation to the start of the data in the original packet/datagram)

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)
 - host with two or more active IP addresses is called *multihomed*



dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1
Network Layer: 4-50

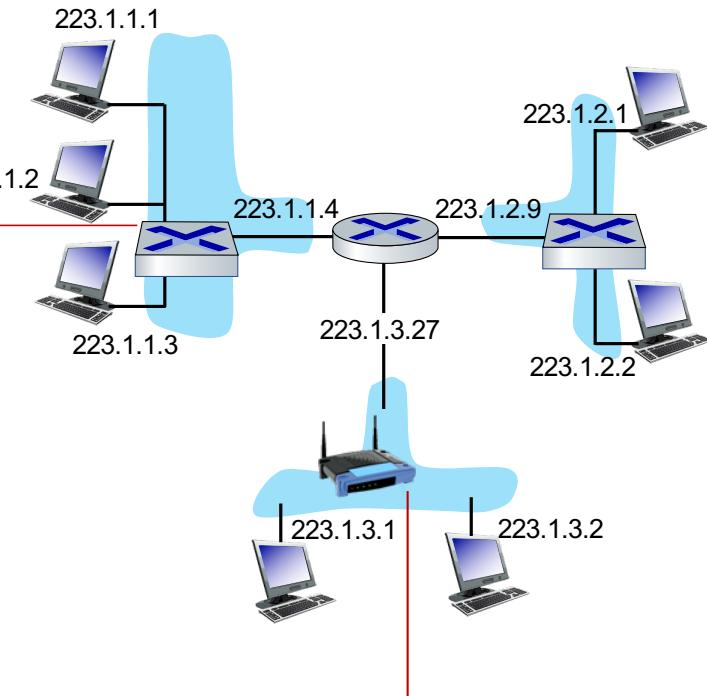
IP addressing: introduction

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

IP addressing: introduction

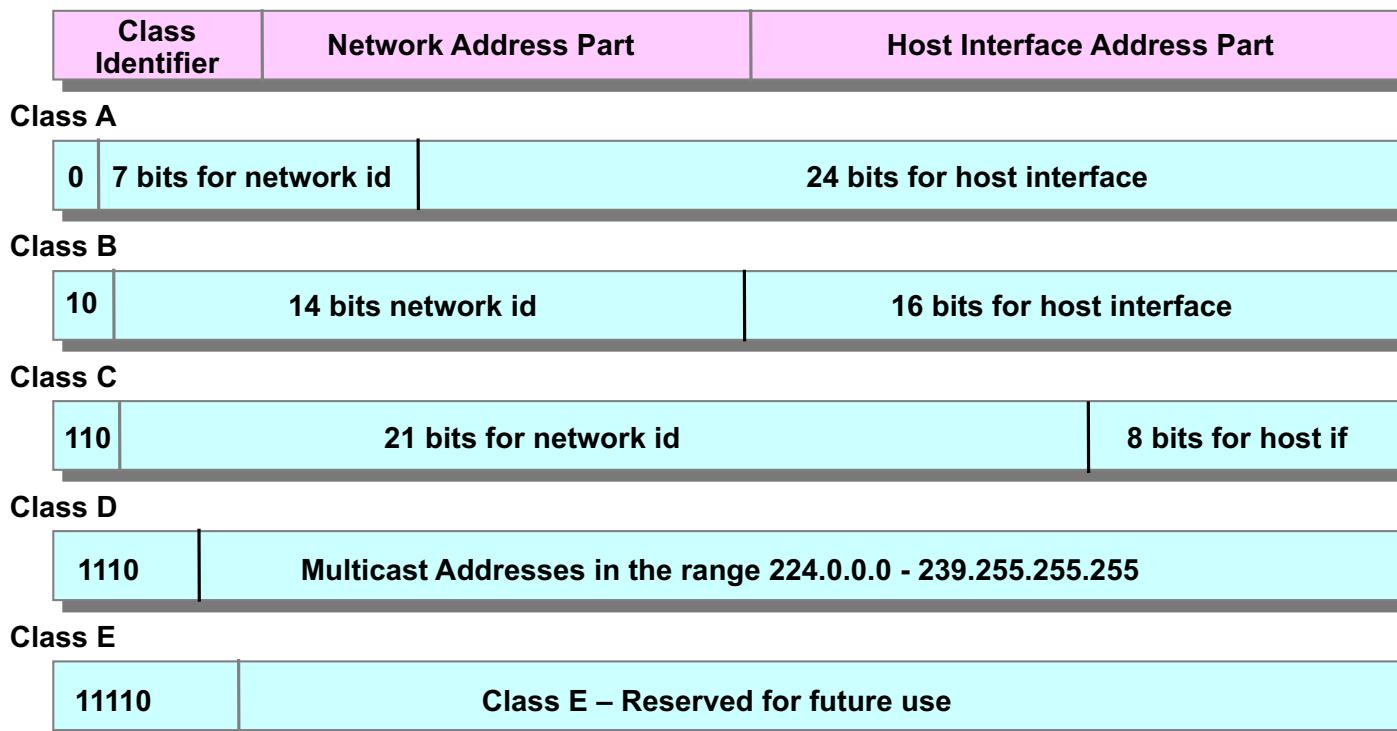
IPv4: 32-bit *unsigned binary value*

xxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx

(dot decimal notation)

- part of the address identifies the **network** (or subnet) and part identifies the **host (interface)** in that network
<network id><host_interface id>
- in the Internet, each IP address has to be unique
- originally distributed in five classes (A to E)
- assigned by ICANN (*Internet Corporation for Assigned Names and Numbers*), in a first instance, delegated by a registered authority/organization (Regional Internet Registries (RIRs))

IP addressing: original scheme



IP addressing: Classful vs Classless Scheme

Addressing based on classes (**Classful**)

- original scheme, based on RFC 791
- the first bits identify the class of addresses (hardcoded)

Addressing without classes (**Classless**)

- does not consider the initial bits for class id; a **network mask** (of 32 bits) is used to determine the network address (class bits may be used to define a default network mask, if not specified)
- allows aggregation of addresses to increase routing efficiency
CIDR (*Classless Internet Domain Routing*)
- aggregation reduces the size of routing tables by aggregating groups of adjacent network addresses (aka network prefixes)
- used massively by ISPs to downsize routing tables

IP addressing: Classless Scheme (CIDR)

Network Mask

- Bit pattern conjugated with the IP address gives the network id (prefix). It works as a **filter** applied over the IP address.
- Specified commonly as **a.b.c.d/n**
- If not specified, the following masks are used (class compatible):

(Class A) 1111111.00000000.00000000.00000000
decimal notation: 255.0.0.0 CIDR notation: /8

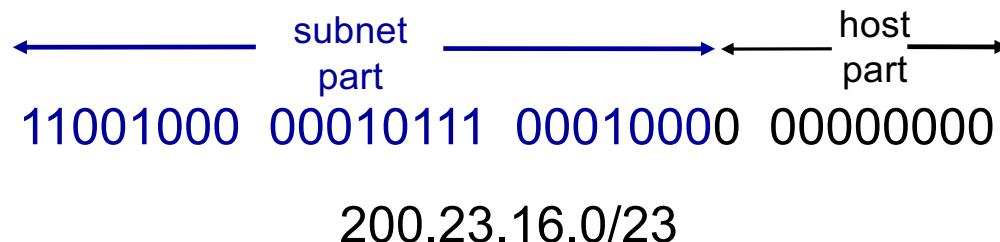
(Class B) 11111111.11111111.00000000.00000000
decimal notation : 255.255.0.0 CIDR notation: /16

(Class C) 11111111.11111111.11111111.00000000
decimal notation: 255.255.255.0 CIDR notation: /24

IP addressing: Classless Scheme (CIDR)

CIDR: Classless InterDomain Routing (pronounced cider)

- subnet portion of address of **arbitrary** length, which allows creating **subnets** of the original network prefix or **supernets** (aggregation of networks prefixes)
- address format: **a.b.c.d/n**, where *n* is the number of bits in subnet portion of address



IP addressing: Classless Scheme (CIDR)

Consider the IP address **130.1.5.1** (former class B as it starts with bits **10000010**)

- without specifying /n, it corresponds to host interface **5.1** in the network **130.1.0.0** (originally a class B), i.e., the default mask is: **255.255.0.0 or /16**

Consider the IP address **130.1.5.1/24**

- as /n is specified, it corresponds to host **1** in subnet **130.1.5.0**
- subnets IDs are defined in the initial addressing space of host ID
 $<\text{network id}> <<\text{subnet id}> \text{host_if id}>$

<u>network</u>	<u>host</u>
130.1	5.1

original per class interpretation

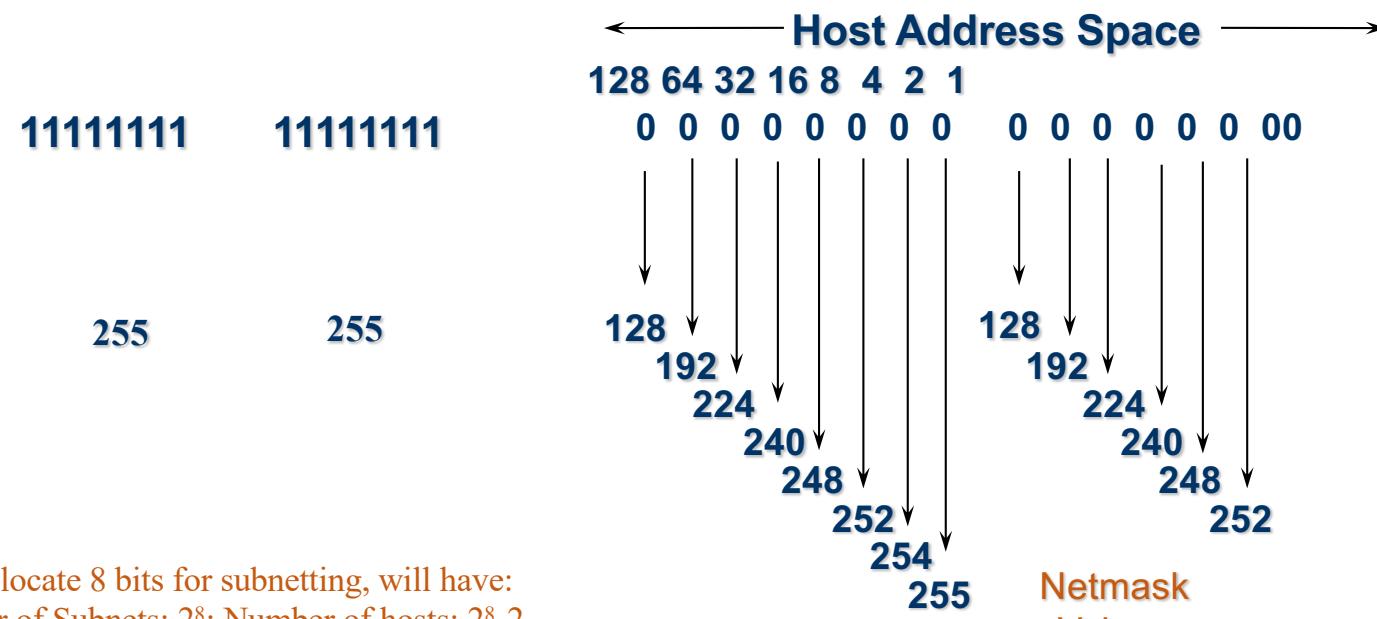
<u>network mask</u>
255.255.255.0

<u>network</u>	<u>subnet</u>	<u>host</u>
130.1	5	1

classless interpretation

IP addressing: Classless Scheme (CIDR)

Example of possible subnet masks for a /16 network prefix (Class B)



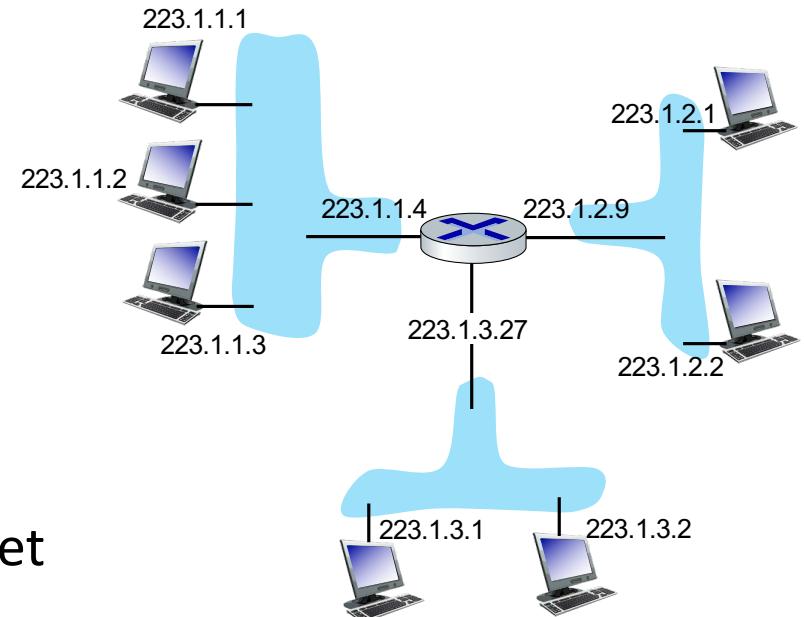
IP addressing: Subnets

■ 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

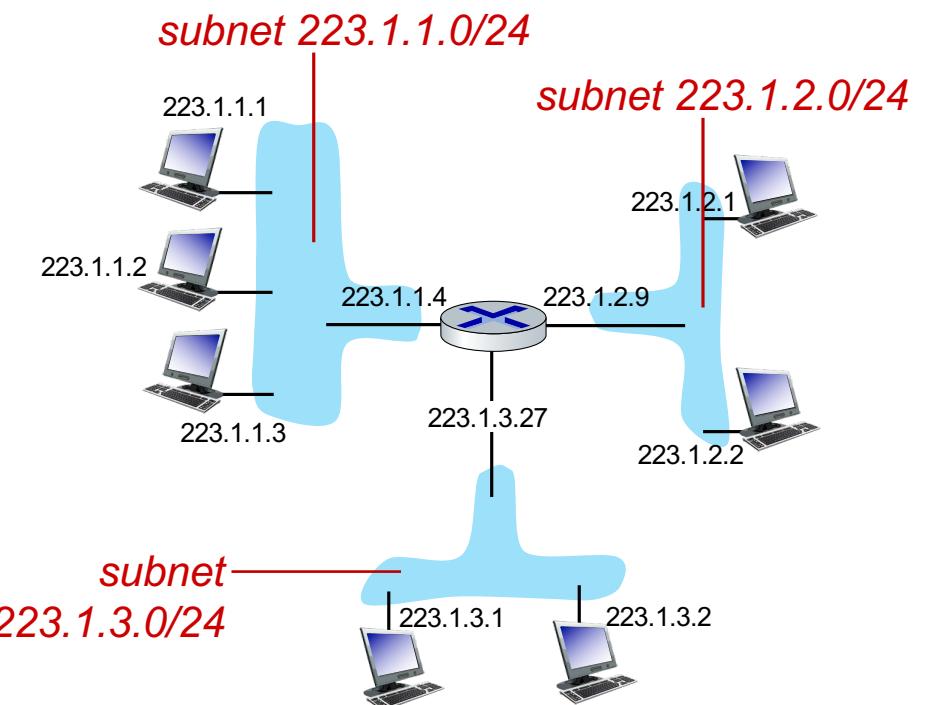


network consisting of 3 subnets

IP addressing: Subnets

Recipe for defining subnets:

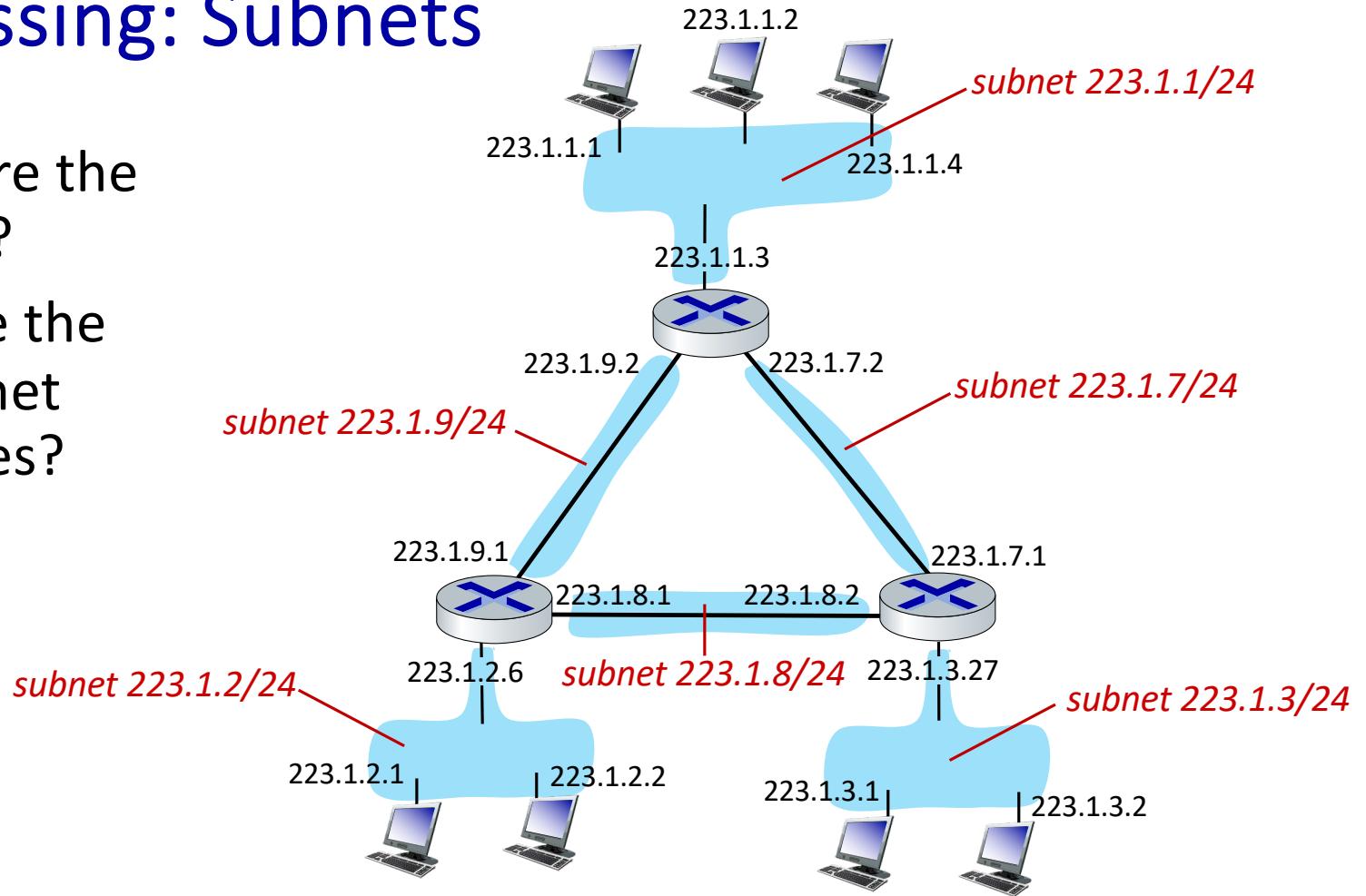
- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*



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

IP addressing: Subnets

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



IP addressing: Subnets

Advantages

- allows better organization of the available addressing space
- allows to establish hierarchical levels for routing (we'll develop soon)

Costs

- reduces the addressing space for host interfaces, as some of the initial addresses cannot be used with the same purpose.
- requires additional addressing management

IP addressing: Reserved, Private Addresses

Reserved IP addresses

- 127.x.x.x reserved for loopback interfaces
- host interface id with all 0s (any host) or all 1s (all hosts)

Range of Private Addresses

- block 192.168.0.0 – 192.168.255.255 /16 (2^{16} addresses)
- block 172.16.0.0 – 172.31.255.255 /12 (2^{20} addresses)
- block 10.0.0.0 – 10.255.255.255 /8 (2^{24} addresses)
- there's no IP routing in the Internet for private addresses

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 a *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 a server
 - “plug-and-play”

DHCP: Dynamic Host Configuration Protocol

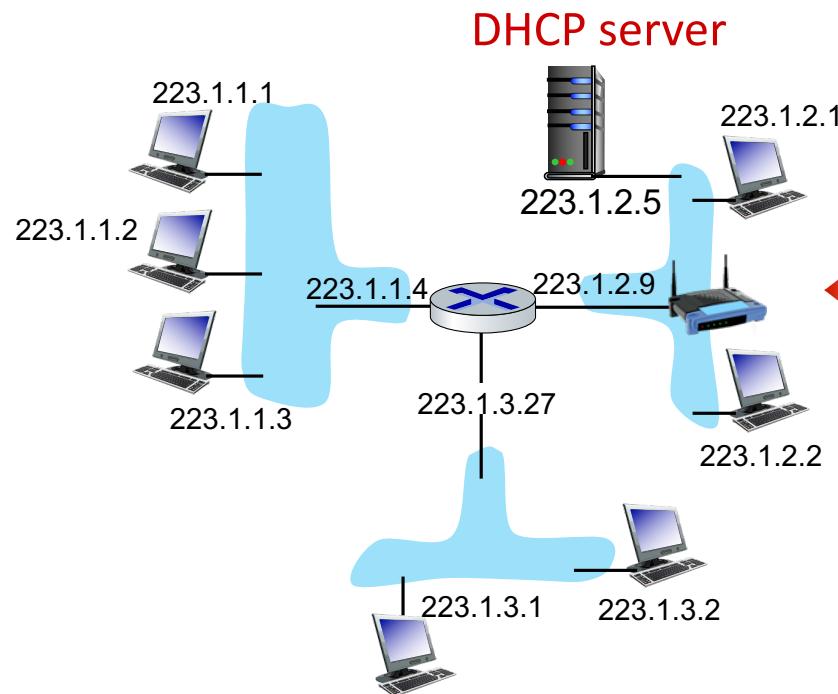
goal: host *dynamically* obtains IP address from network server when it “joins” network

- 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

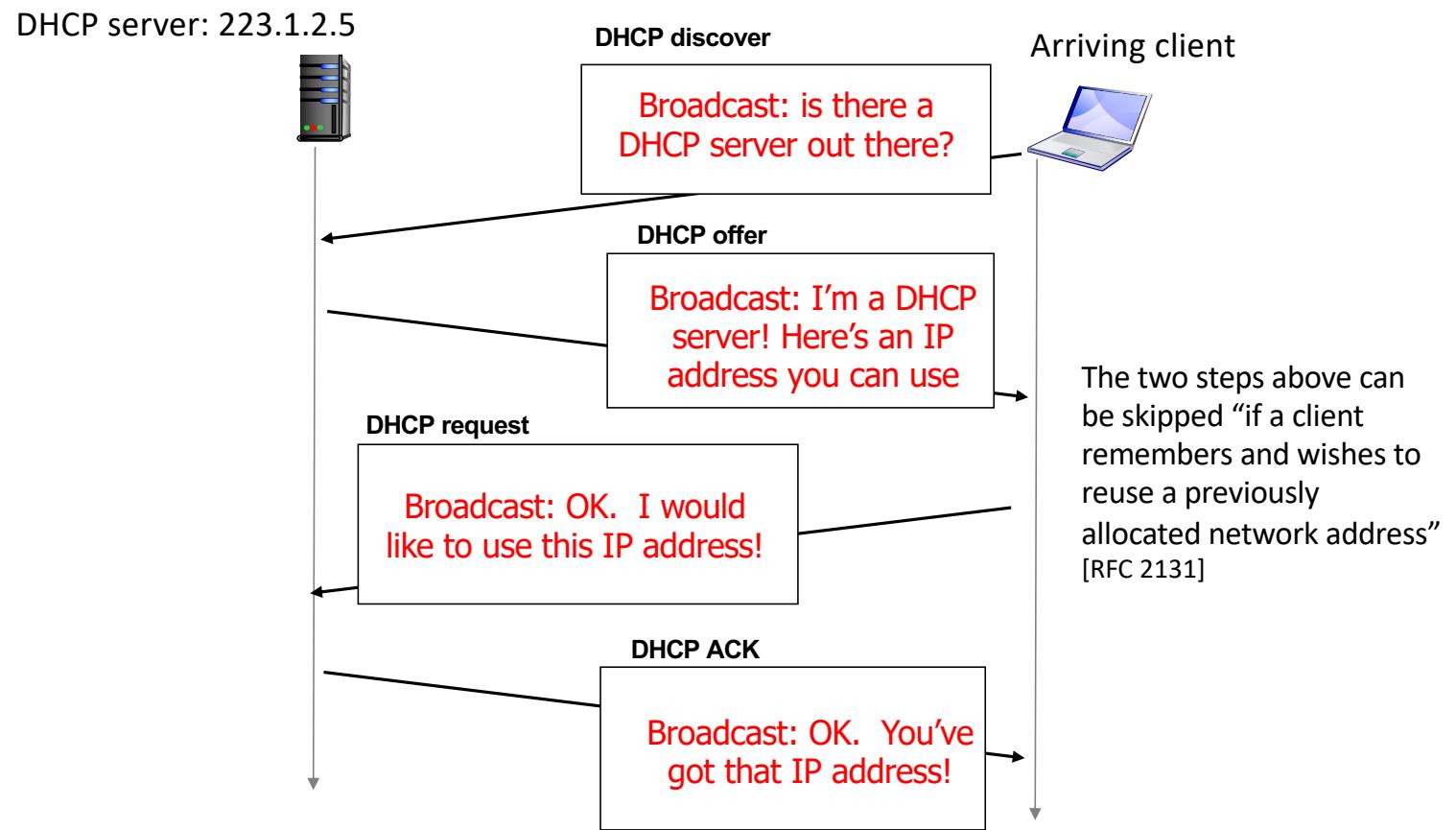


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached



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

DHCP client-server scenario

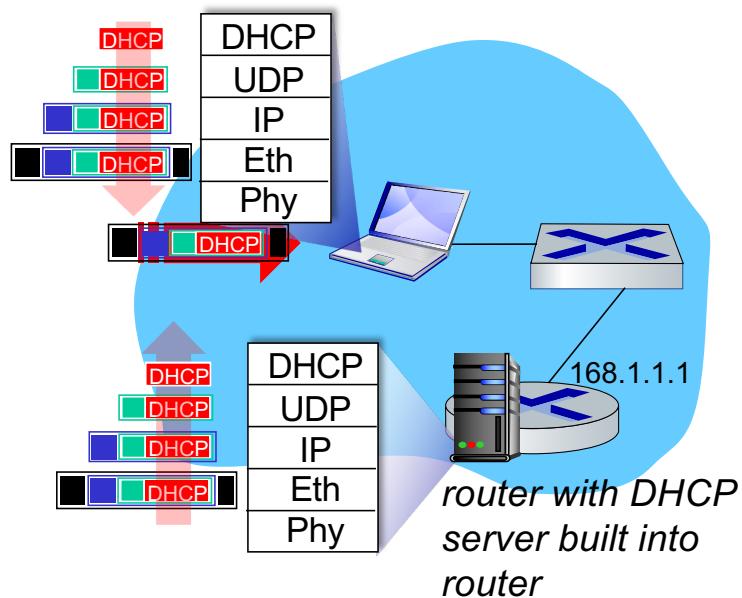


DHCP: more than IP addresses

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

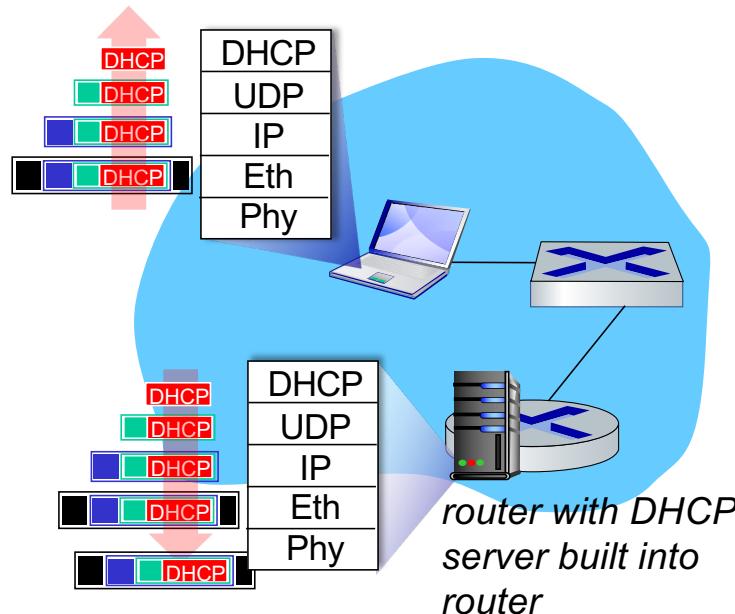
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

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



- DHCP 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 a *network* get subnet part of IP address?

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

ISP's block 11001000 00010111 00010000 00000000 200.23.16.0/20

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

Organization 0 11001000 00010111 00010000 00000000 200.23.16.0/23

Organization 1 11001000 00010111 00010010 00000000 200.23.18.0/23

Organization 2 11001000 00010111 00010100 00000000 200.23.20.0/23

...

.....

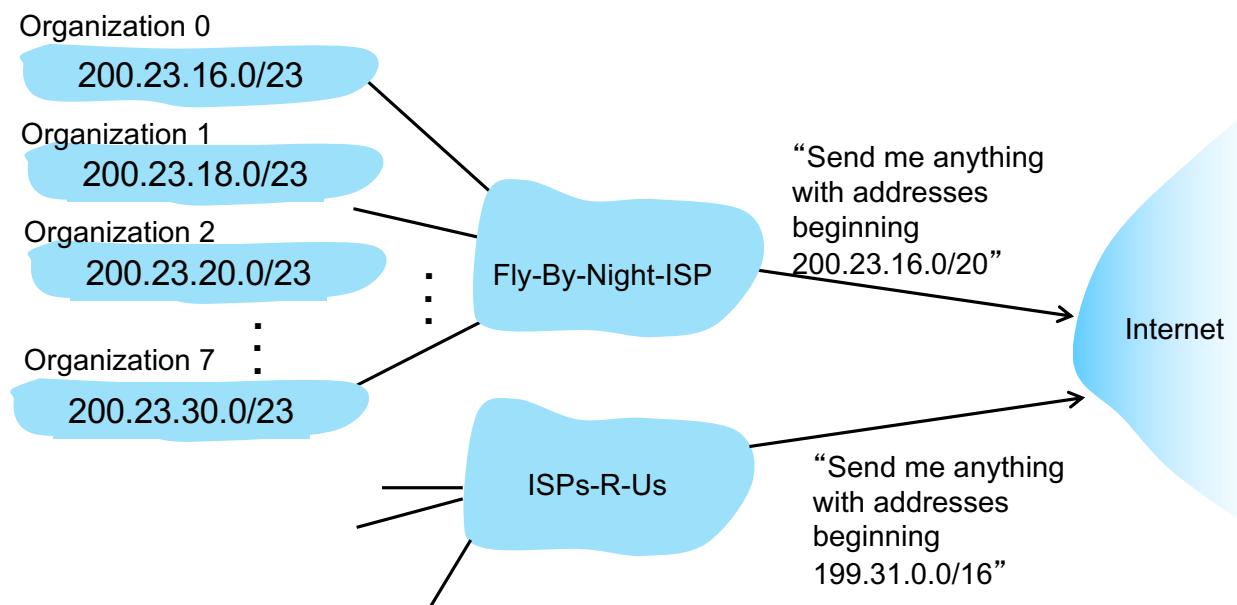
....

....

Organization 7 11001000 00010111 00011110 00000000 200.23.30.0/23

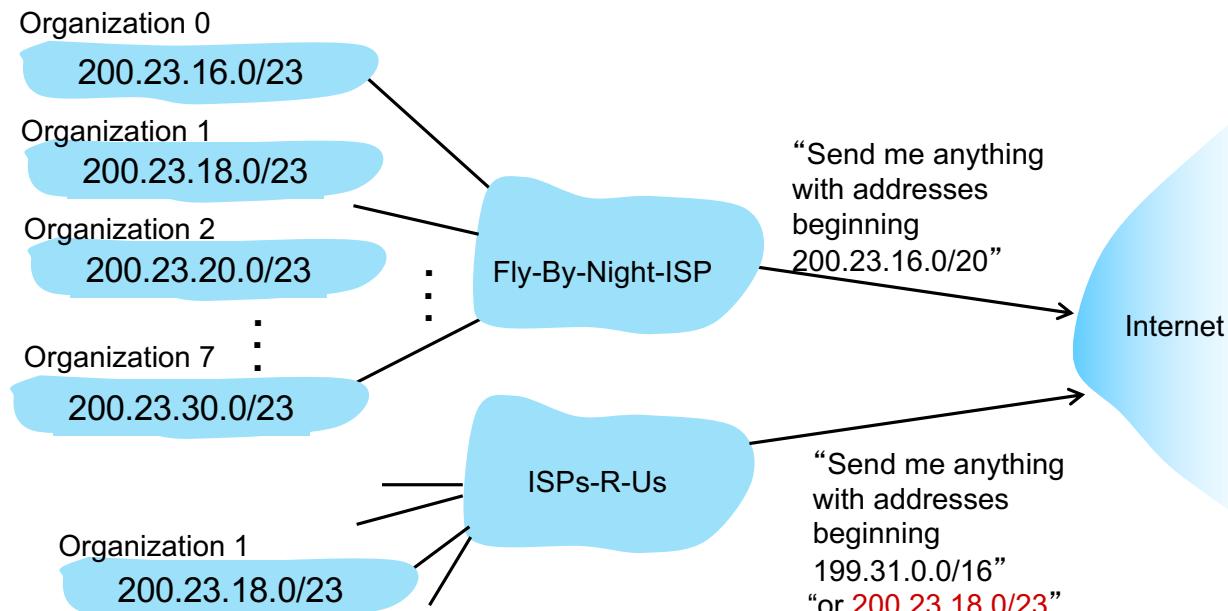
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



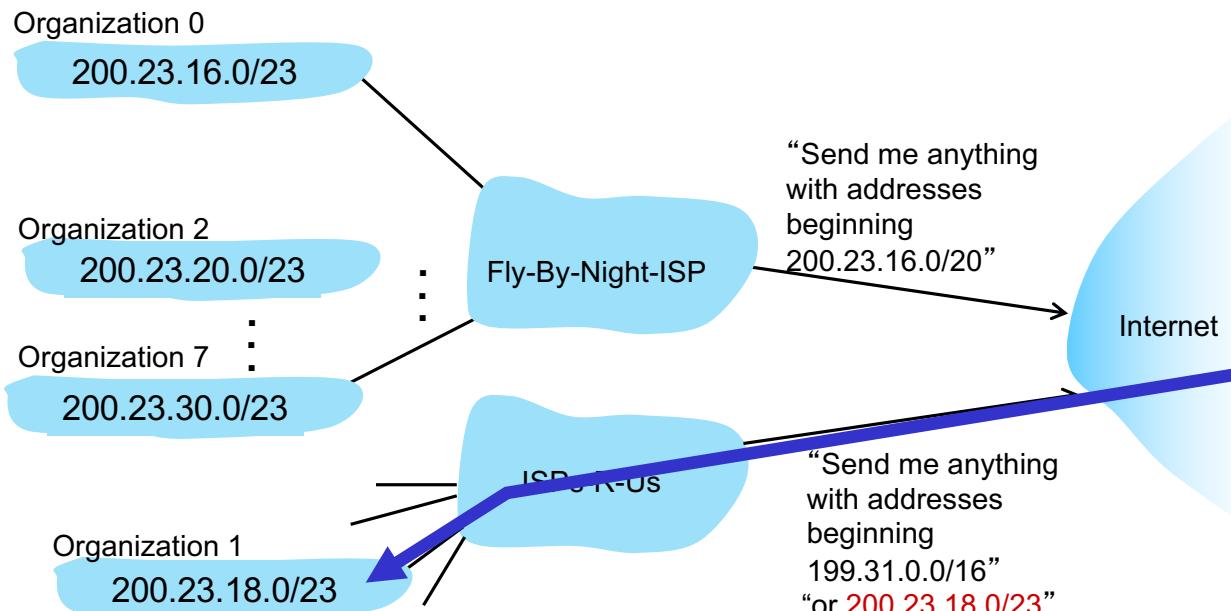
Hierarchical addressing: more specific routes

- 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 Internet Registries (RIRs)** (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 RIRs 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)

IP forwarding

Routers and hosts maintain an **IP forwarding table**

The table entries include:

- 1st column: destination IP network (although forwarding per IP host address is also possible)
- 2nd column: IP address of the host interface of next hop
- 3rd column: netmask
- last column: interface id of the local link layer interface
- other columns (depend on OS / implementation): flags, traffic volume, metric, etc.

Packet forwarding to next hop is decided based on IP destination address of the packet after applying the corresponding netmask.

IP forwarding

Example: forwarding table of host with IP address 192.110.1.240

> netstat -nr					
destination	next_hop	netmask	flags	use	interface
default	192.110.1.254	0.0.0.0	UG	102410	tu0
192.110.1.0	192.110.1.240	255.255.255.0	UH	234576	tu0
.....
192.168.1.0	192.110.1.253	255.255.255.0	UG	124586	tu0

Reading last line:

A datagram with IP network destination 192.168.1.0 will be delivered to IP address 192.110.1.253 using local interface tu0

Try to infer the network topology from the IP forwarding table!

IP forwarding: algorithm

- Consider the IP address $a.b.c.d/n$ as $X.Y$ ($<\text{network id}>.<\text{host id}>$)

Delivery:

1. netmask n is used to extract the network (or subnet) address X
2. lookup in forwarding table for the entry that **best** matches X
 - If X is local, deliver to $X.Y$ (*direct delivery*)
 - otherwise use X to determine the next hop (*adjacent delivery*)
3. If X not in table, the default route entry (default or 0.0.0.0) is used, as it matches all possible destinations.

The **default** route entry has **low priority** than other table entries.
Allows to reduce the size of forwarding table **at cost of less control**.

IP forwarding: Supernetting

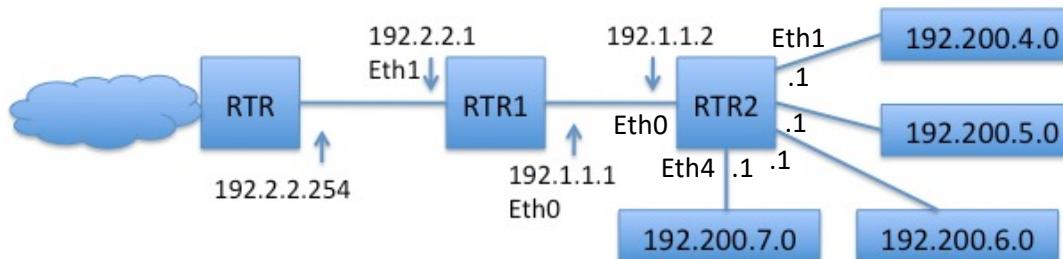


Tabela de encaminhamento de RTR1 - sem Supernetting

Destino	Próximo Nó	Máscara	Interface
192.2.2.0	192.2.2.1	255.255.255.0	Eth1
192.1.1.0	192.1.1.1	255.255.255.0	Eth0
192.200.4 (0000 0100).0	192.1.1.2	255.255.255.0	Eth0
192.200.5 (0000 0101).0	192.1.1.2	255.255.255.0	Eth0
192.200.6 (0000 0110).0	192.1.1.2	255.255.255.0	Eth0
192.200.7 (0000 0111).0	192.1.1.2	255.255.255.0	Eth0
Default	192.2.2.254	0.0.0.0	Eth1

Aggregation
of network
prefixes

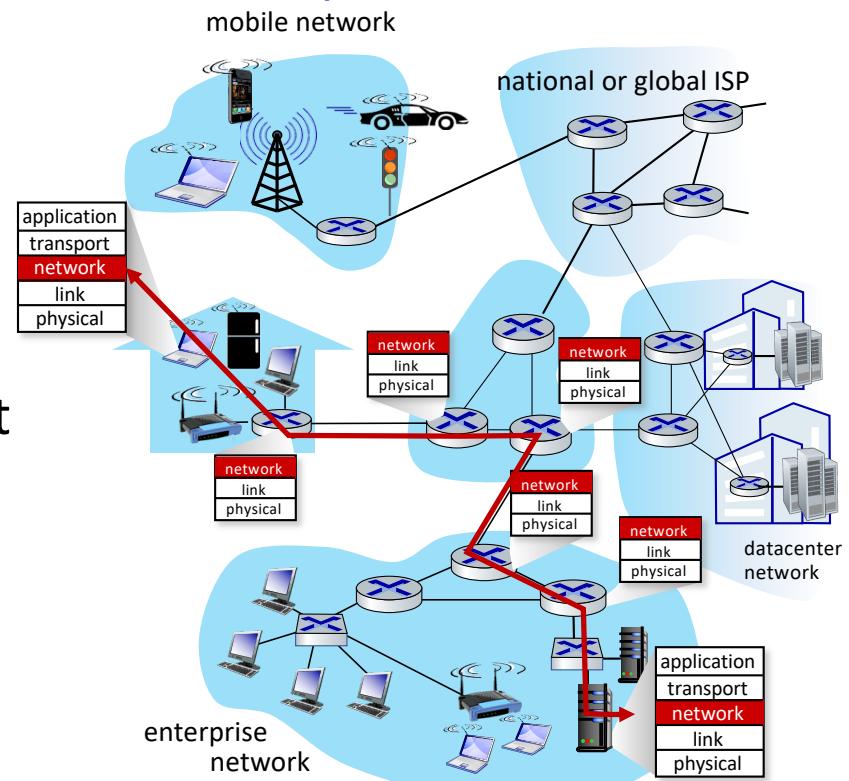
Supernetting

Using Supernetting, the size of RTR1 table is reduced from 7 to 4 entries, allowing faster lookups

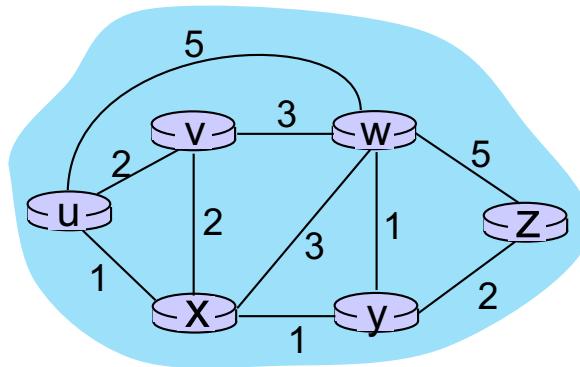
Routing protocols (Control Plane)

Routing protocol goal: determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”
- **routing:** a “top-10” networking challenge!



Graph abstraction: link costs



graph: $G = (N, E)$

N : set of routers = { u, v, w, x, y, z }

E : set of links = { $(u, v), (u, x), (v, x), (v, w), (w, x), (x, y), (w, y), (w, z), (y, z)$ }

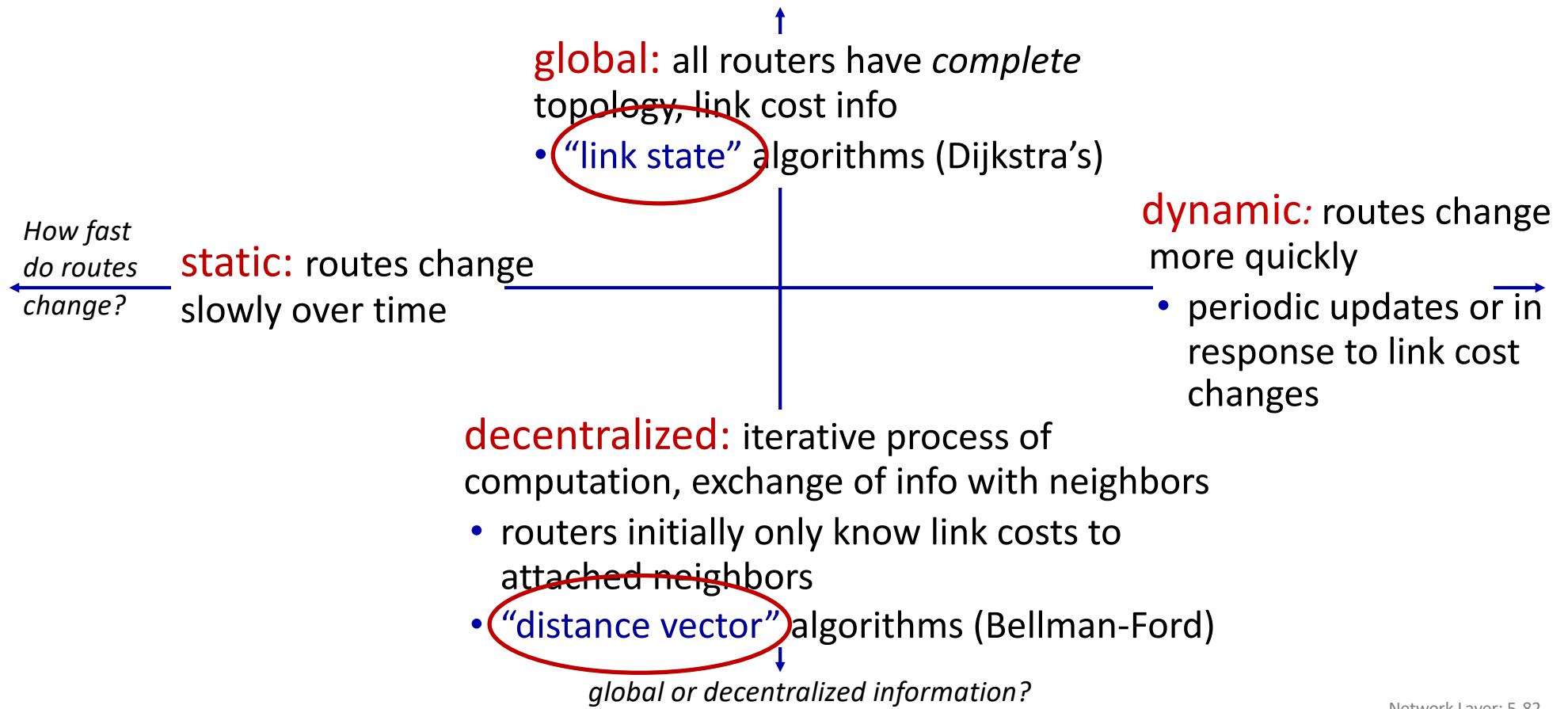
$c_{a,b}$: cost of *direct* link connecting a and b

e.g., $c_{w,z} = 5, c_{u,z} = \infty$

cost defined by network operator:
could always be 1, or inversely related
to bandwidth, or inversely related to
congestion

The network is modelled as a graph, then algorithms such as Dijkstra or Bellman-Ford evaluate the shortest path.

Routing algorithm classification (Control Plane)



Static vs Dynamic Routing (Control Plane)

- Static Routing
 - based on routes defined manually or pre-defined in a configuration file
 - reduces network traffic as no route advertisements take place
 - simple scheme but unable to accommodate network topology changes
- Dynamic Routing
 - router sends route advertisements to adjacent neighbors
 - network traffic increases due to periodic announcements or link changes
 - flexible scheme, able to adapt to network topology changes or router failure
 - common routing protocols:
 - Distance Vector: RIP (Routing Information Protocol) (within an Autonomous System (AS)), BGP (Border Gateway Protocol) (among ASs)
 - Link State: OSPF (Open Shortest Path First) (within an AS)

Routing Concepts (Control Plane)

- An **Autonomous System** (AS) is a set of IP routing prefixes under control of a single administrative entity or domain, that presents a clearly defined routing policy to the Internet.
 - each ISPs has a unique AS number on the Internet
 - NOS (AS2860), MEO Res. (AS3243), Vodafone (AS12353), RCCN (AS1930)
- Routers may know multiple routes (static and dynamic) for the same IP destination. **How to choose the best one?**
- **Distance** – administrative indicator which establishes an order of preference to learn routes from *different* routing protocols
- **Metric** – measure indicating the cost of forwarding traffic on a link, establishing a preference among routes offered by the *same* protocol

Network layer: “data plane” roadmap

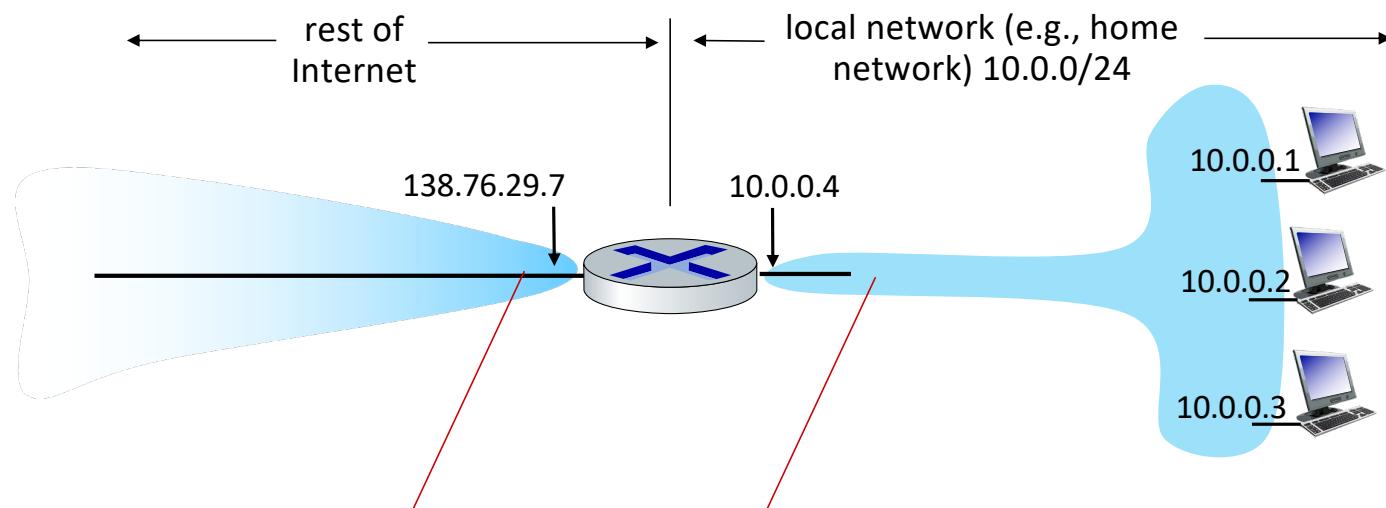
- Network layer: overview
 - data plane
 - control plane
- What's inside a router
 - input ports, switching, output ports
 - buffer management, scheduling
- IP: the Internet Protocol
 - datagram format
 - addressing
 - network address translation
 - IPv6



- 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 (aka PAT)

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**
- to Internet: IP or IP and port translation (static or dynamic) is required
- 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 net
 - security: devices inside local net not directly addressable, visible by outside world

* up to 64k simultaneous private hosts (16 bits port number)

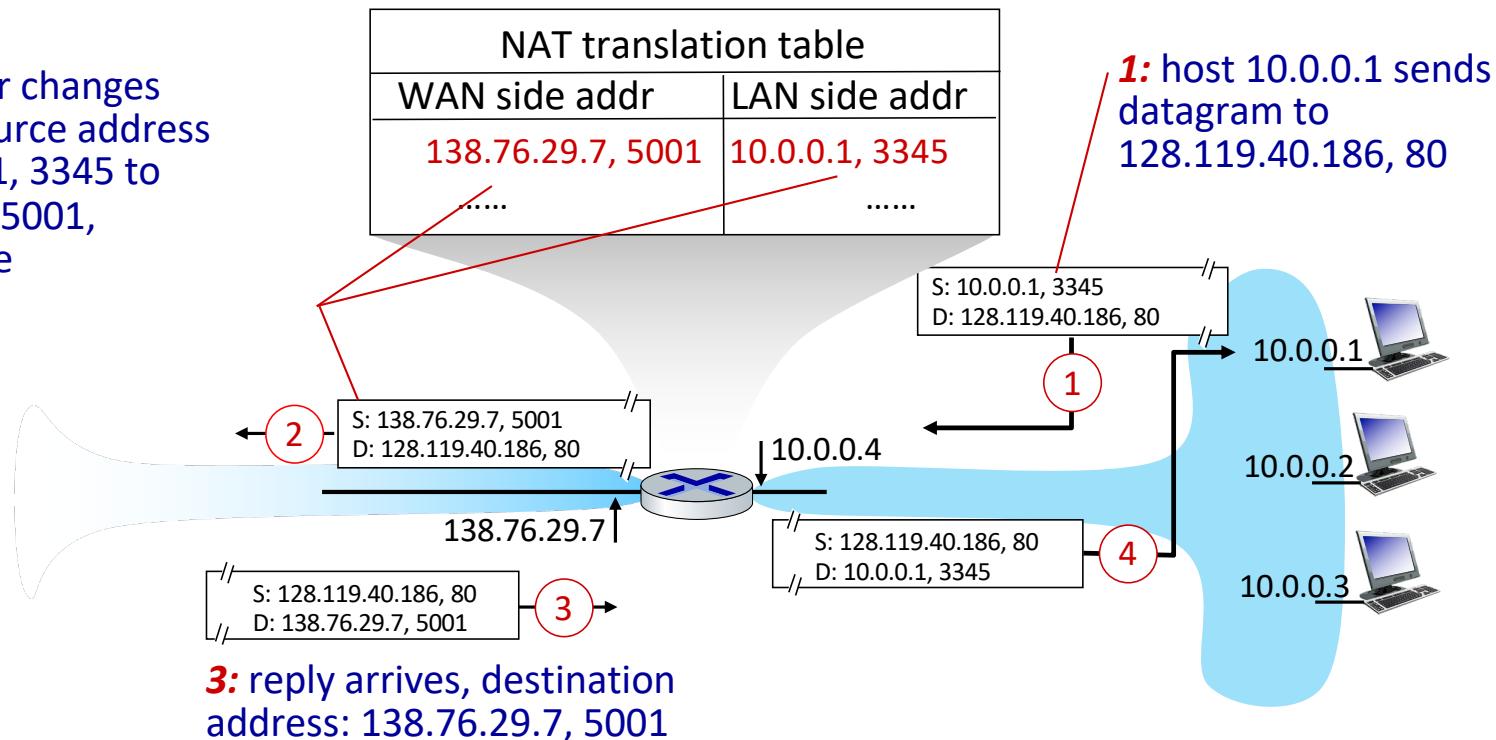
NAT: network address translation

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



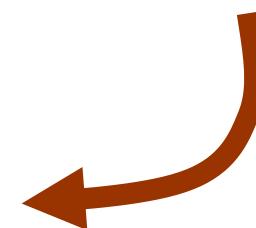
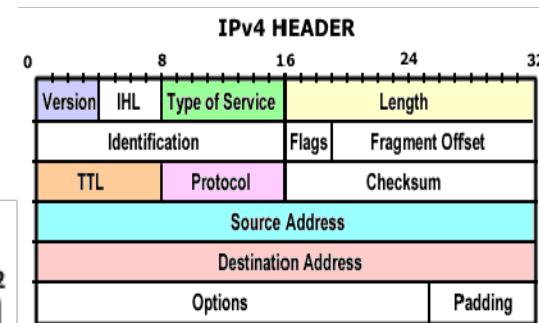
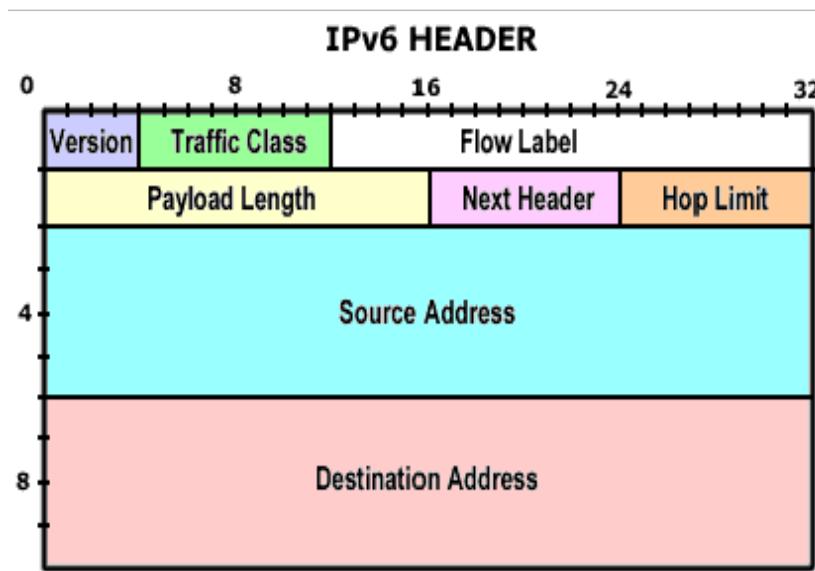
NAT: network address translation

- 16 bits port # allows mapping ~64k simultaneous private hosts
- 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?
 - solution: Static NAT/PAT combined with Dynamic NAT/PAT
- 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: motivation

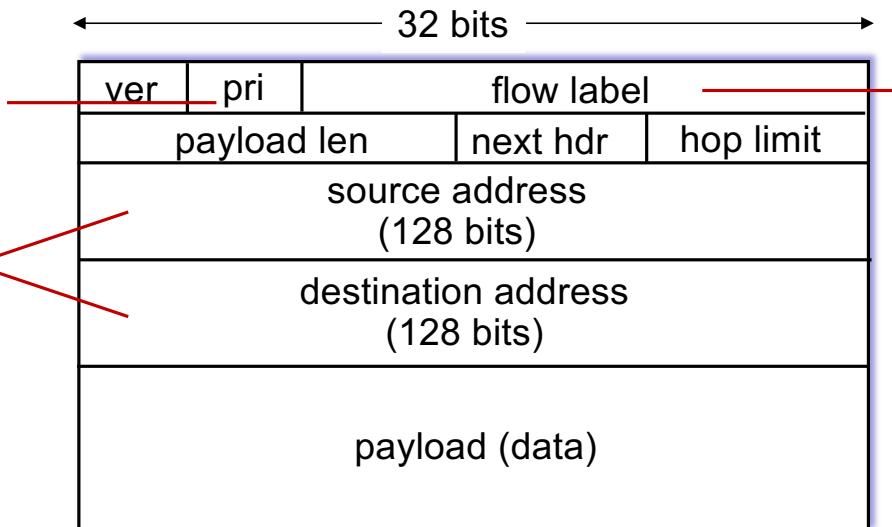


Moving to a simpler header...

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).

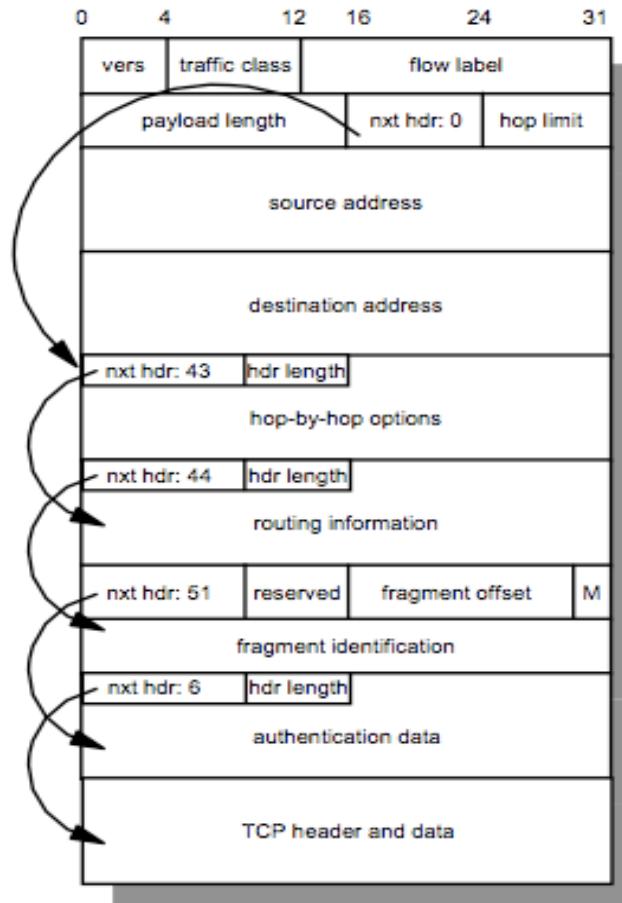
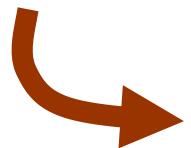
What's missing (compared with IPv4):

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

IPv6 datagram format

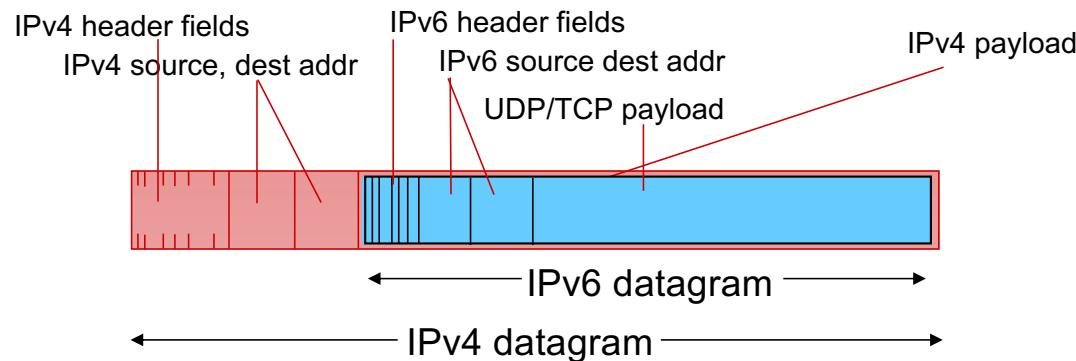
The field ***next header*** (equivalent to “Protocol” in IPv4) is used to implement specific options, as fragmentation

Example of an IPv6 packet including multiple headers



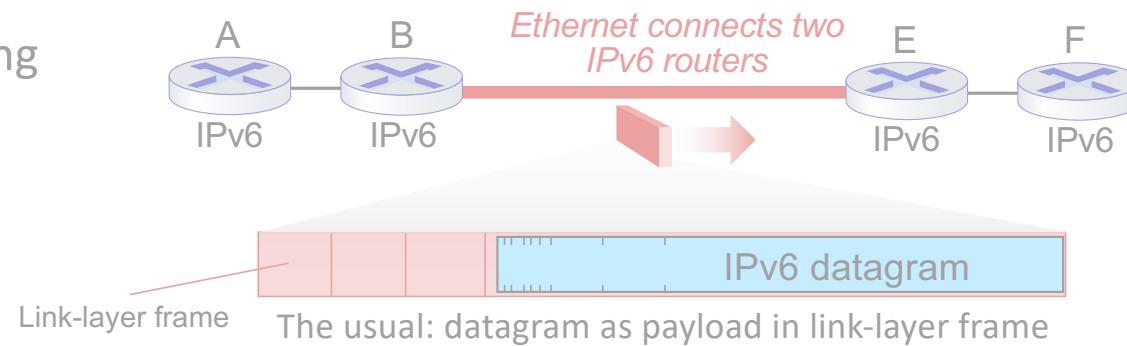
Transition from IPv4 to IPv6

- 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)

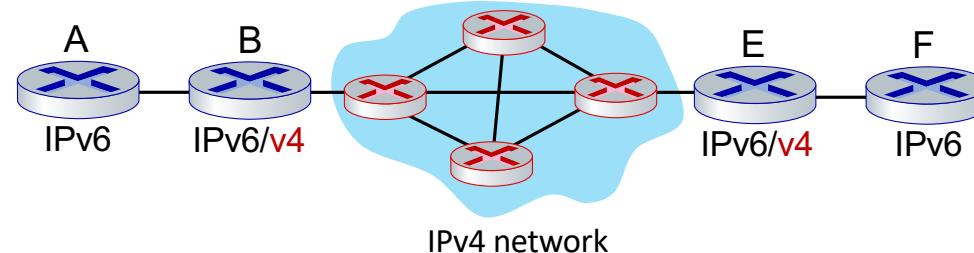


Tunneling and encapsulation

Ethernet connecting
two IPv6 routers:

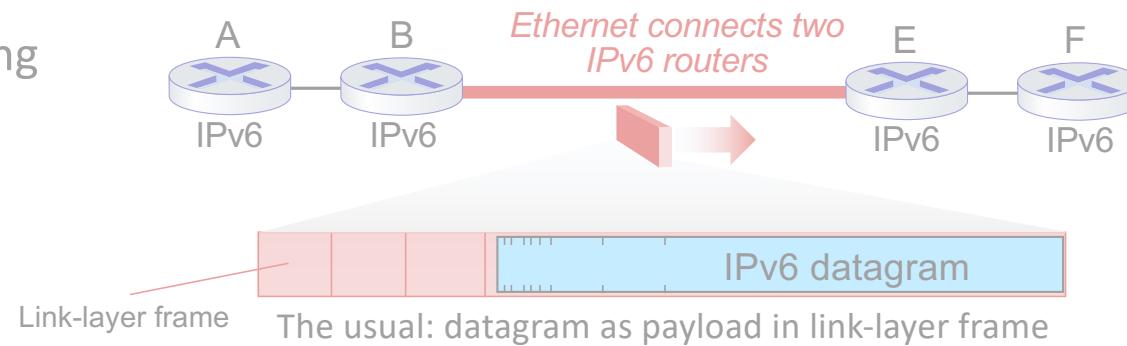


IPv4 network
connecting two
IPv6 routers

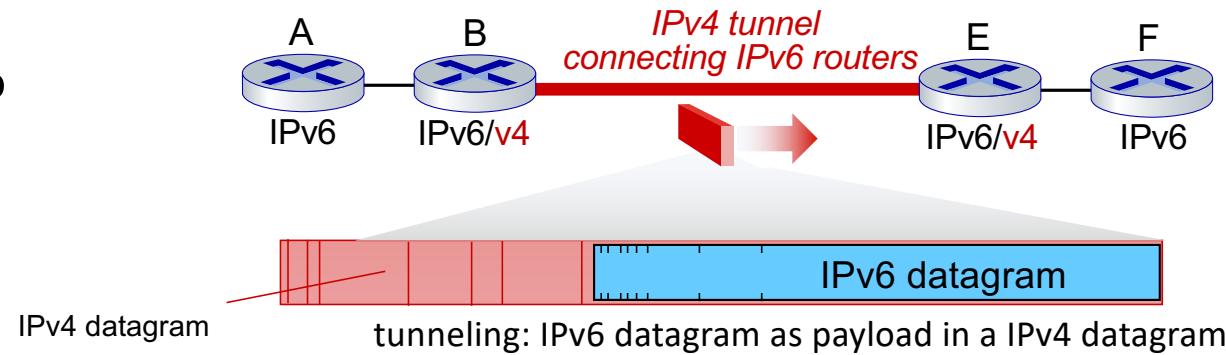


Tunneling and encapsulation

Ethernet connecting
two IPv6 routers:



IPv4 tunnel
connecting two
IPv6 routers



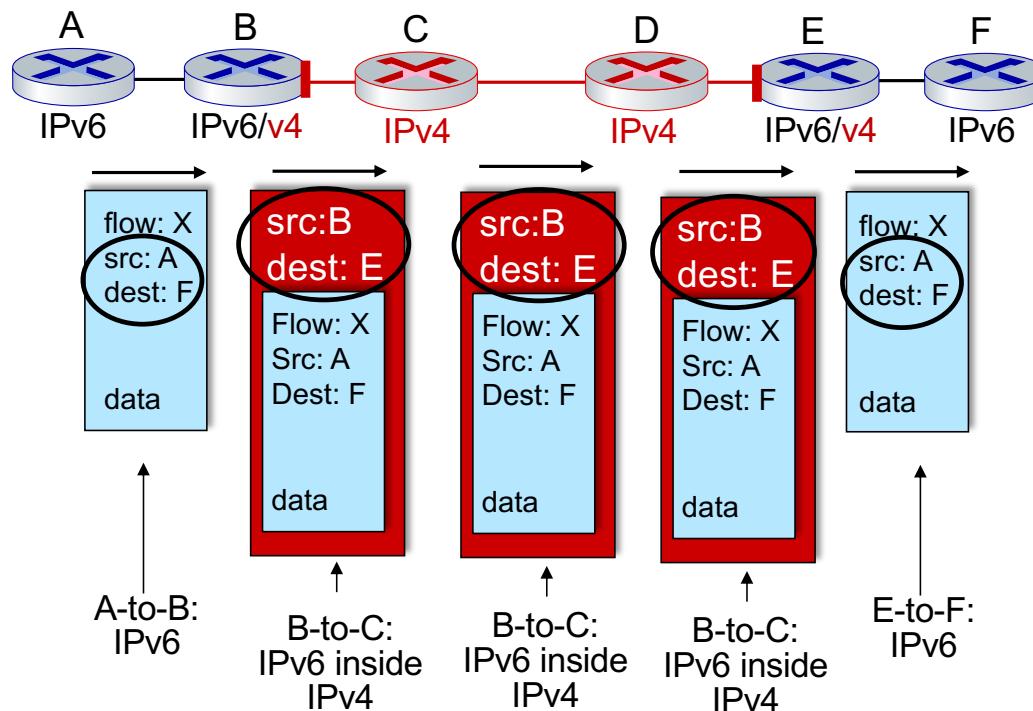
Tunneling

logical view:



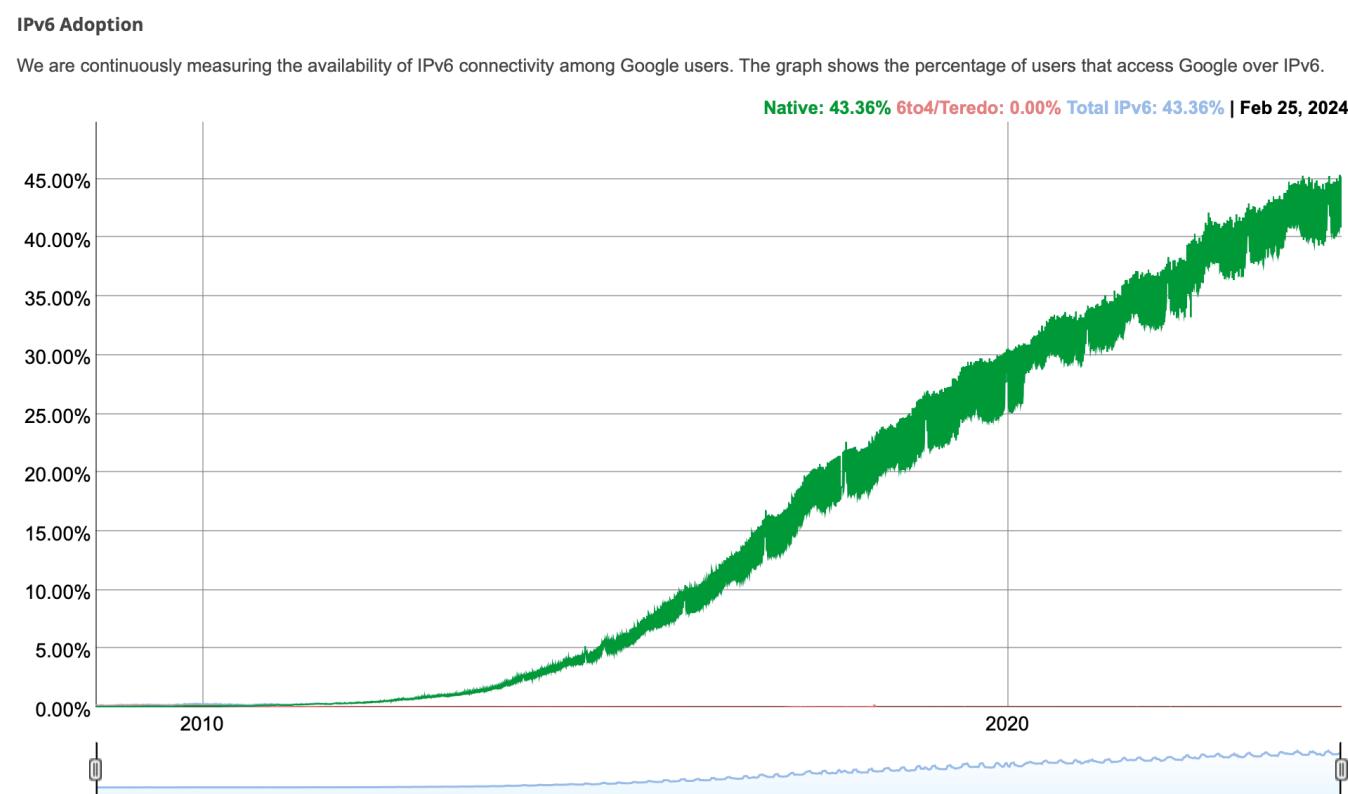
physical view:

Note source and destination addresses!



IPv6: adoption

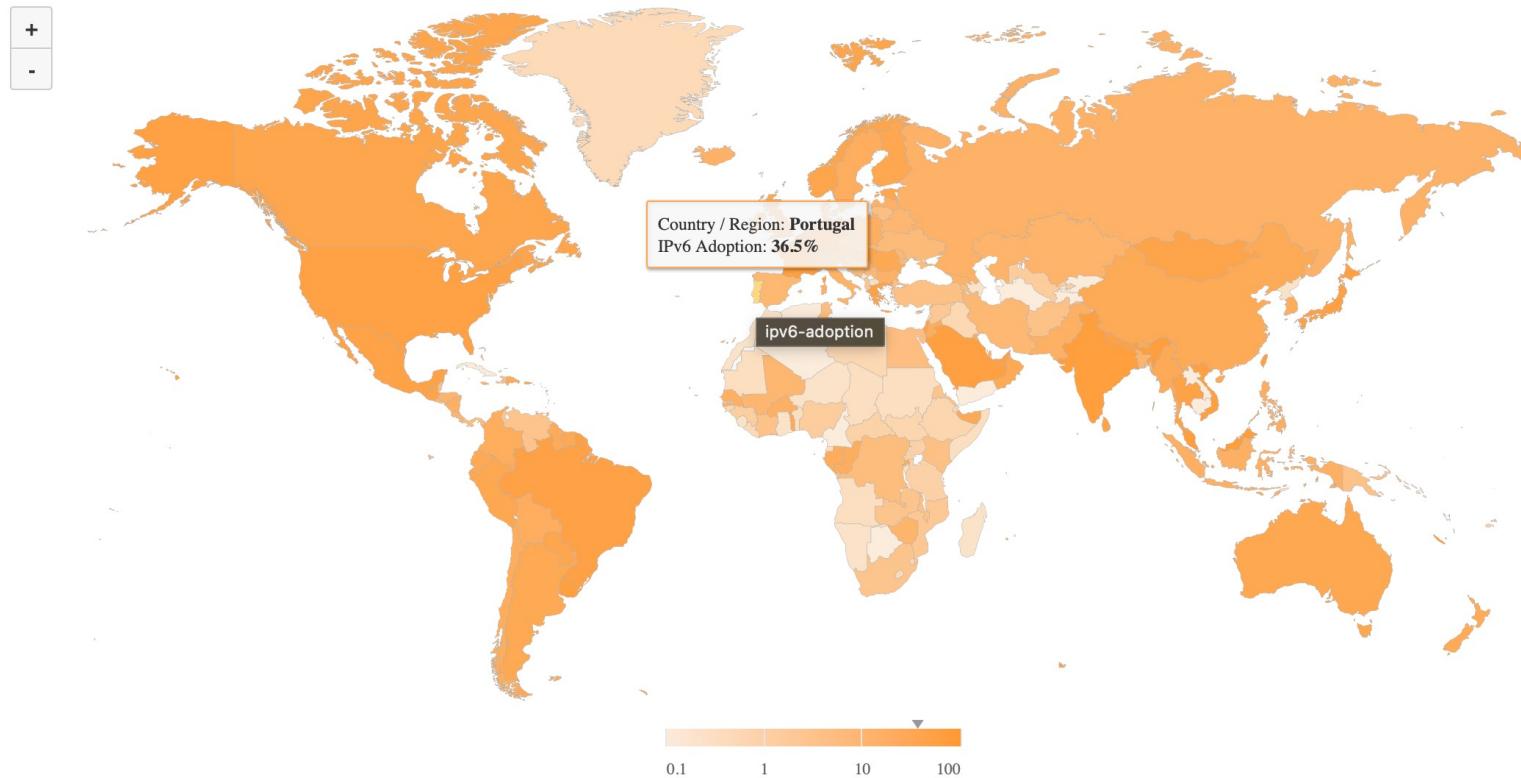
- Google¹: ~ 40% of clients access services via IPv6 (2024)



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

IPv6: adoption

IPv6 Adoption By Country / Region



(Source: AKAMAI, <https://www.akamai.com/visualizations/state-of-the-internet-report/ipv6-adoption-visualization>, March 2024)

Network Layer: 4-100

IPv6: adoption

- Google¹: ~ 40% of clients access services via IPv6 (2024)
- 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?*

¹ <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

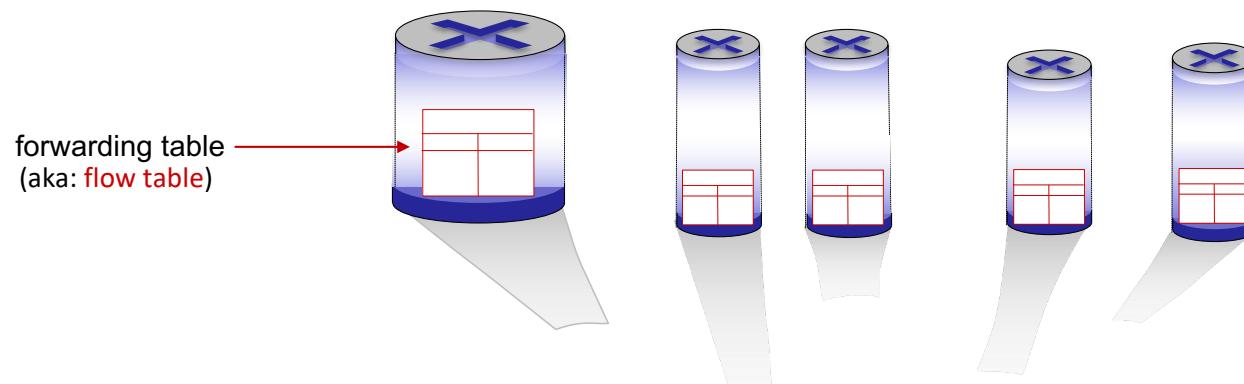


- Generalized Forwarding, SDN
 - Match+action
 - OpenFlow: match+action in action
- Middleboxes

Generalized forwarding: match plus action

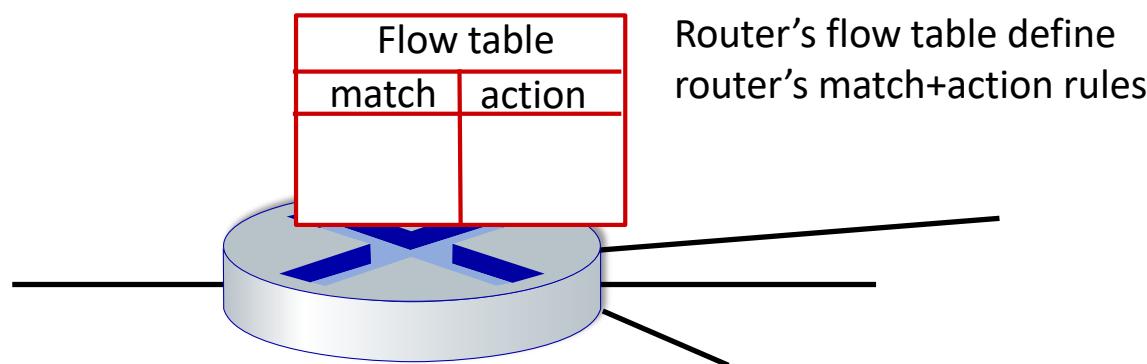
Review: each router contains a **forwarding table** (aka: **flow table**)

- “**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



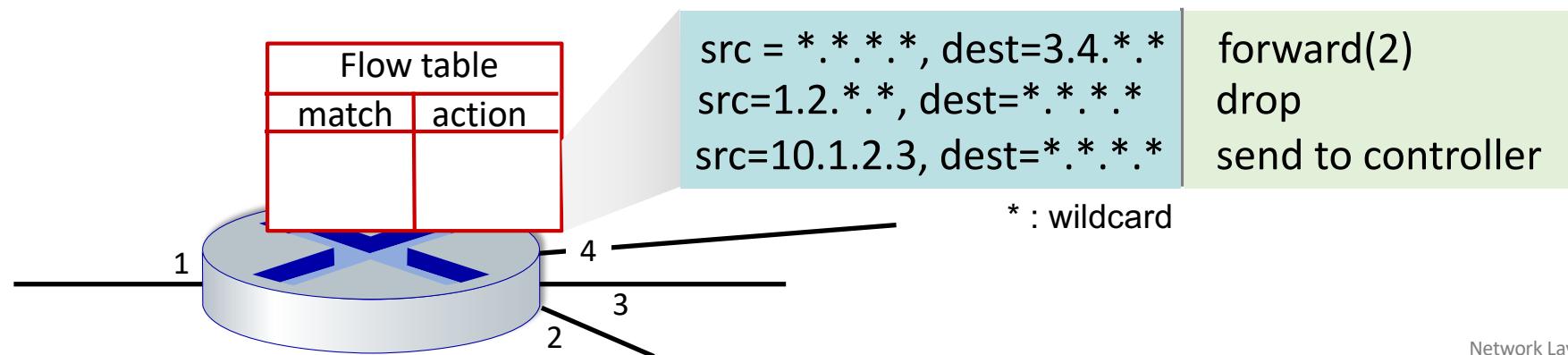
Flow table abstraction

- **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

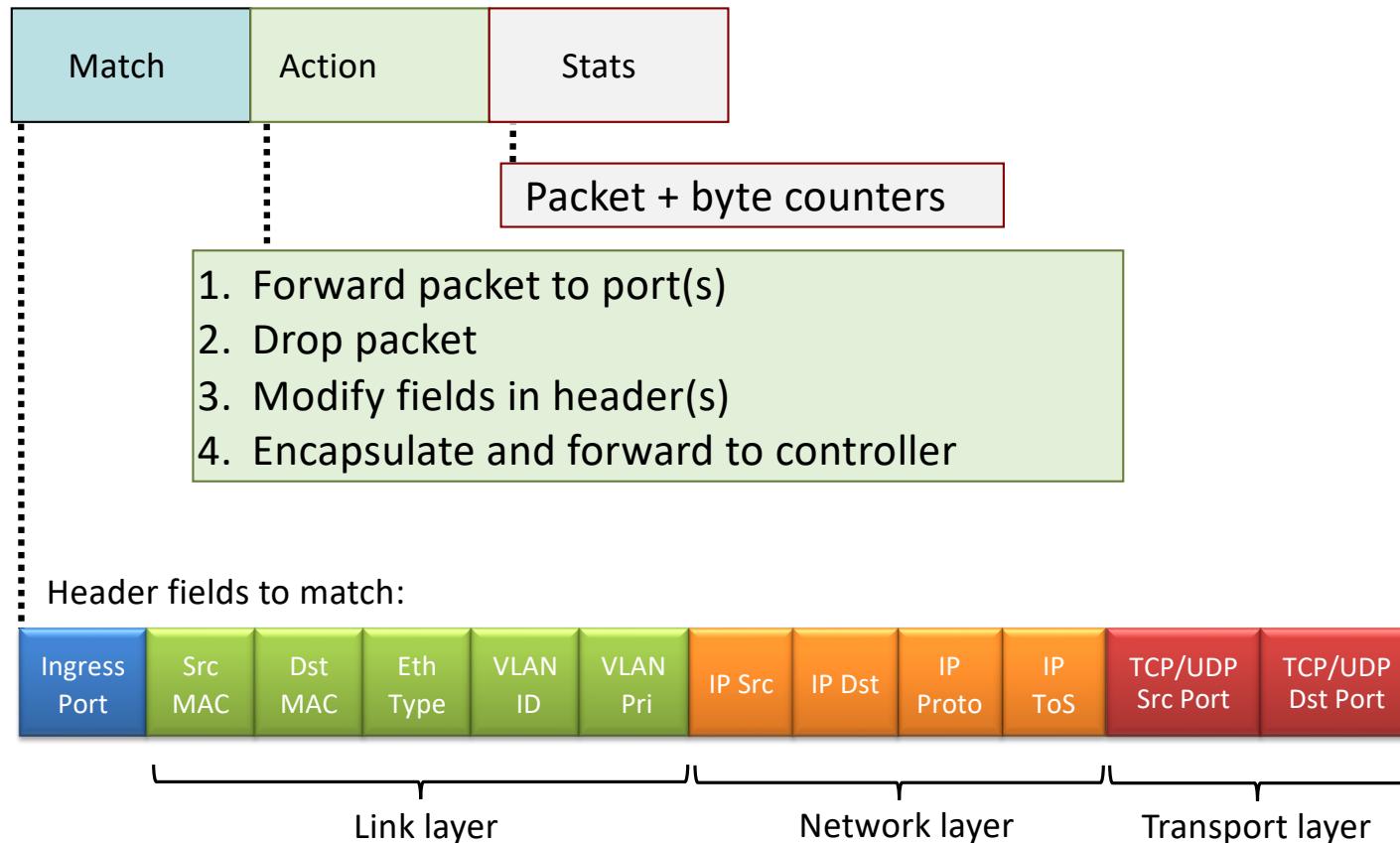


Flow table abstraction

- **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

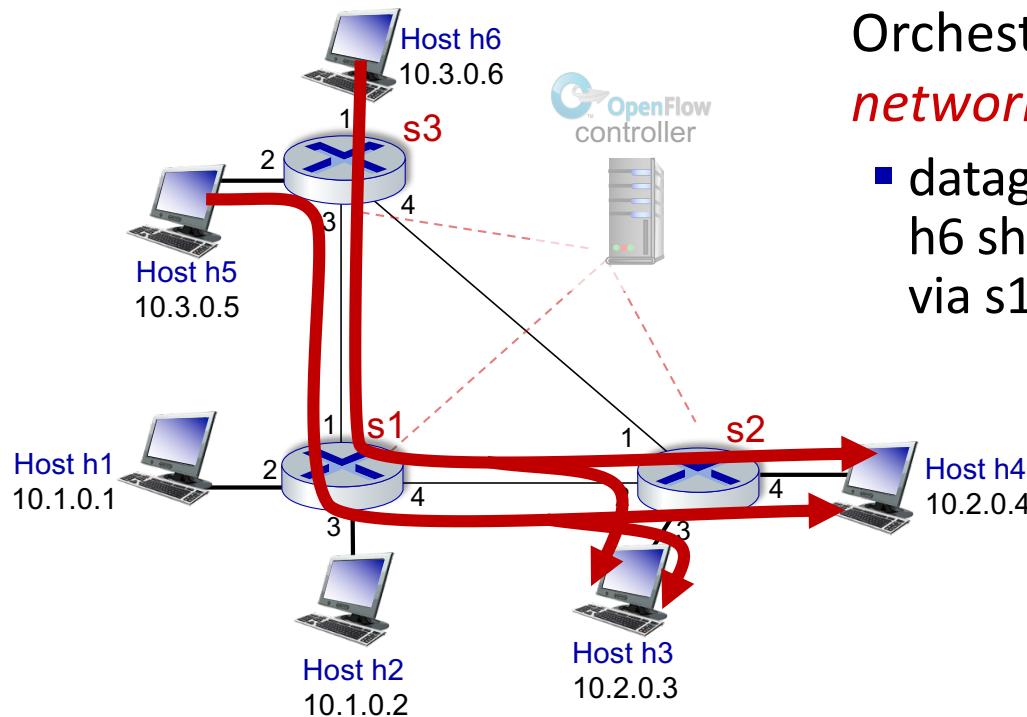
Firewall

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

NAT

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

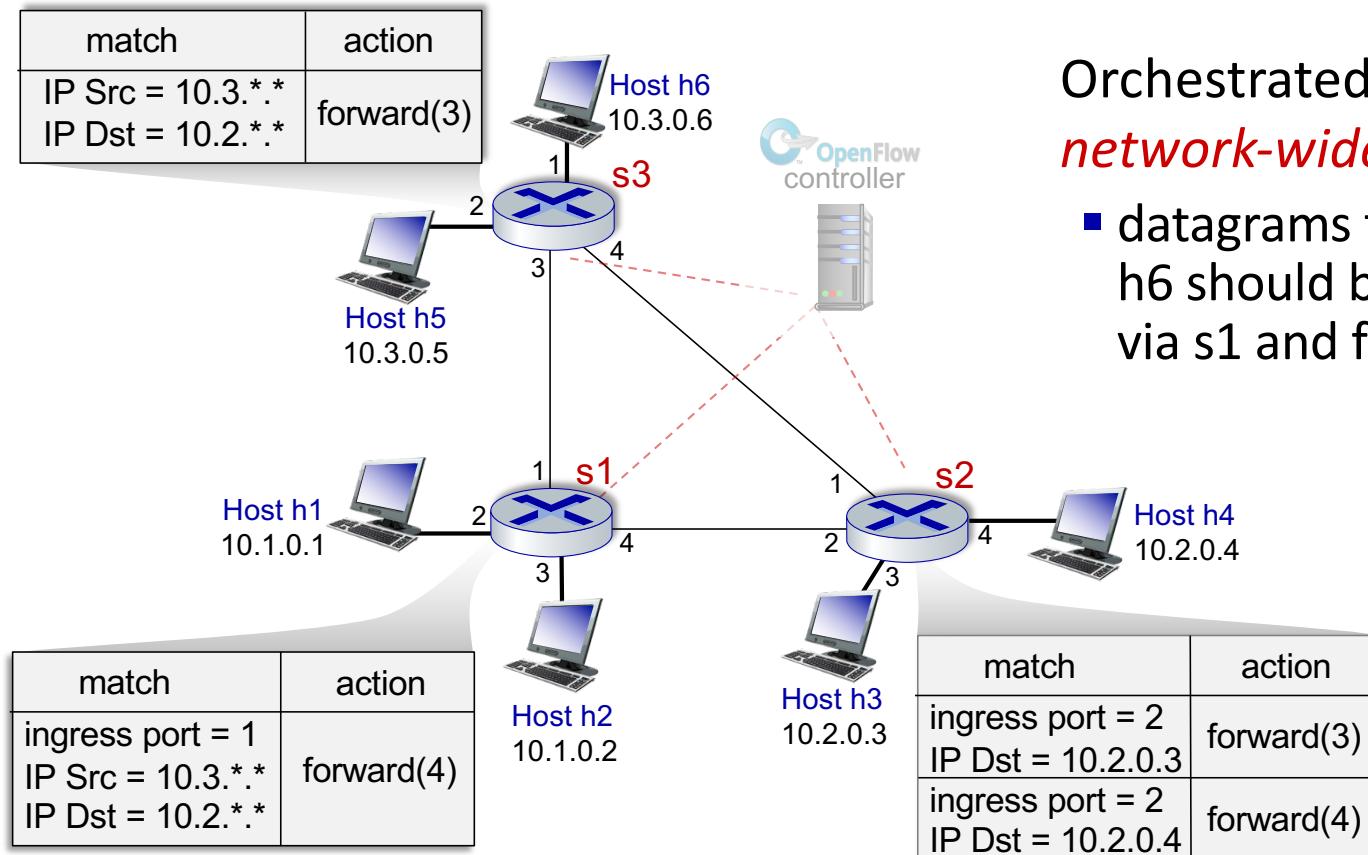
OpenFlow example



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



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

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

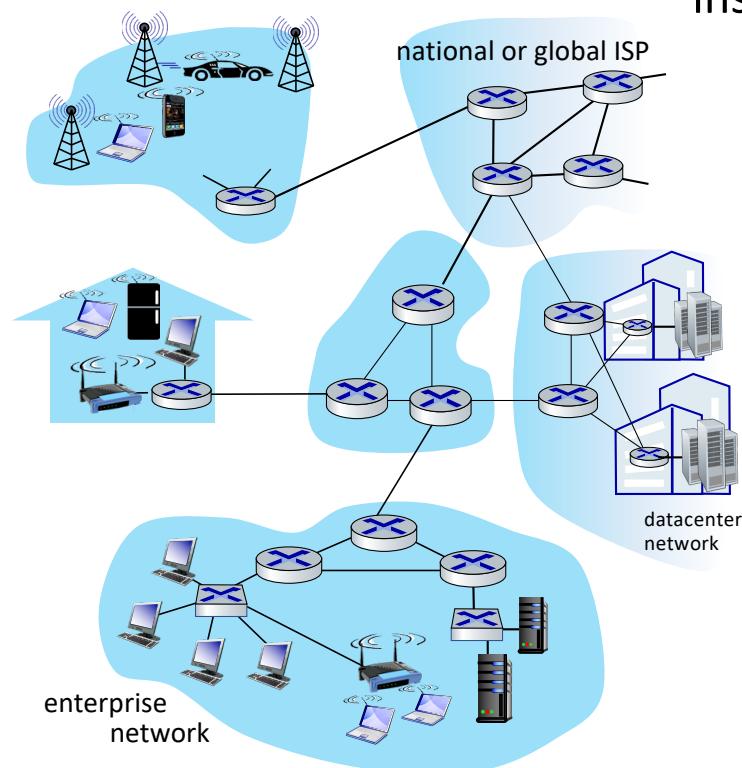
Middlebox (RFC 3234)

“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

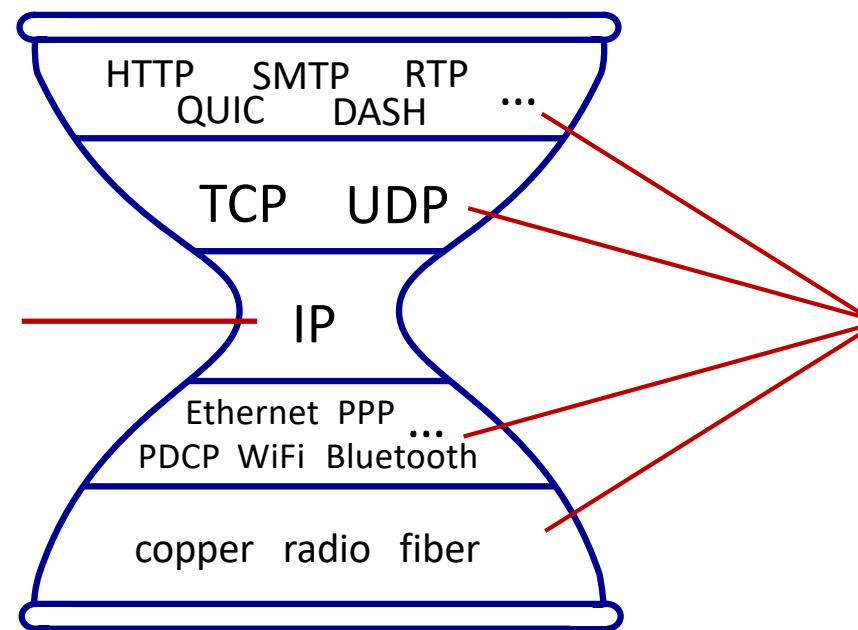
Caches: service provider, mobile, CDNs

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

The IP hourglass

- Internet's "thin waist":
- one network layer protocol: IP
 - must be implemented by every (billions) of Internet-connected devices

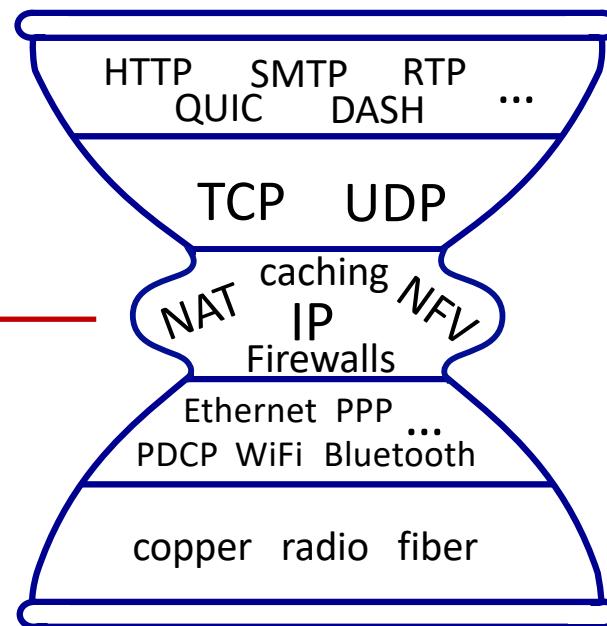


many protocols
in physical, link,
transport, and
application
layers

The IP hourglass, at middle age

Internet's middle age
“love handles”?

- middleboxes,
operating inside the
network



Architectural Principles of the Internet

RFC 1958

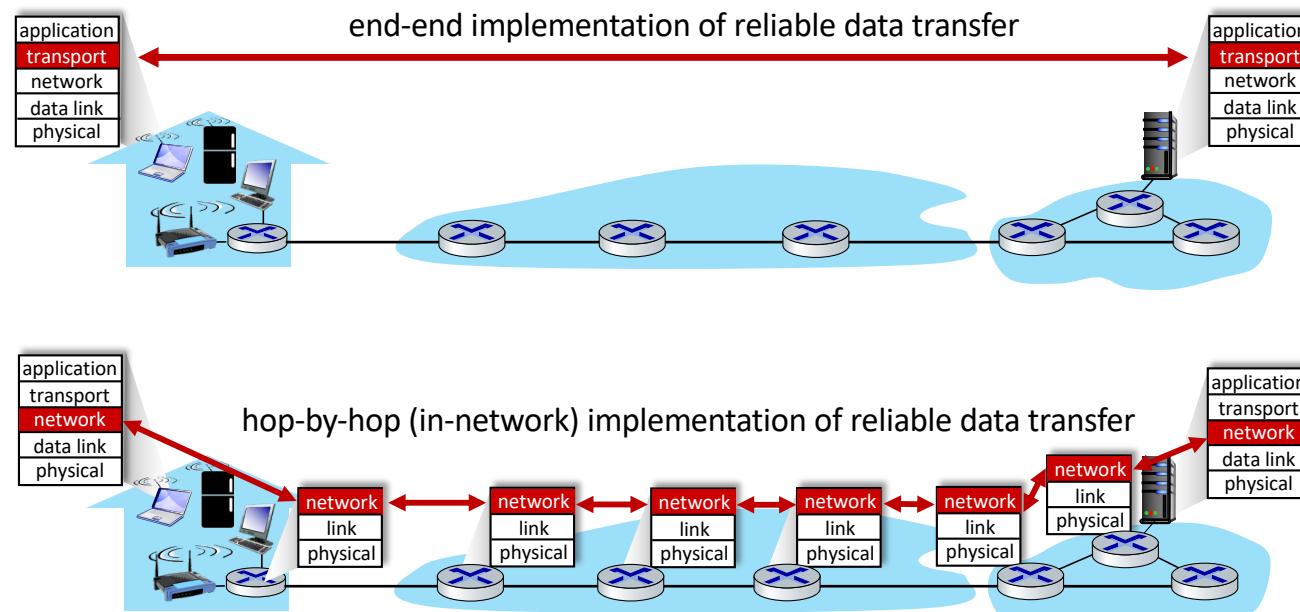
“Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that **the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network.**”

Three cornerstone beliefs:

- simple connectivity
- IP protocol: that narrow waist
- intelligence, complexity at network edge

The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented **in network**, or at **network edge**



The end-end argument

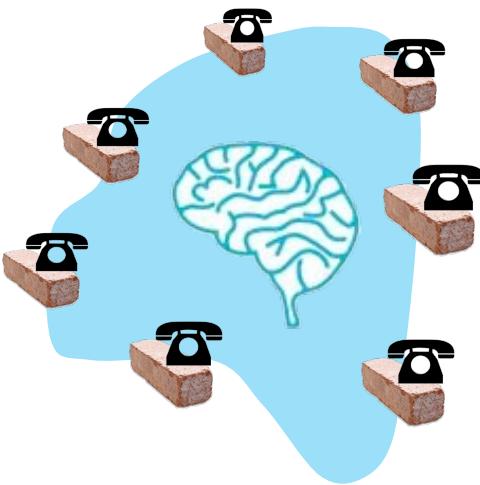
- some network functionality (e.g., reliable data transfer, congestion) can be implemented **in network**, or at **network edge**

“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

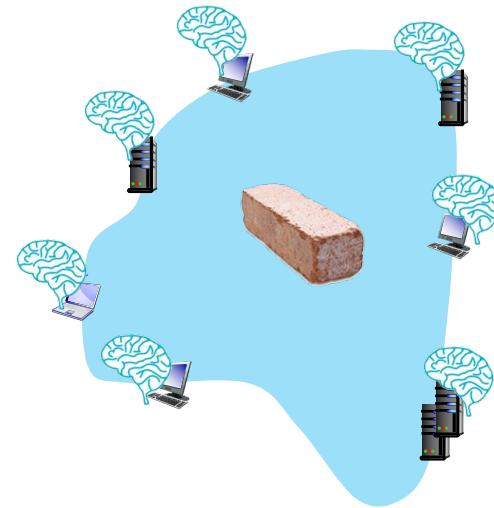
We call this line of reasoning against low-level function implementation the “end-to-end argument.”

Saltzer, Reed, Clark 1981

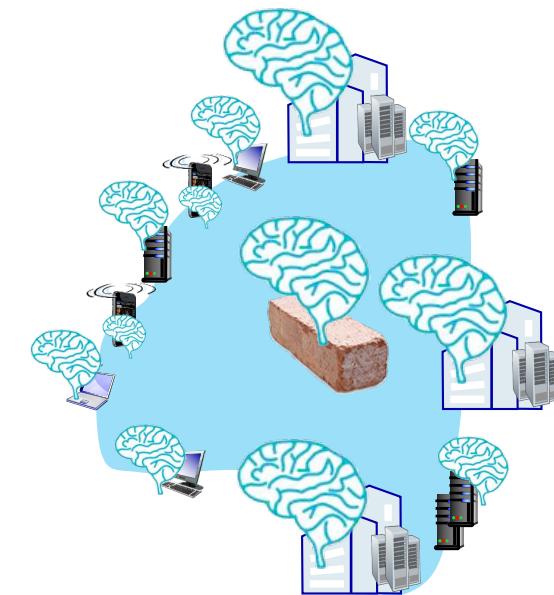
Where's the intelligence?



20th century phone net:
• intelligence/computing at
network switches



Internet (pre-2005)
• intelligence, computing at
edge



Internet (post-2005)
• programmable network devices
• intelligence, computing, massive
application-level infrastructure at edge

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)