CSC/CPE 138

COMPUTER NETWORK FUNDAMENTALS

SACRAMENTO STATE

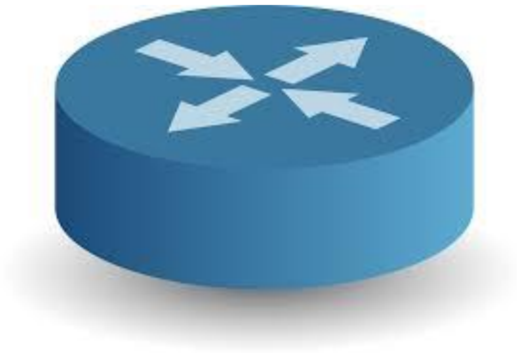
Lecture 4_2: The Network Layer (Data Plane)

California State University, Sacramento
Fall 2024

Slide Courtesy: Computer Networking: A Top-Down Approach, Kurose Ross, 8th Edition

Lecture 4_2 Overview

- Dynamic Host Configuration Protocol
- Route Aggregation
- Network Address Translation
- IPv6 Transition
- Generalized Forwarding Vs OpenFlow
- Middleboxes
- IP Hourglass



DHCP: Dynamic Host Configuration Protocol

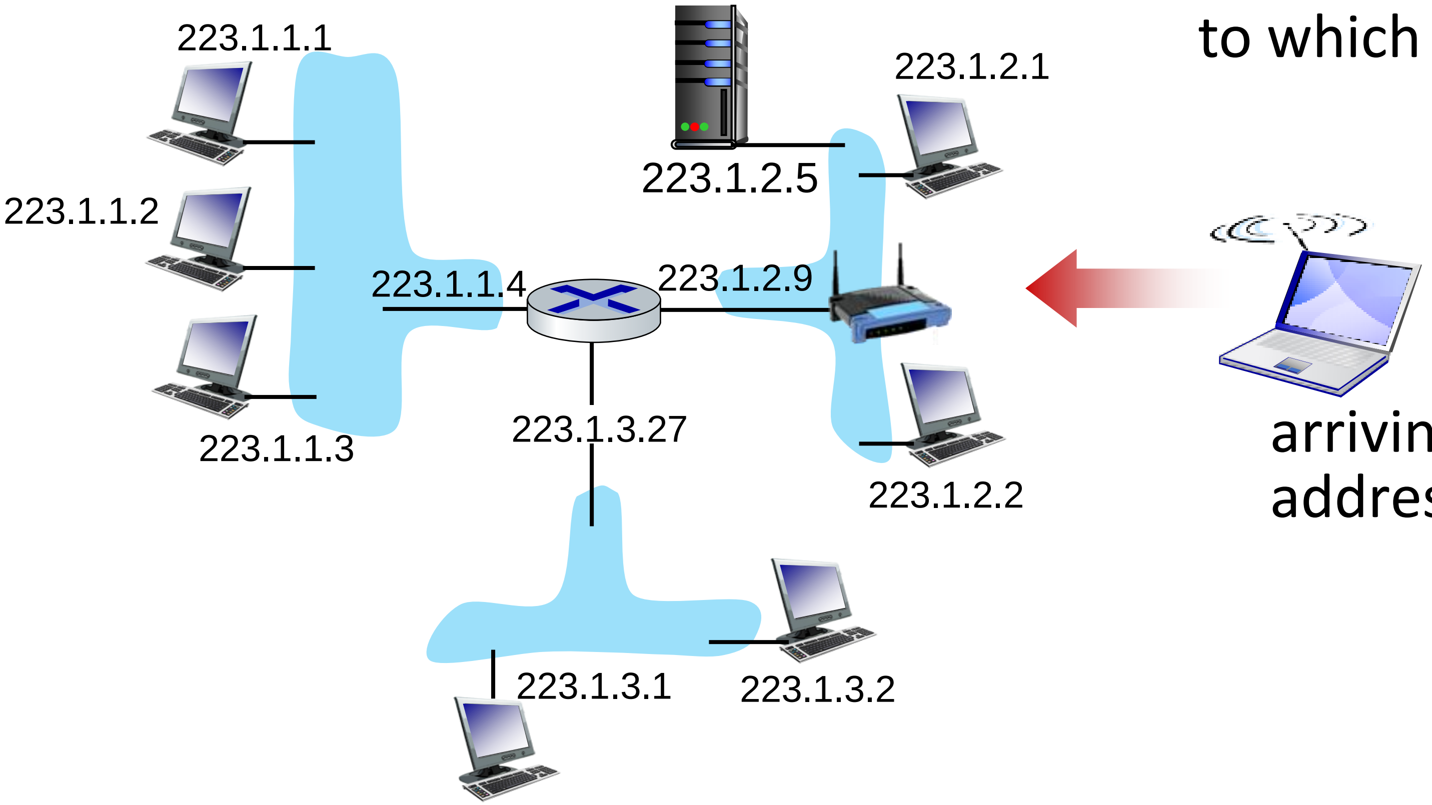
Goal: host *dynamically* obtains IP address from network server when it "joins" network

- 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
- DHCP server responds with DHCP offer msg
- host requests IP address: DHCP request msg
- DHCP server sends address: DHCP ack msg

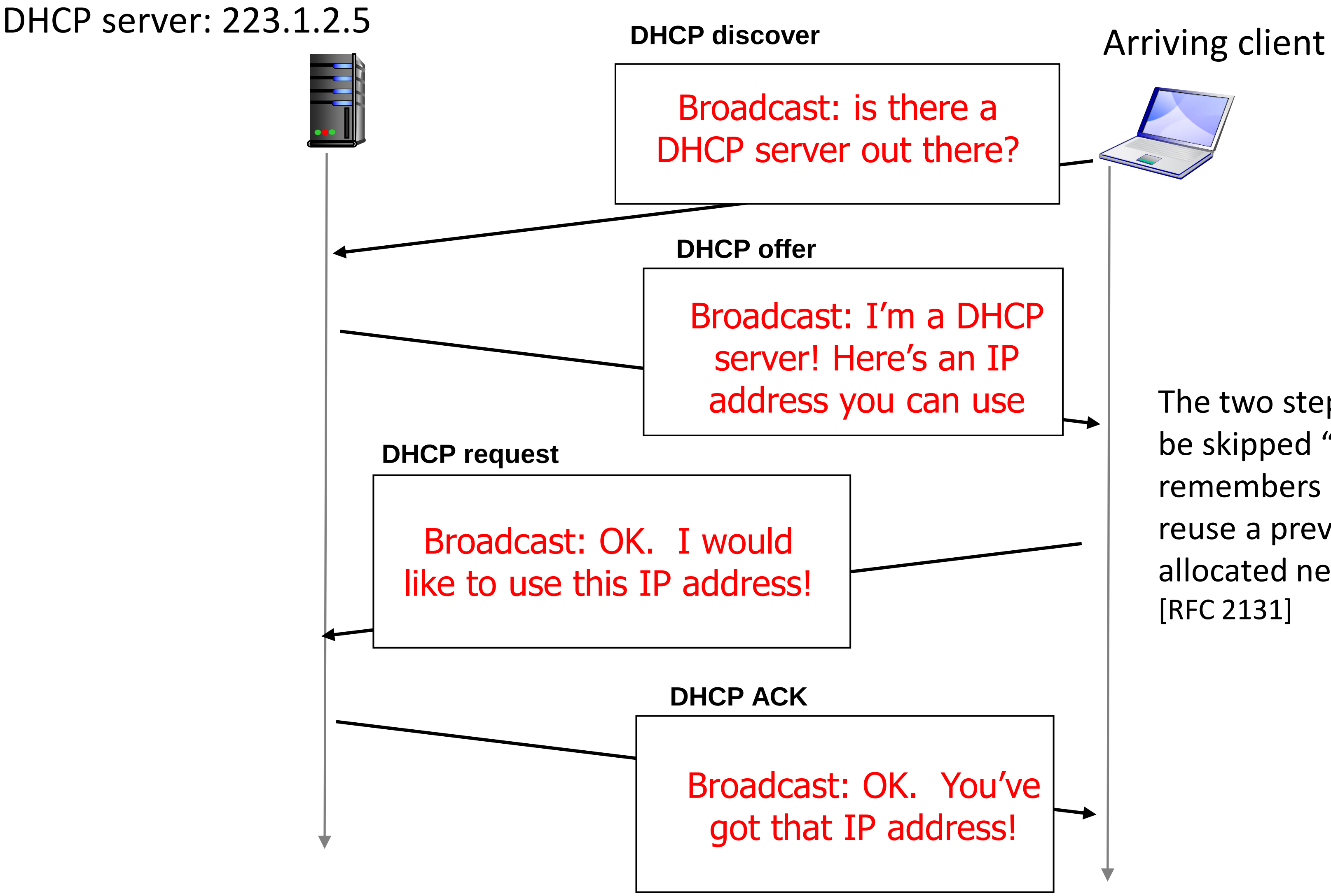
DHCP client-server scenario



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



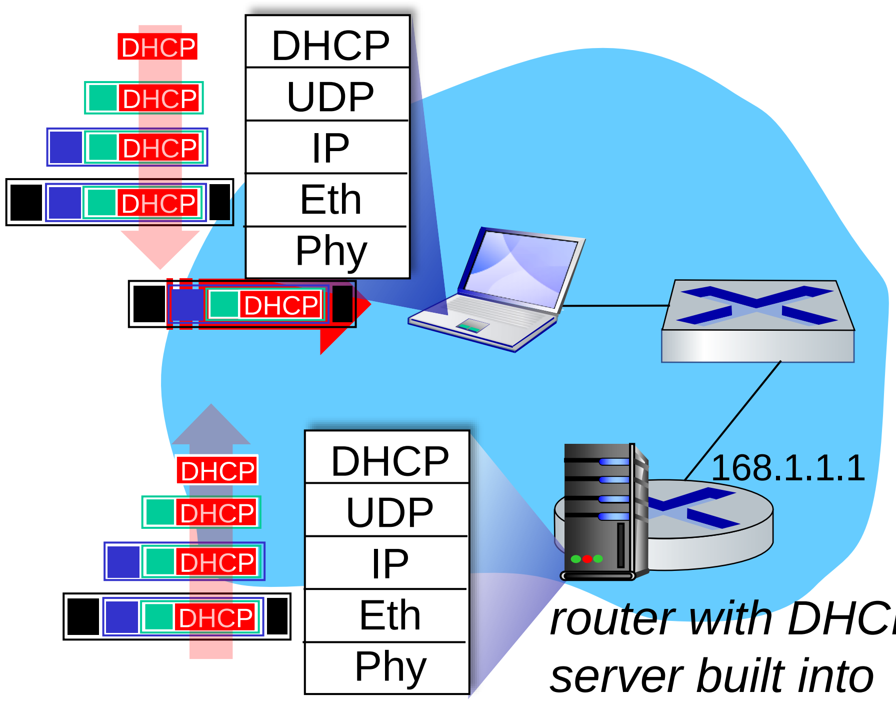
The two steps above can be skipped "if a client remembers and wishes to reuse a previously allocated network address" [RFC 2131]

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)

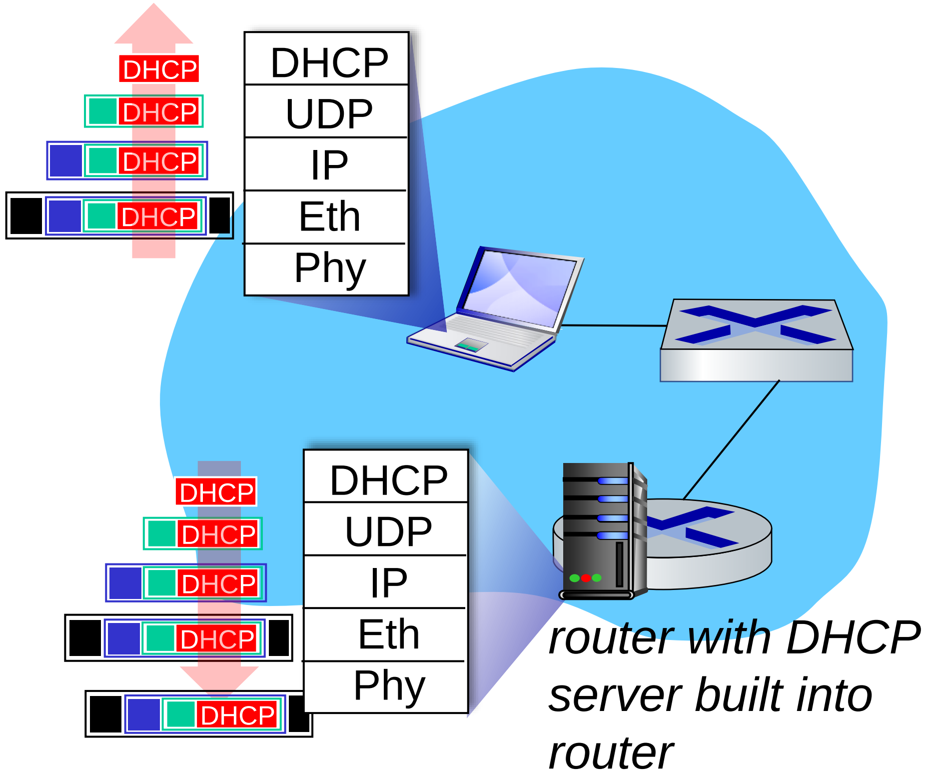
DHCP: example



router with DHCP server built into router

- 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 *network* get subnet part of IP address?

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

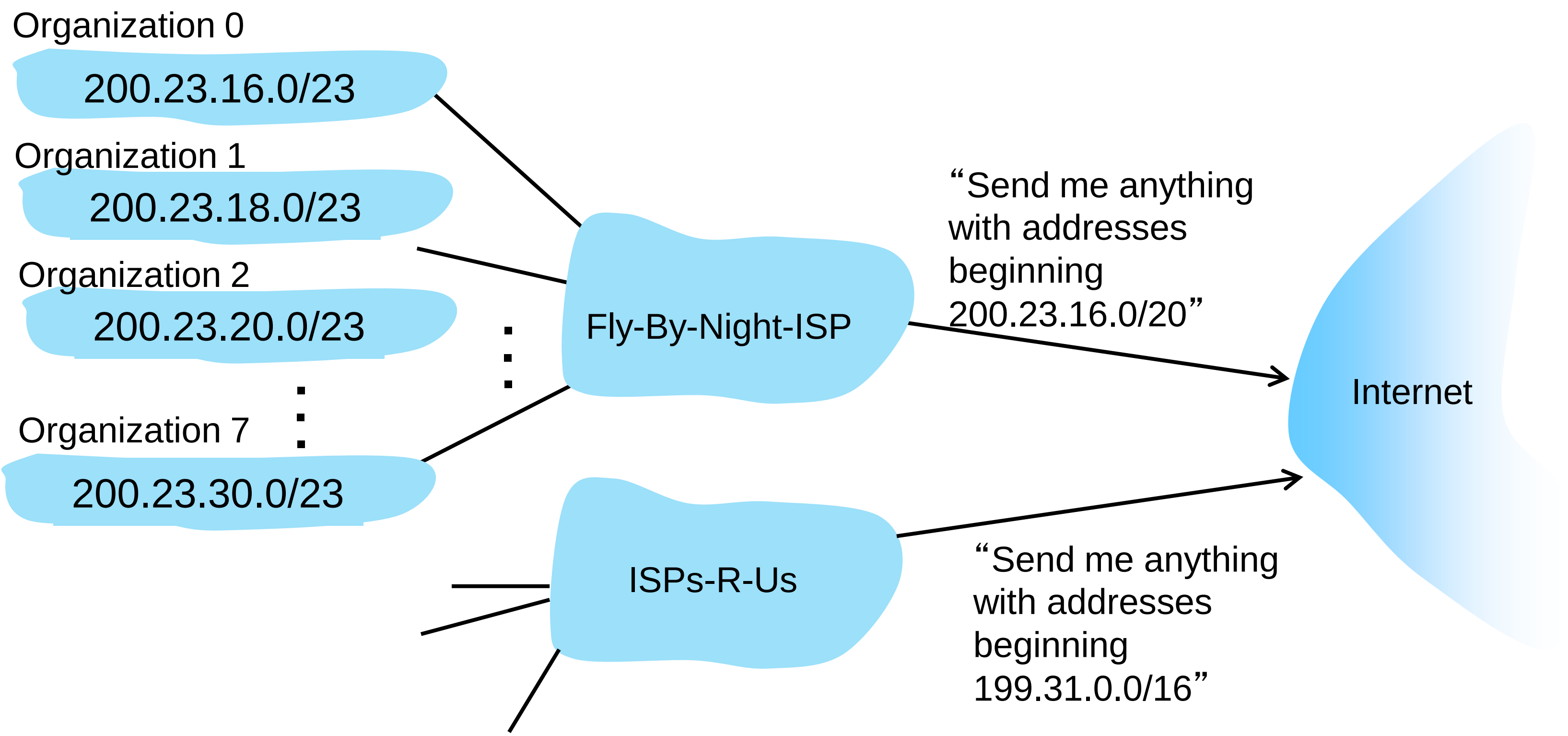
ISP's block	<u>11001000 00010111 0001</u>0000 00000000	200.23.16.0/20

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

Organization 0	<u>11001000 00010111 0001000</u>0 00000000	200.23.16.0/23
Organization 1	<u>11001000 00010111 0001001</u>0 00000000	200.23.18.0/23
Organization 2	<u>11001000 00010111 0001010</u>0 00000000	200.23.20.0/23
...	….. ….	….
Organization 7	<u>11001000 00010111 0001111</u>0 00000000	200.23.30.0/23

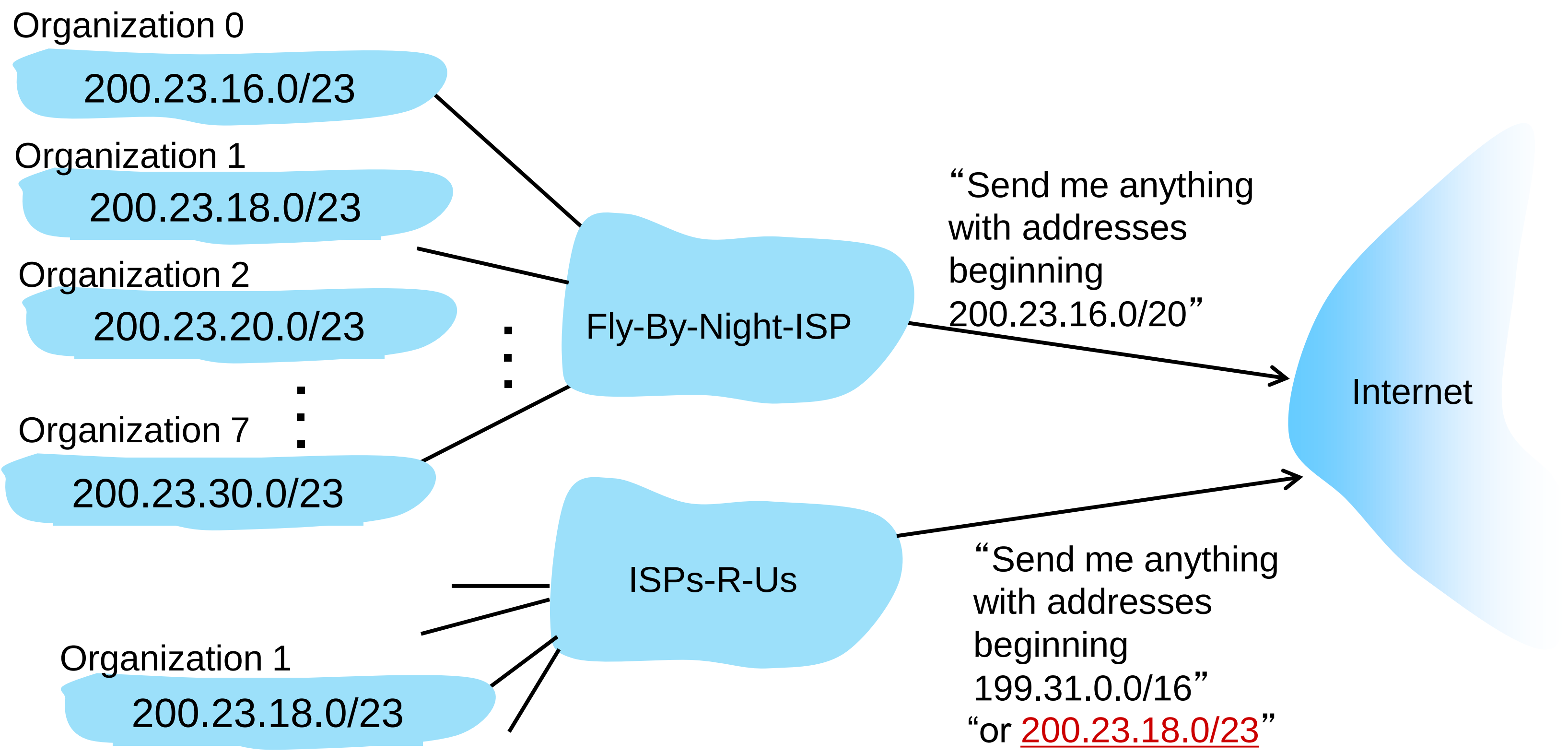
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



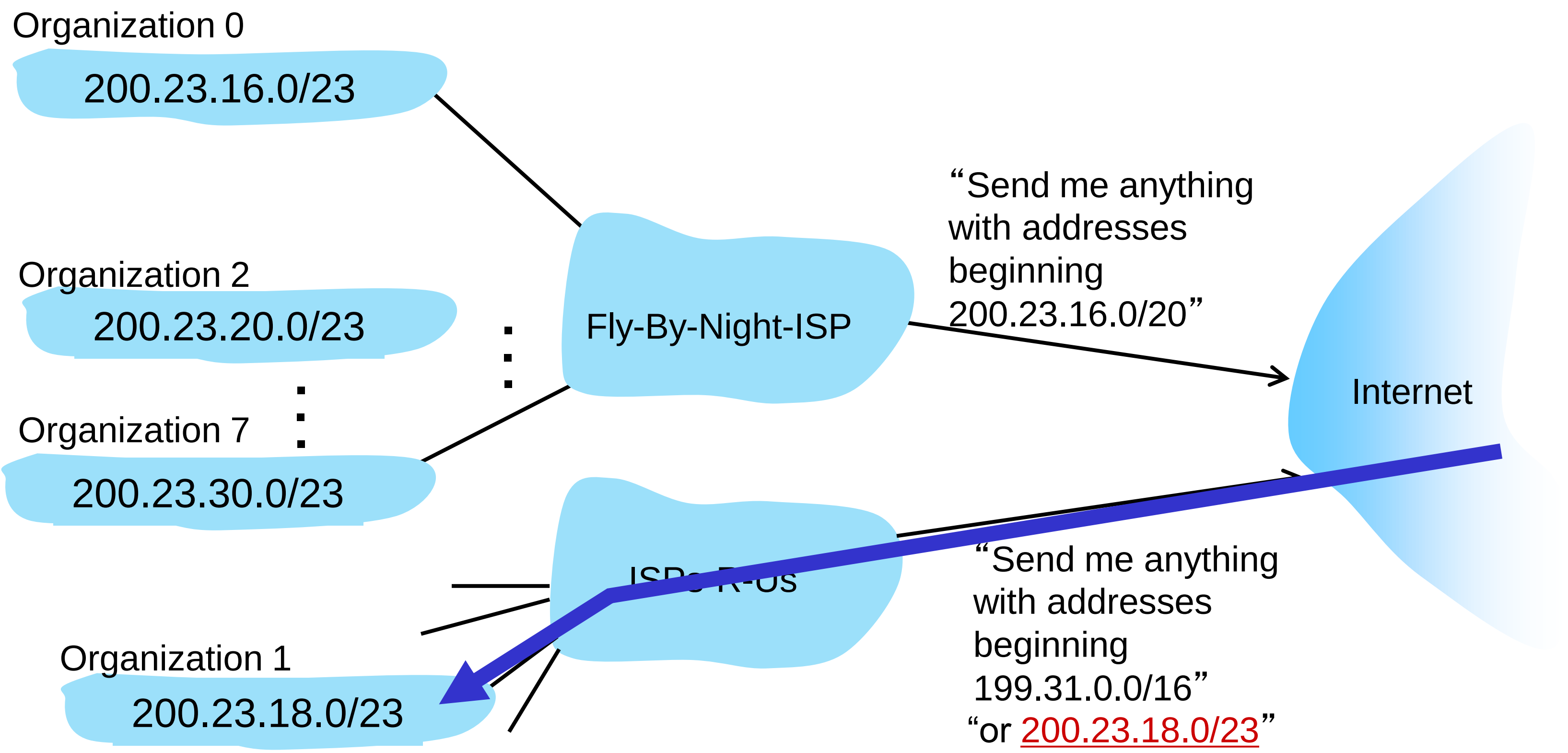
Hierarchical addressing: more specific routes

- 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/

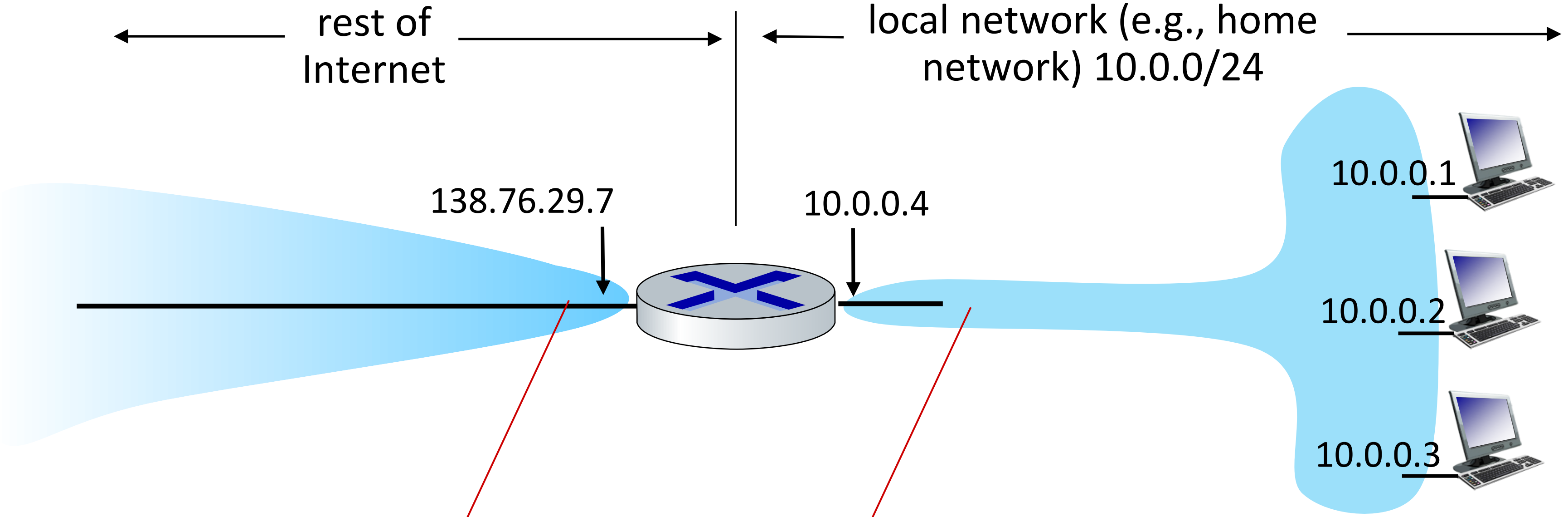
- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , …) management

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

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

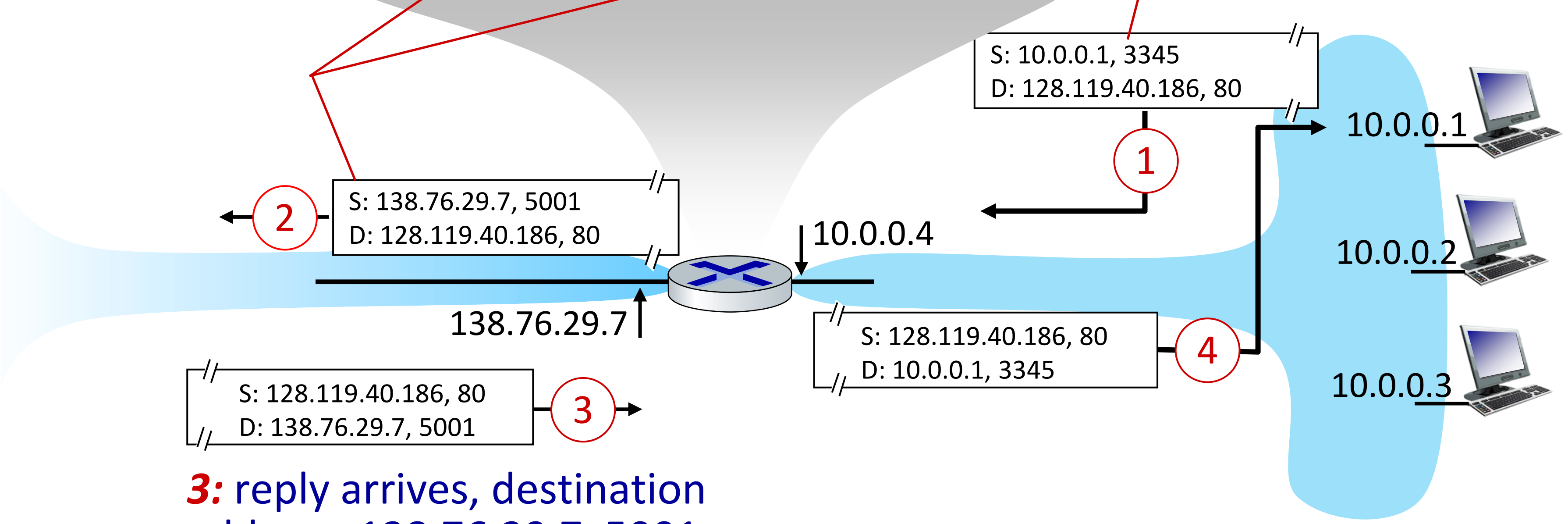
NAT: network address translation

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

NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80



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

Activity 4-4 : Route Summarization

- Given one Network IP address of each subnet, summarize the network addresses to one address
  - 120.41.32.0/22
  - 120.41.36.0/23
  - 120.41.38.0/23
  - 120.41.40.0/21

Solution : 120.41.32.0/20

Activity 4-5 : NAT Traversal

- Discuss in groups
  - Suppose a peer with user name Alice discovers through querying that a peer with user name Bob has a file she wants to download. Also suppose that Bob is behind a NAT whereas Alice is not. Let 138.76.29.7 be the WAN-side address of the NAT and let 10.0.0.1 be the internal IP address for Bob. Assume that the NAT is not specifically configured for the P2P application.
  - Can you translate a single server behind a logical port to a real IP address for internet?
    - If so, how?
    - If not, why?
  - Solution: Yes, you can do a NAT Traversal and perform port forwarding.

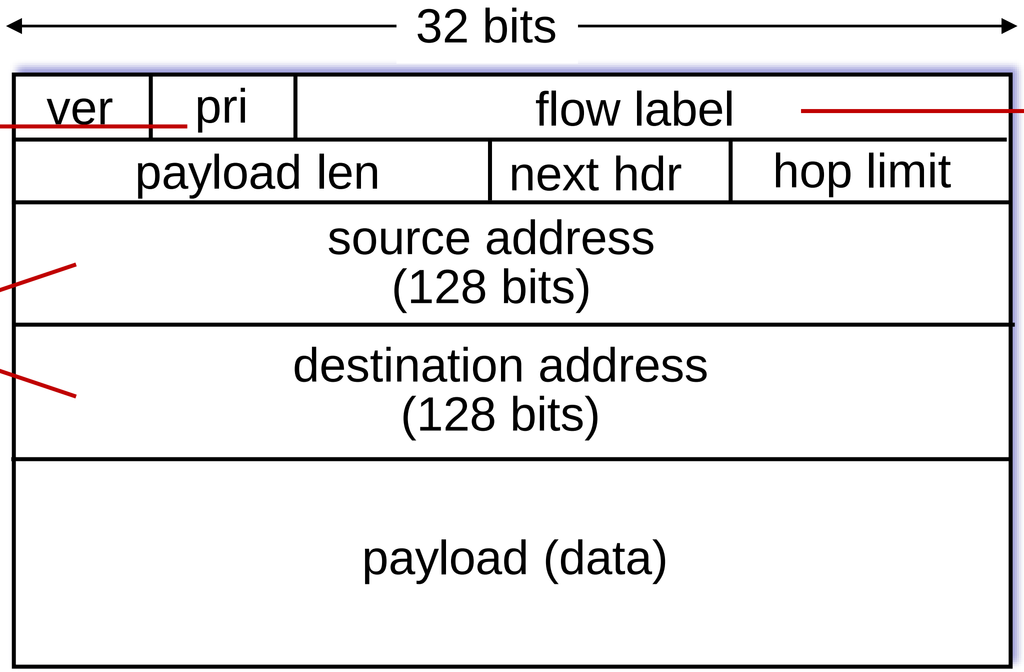
IPv6: motivation

- 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



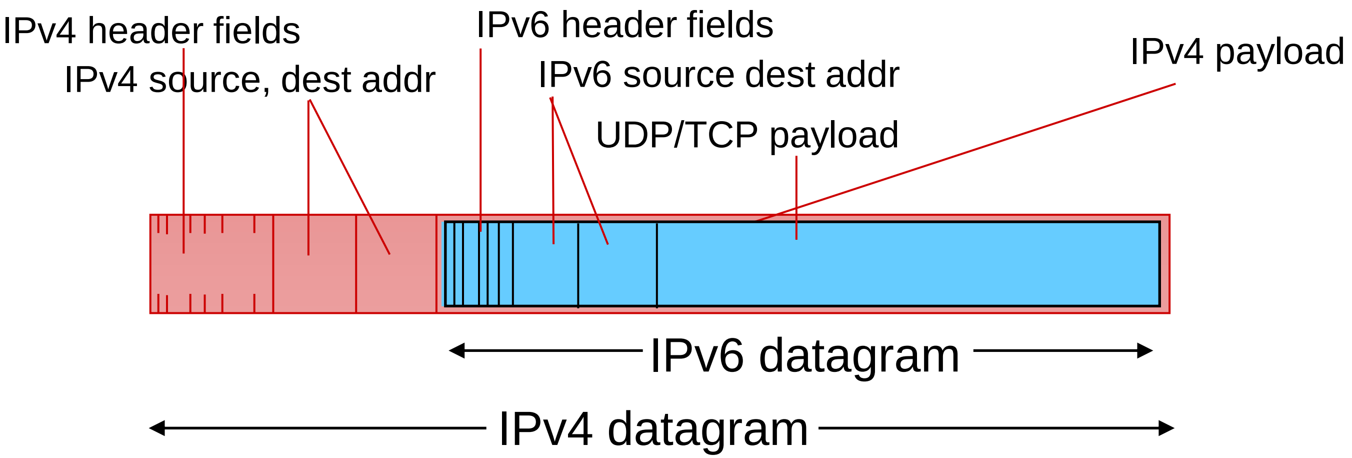
flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

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)

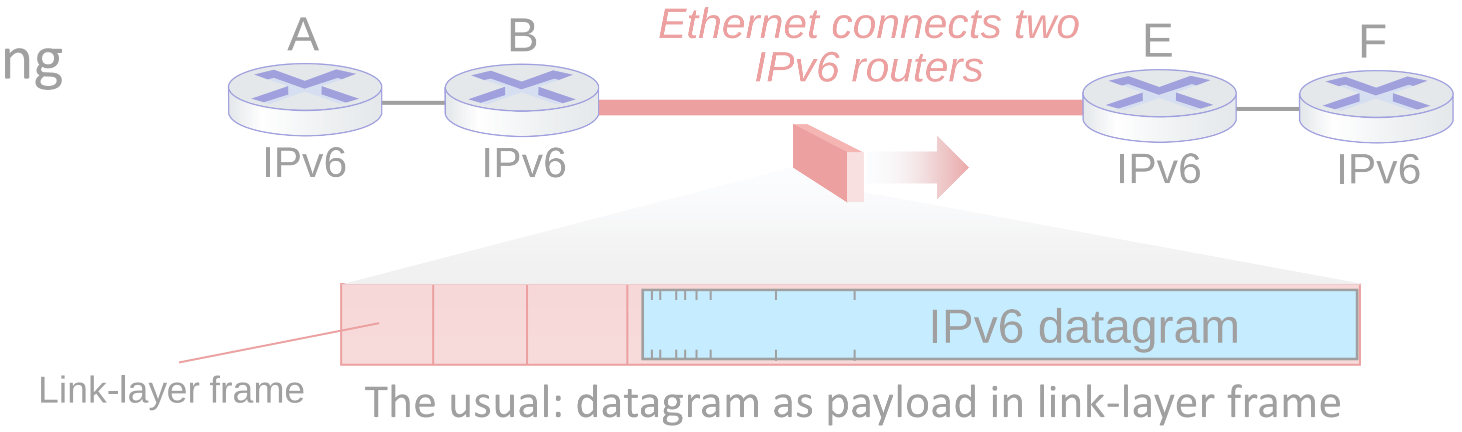
Transition from IPv4 to IPv6

- 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)

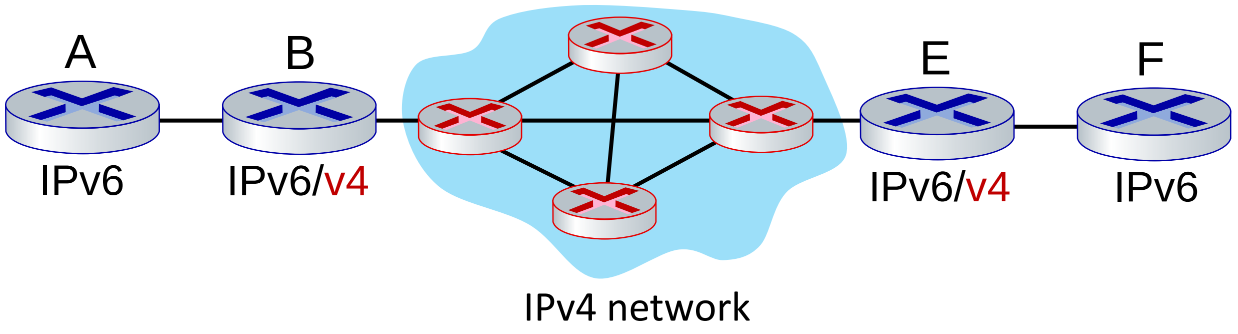


Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

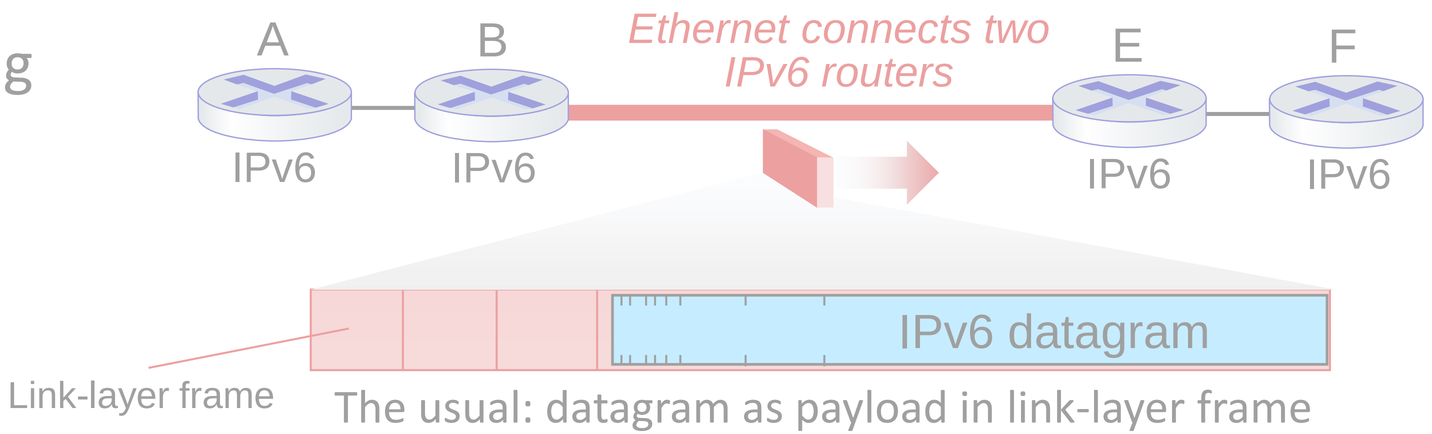


IPv4 network connecting two IPv6 routers



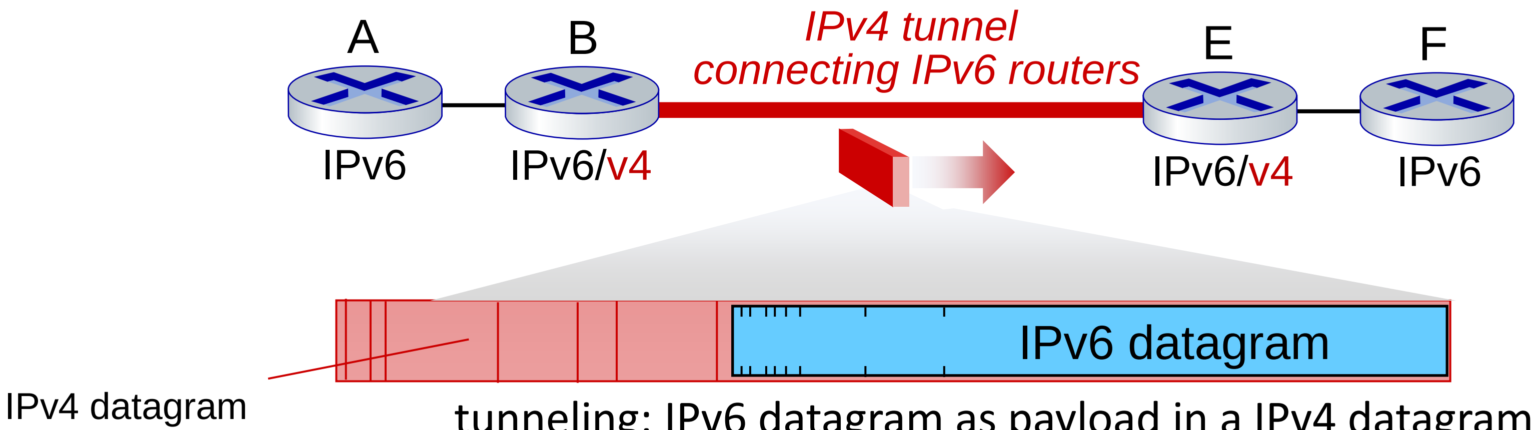
Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



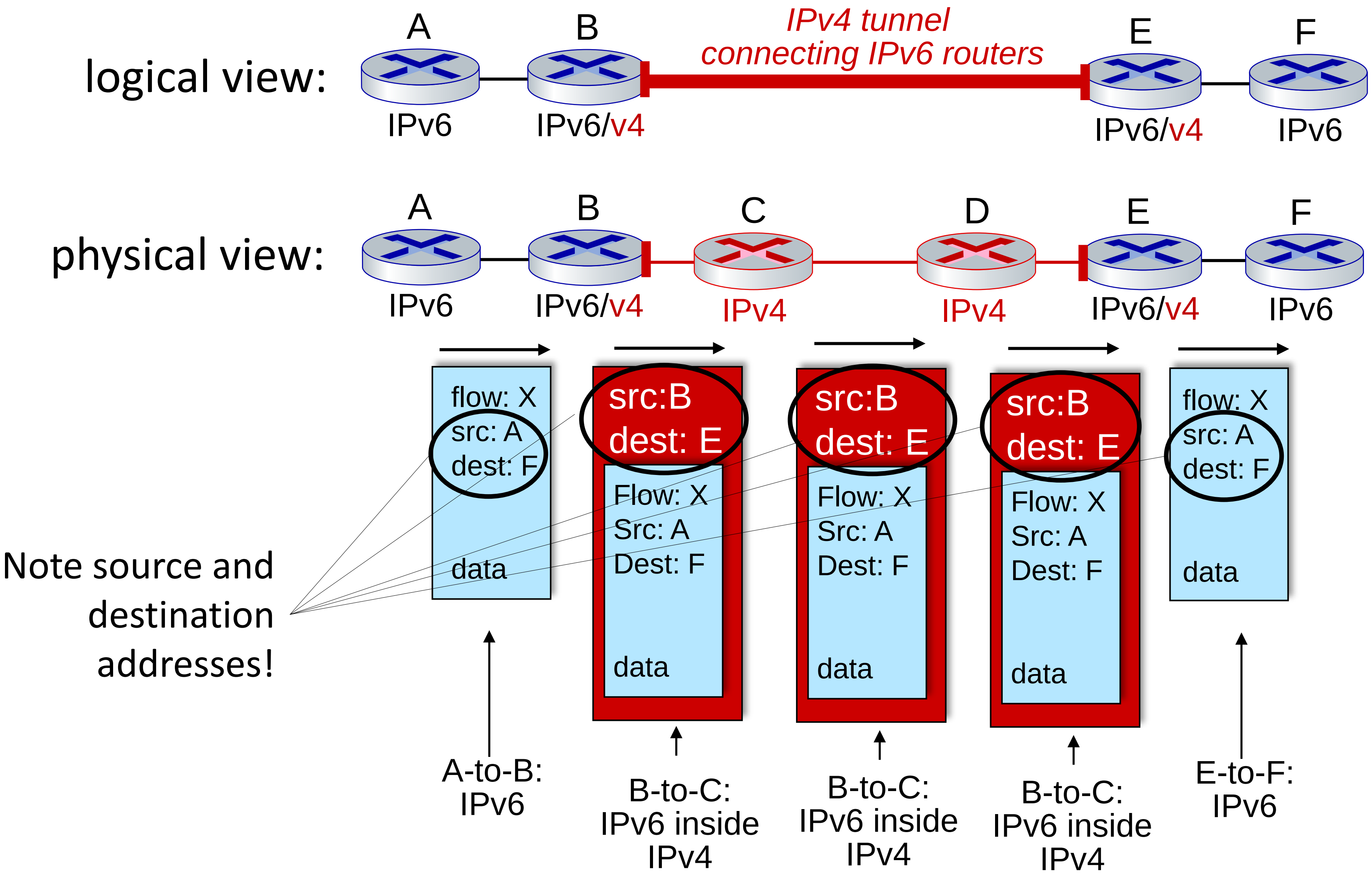
The usual: datagram as payload in link-layer frame

IPv4 tunnel connecting two IPv6 routers



tunneling: IPv6 datagram as payload in a IPv4 datagram

Tunneling

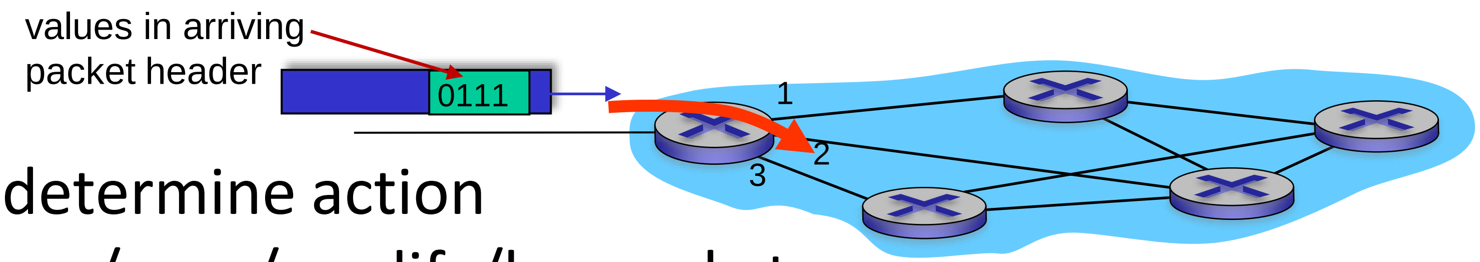


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6/v4
IPv6/v4
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6/v4
IPv4
IPv4
IPv6/v4
IPv6
flow: X
src: A
dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
flow: X
src: A
dest: F
data
Note source and destination addresses!
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

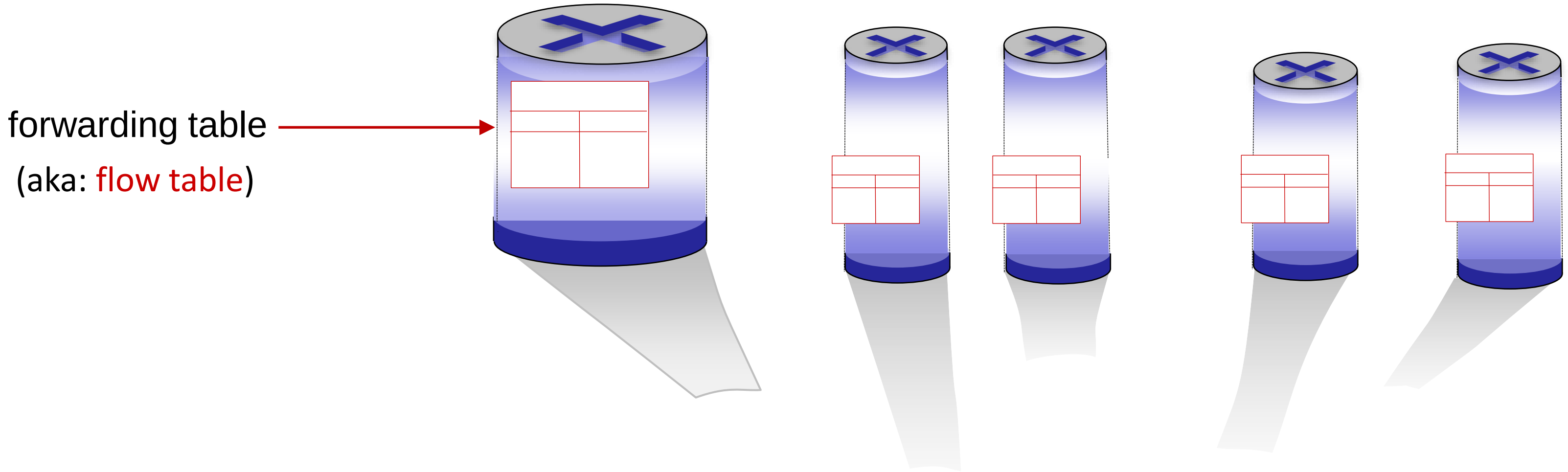
Generalized forwarding: match plus action

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

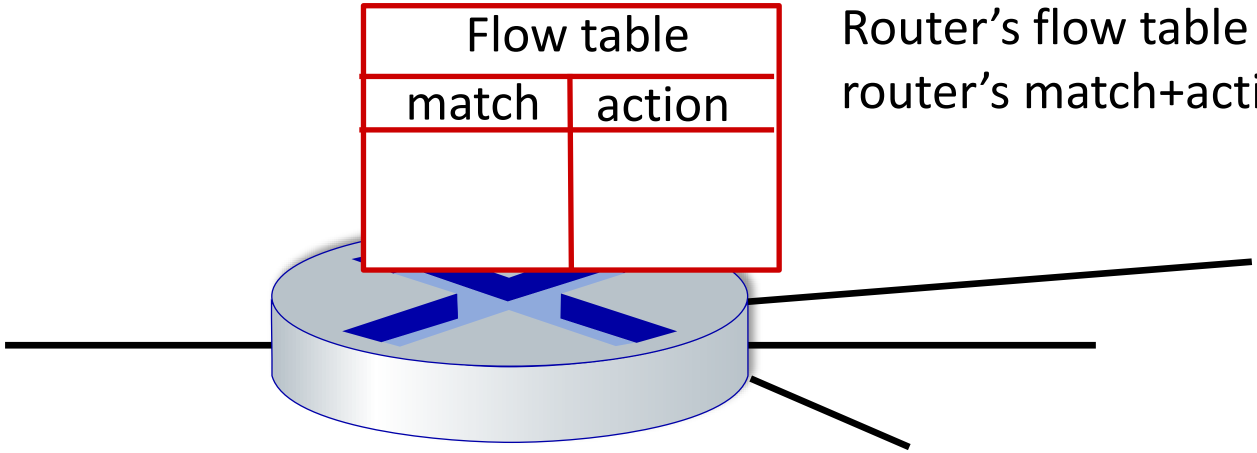


- *generalized forwarding*:
  - many header fields can determine action
  - many action possible: drop/copy/modify/log packet



Flow table abstraction

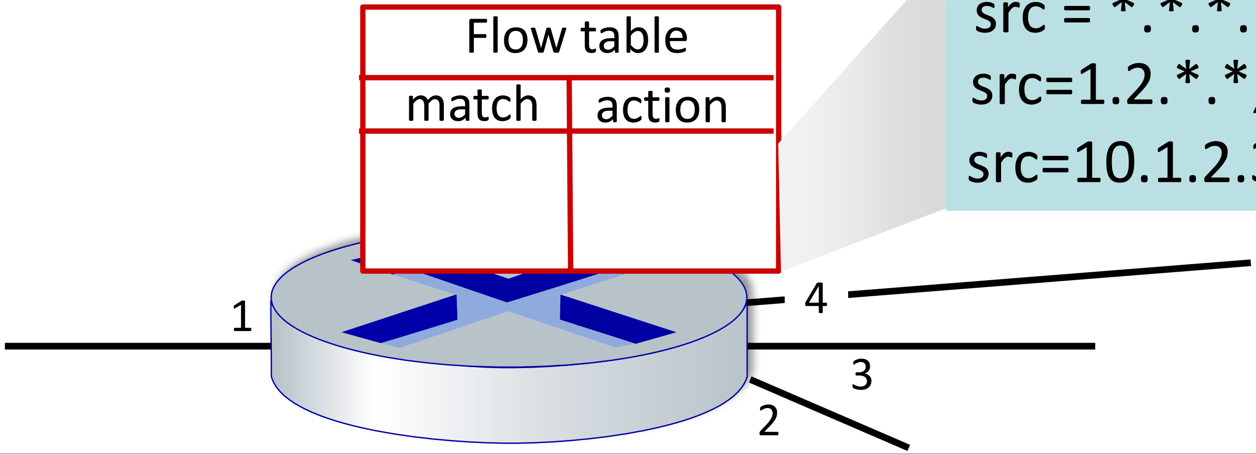
- flow: defined by header field values (in link-, network-, transport-layer fields)
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  -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



Router's flow table define router's match+action rules

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



src = *.*.*.*, dest=3.4.*.*	forward(2)
src=1.2.*.*, dest=*.*.*.*	drop
src=10.1.2.3, dest=*.*.*.*	send to controller

* : wildcard

OpenFlow: flow table entries

Match	Action	Stats

Packet + byte counters

1. Forward packet to port(s)
2. Drop packet
3. Modify fields in header(s)
4. Encapsulate and forward to controller

Header fields to match:

Ingress Port	Src MAC	Dst MAC	Eth Type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Proto	IP ToS	TCP/UDP Src Port	TCP/UDP Dst Port
	Link layer					Network layer				Transport layer	

OpenFlow: examples

SACRAMENTO STATE

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

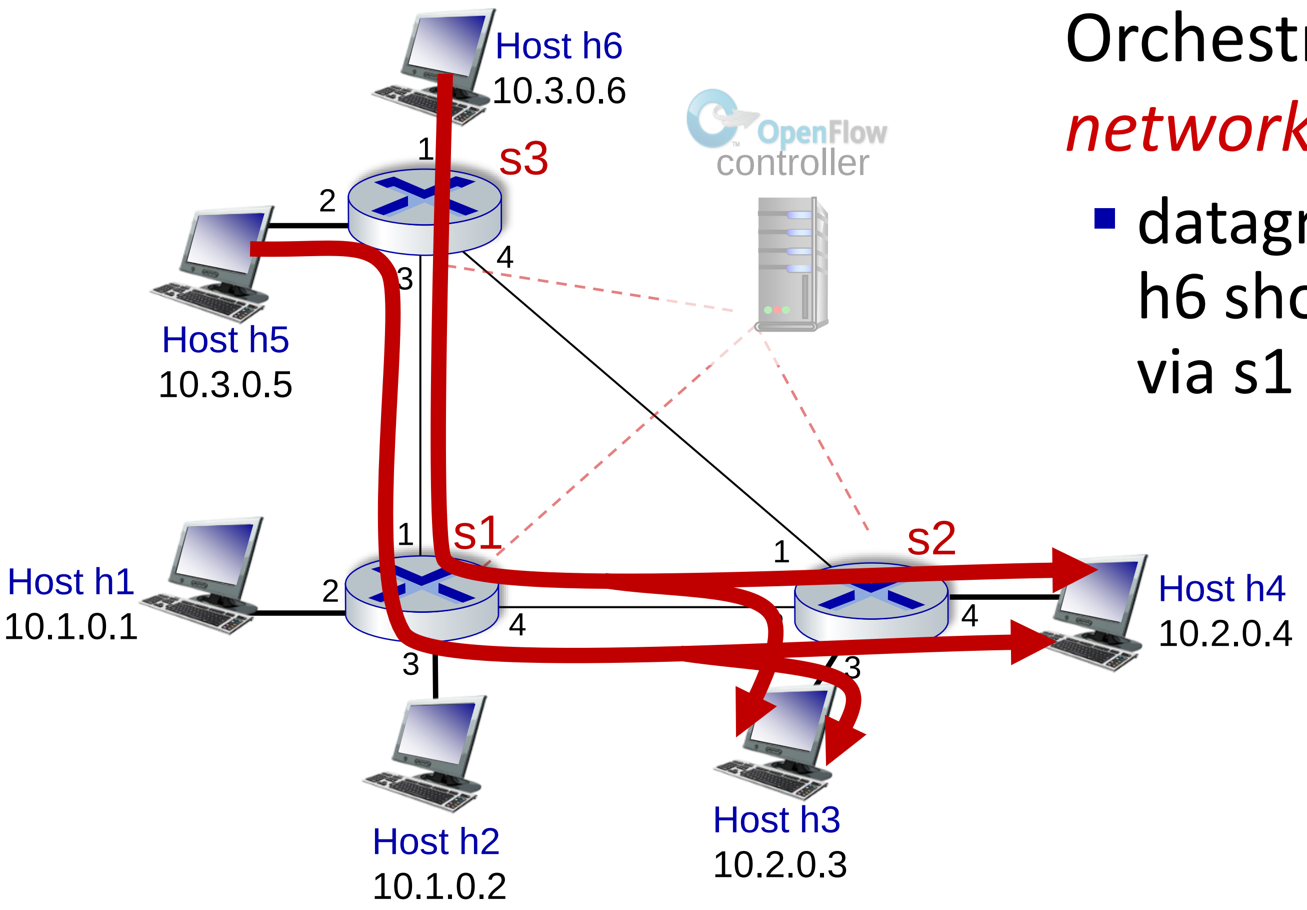
Router
- *match:* longest destination IP prefix
- *action:* forward out a link

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

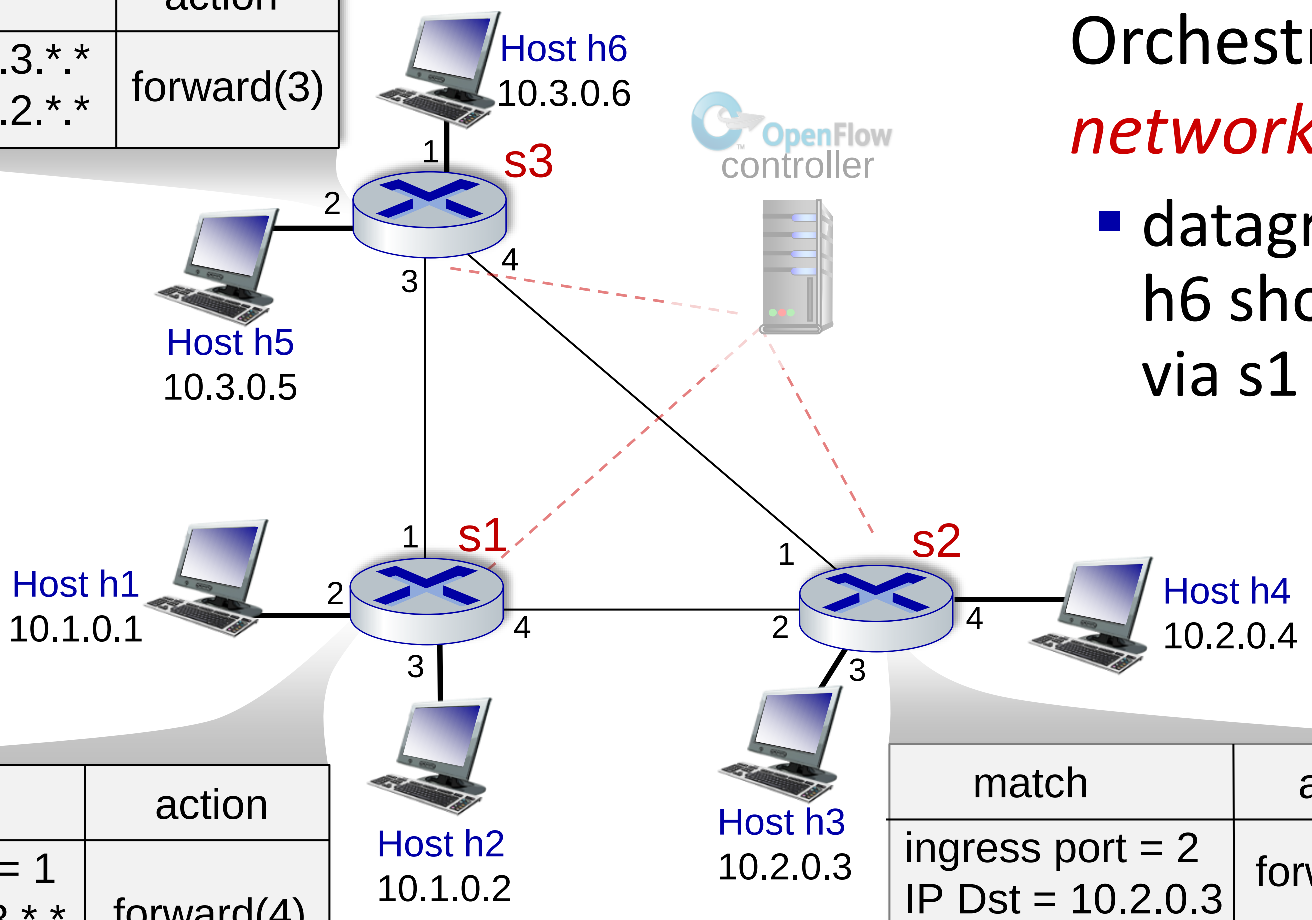


Orchestrated tables can create *network-wide* behavior, e.g.,:

- 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.*.* IP Dst = 10.2.*.*	forward(4)

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

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

Middleboxes

SACRAMENTO STATE

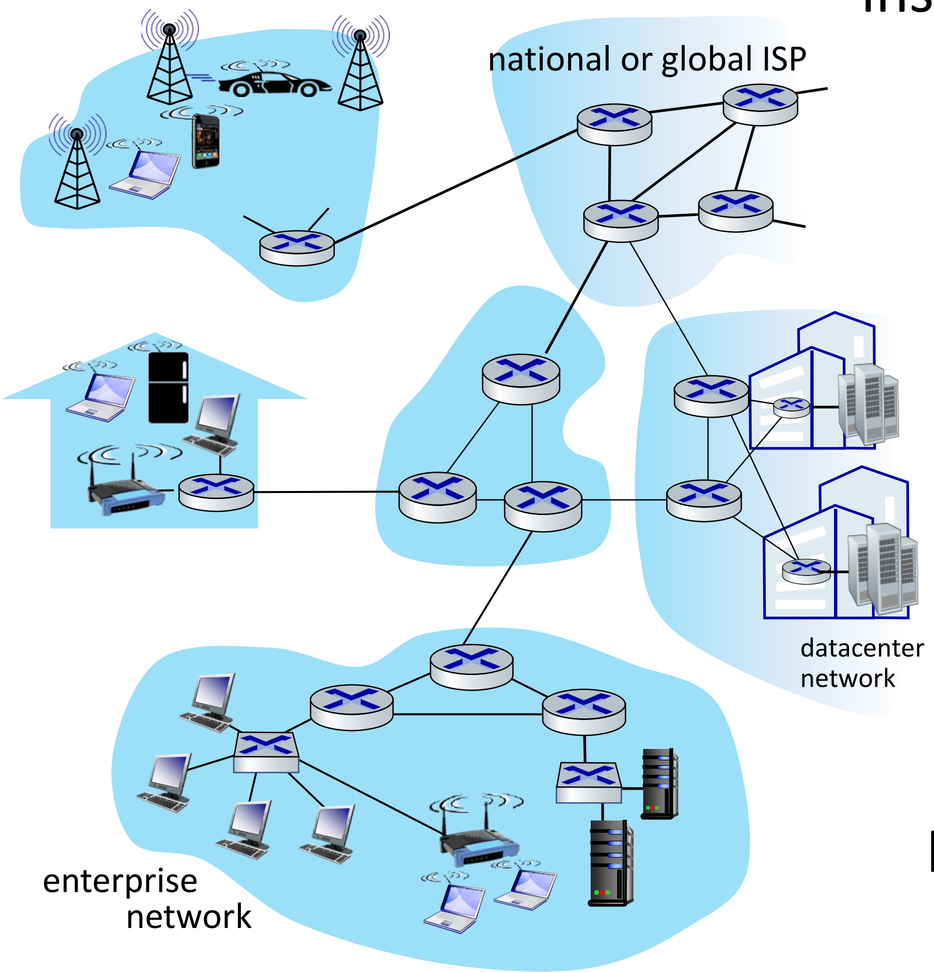
Middlebox (RFC 3234)

"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

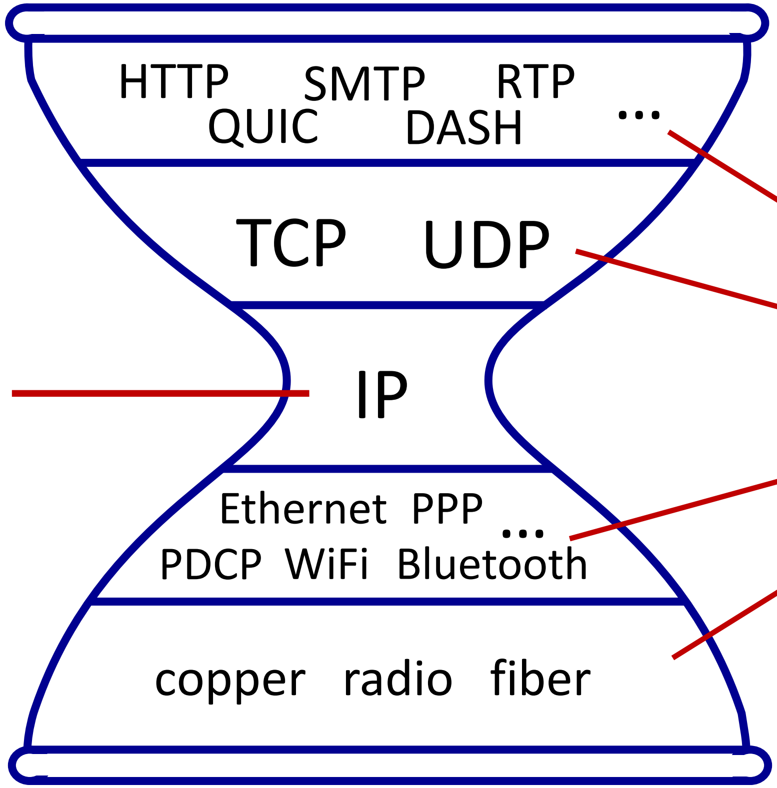
Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs

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

Lecture 4_2 Summary

- Dynamic Host Configuration Protocol
- Route Aggregation
- Network Address Translation
- IPv6 Transition
- Generalized Forwarding Vs OpenFlow
- Middleboxes
- IP Hourglass