# Chapter 4
# Network Layer

**Computer Networking: A Top Down Approach**
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
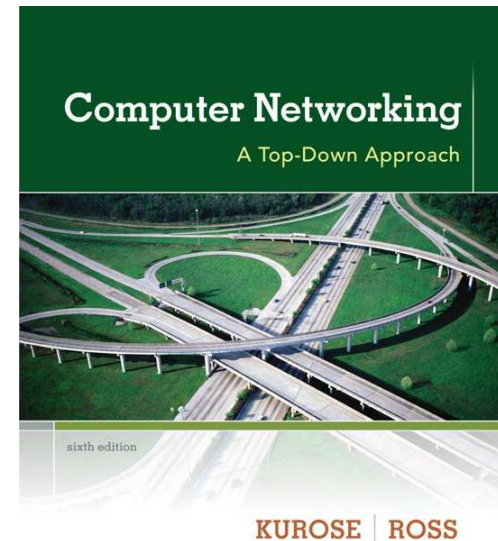March 2012

A note on the use of these ppt 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

Slides modified by M. Karsten.

# Chapter 4: network layer

*chapter goals:*

- ❖ understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - routing (path selection)
  - broadcast, multicast
- ❖ instantiation, implementation in the Internet

# Chapter 4: outline

# Network layer

❖ **transport segment from sending to receiving host**

❖ **on sending side encapsulates segments into datagrams**

❖ **on receiving side, delivers segments to transport layer**

❖ **network layer protocols in *every* host, router**

❖ **router examines header fields in all IP datagrams passing through it**
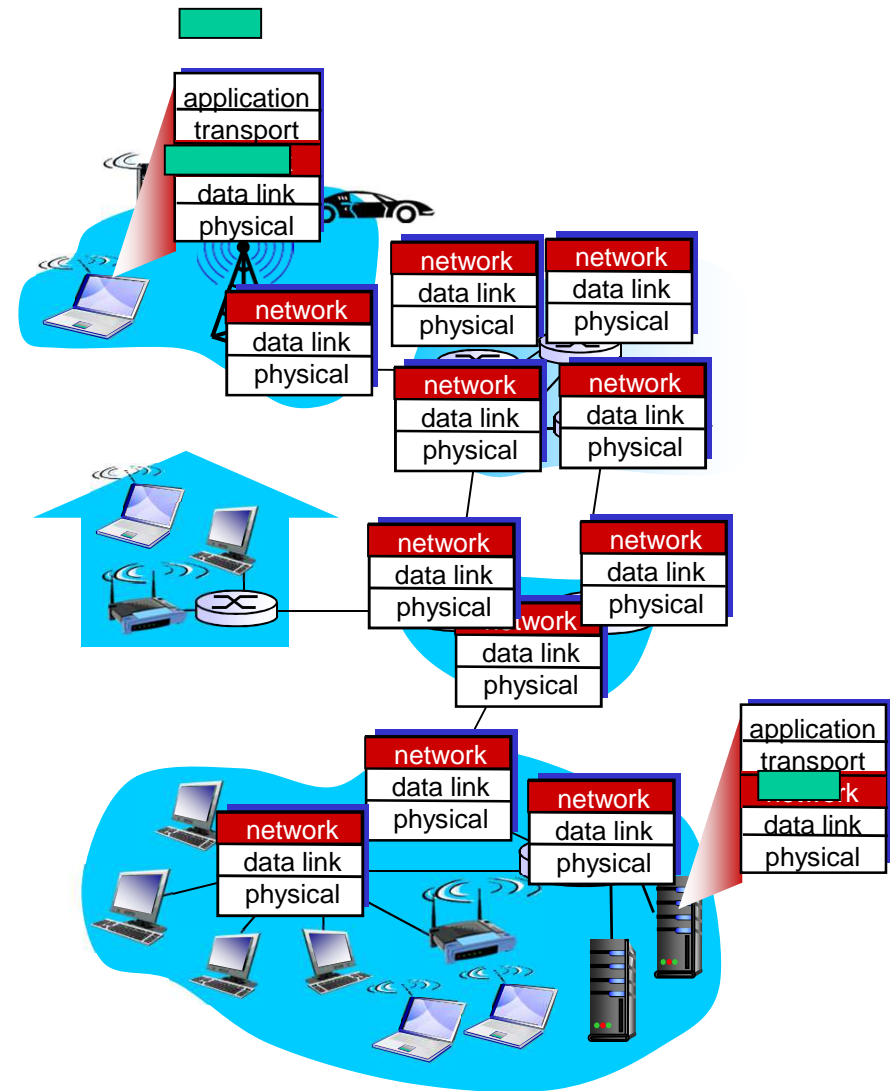
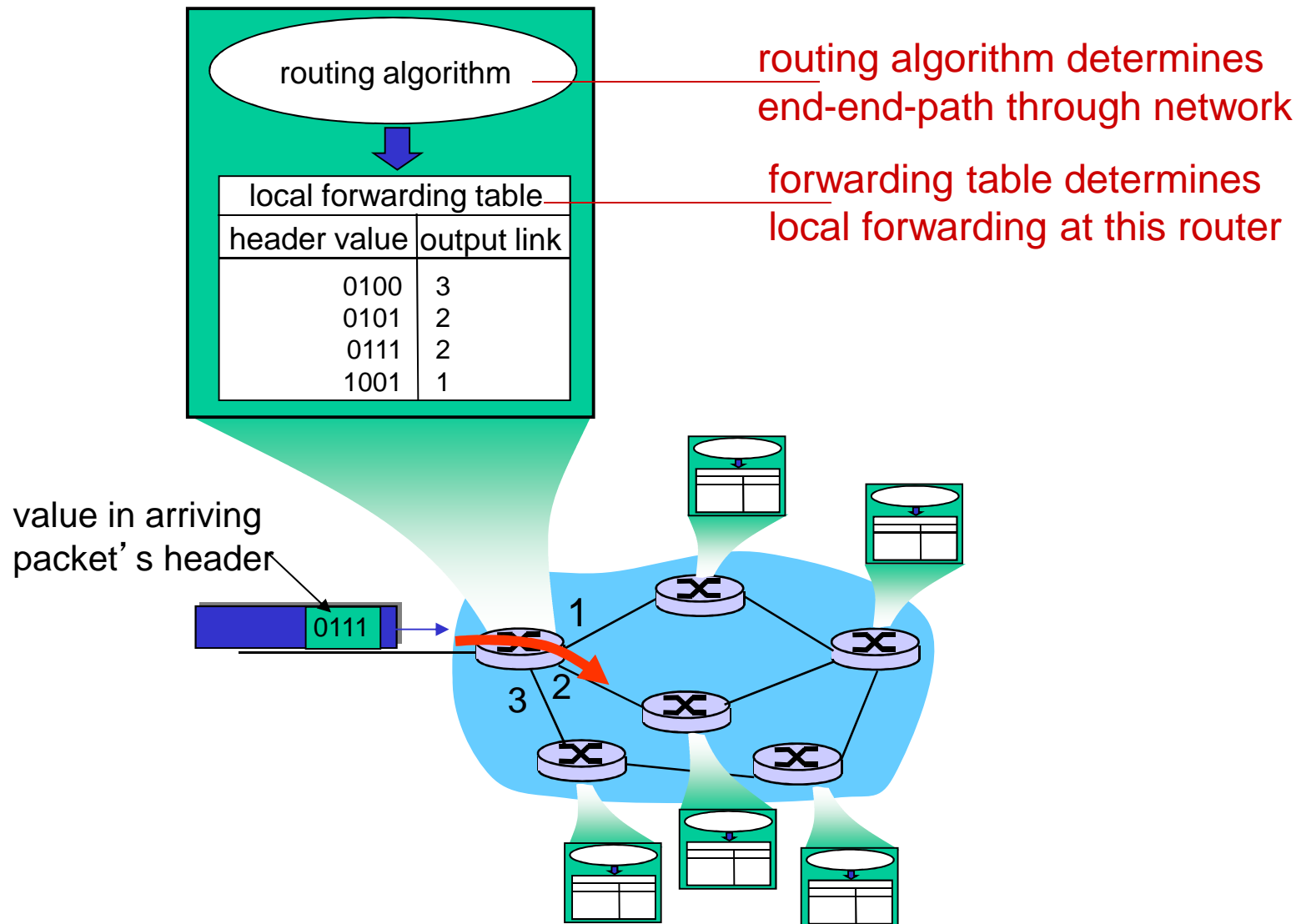# Two key network-layer functions

❖ *forwarding:* move packets from router's input to appropriate router output

❖ *routing:* determine route taken by packets from source to dest.

  ▪ *routing algorithms*

*analogy:*

❖ *routing:* process of planning trip from source to dest

❖ *forwarding:* process of getting through single interchange

# Interplay between routing and forwarding



routing algorithm

local forwarding table

| header value | output link |
|---|---|
| 0100 | 3 |
| 0101 | 2 |
| 0111 | 2 |
| 1001 | 1 |

routing algorithm determines
end-end-path through network

forwarding table determines
local forwarding at this router

value in arriving
packet's header

0111

1

3  2

# Connection setup

❖ 3<sup>rd</sup> important function in *some* network architectures:
  ▪ ATM, frame relay, X.25

❖ before datagrams flow, two end hosts *and* intervening routers establish virtual connection
  ▪ routers get involved

❖ network vs transport layer connection service:
  ▪ *network:* between two hosts (may also involve intervening routers in case of VCs)
  ▪ *transport:* between two processes

# 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 models:

| Network Architecture | Service Model | Guarantees ? | | | | Congestion feedback |
|---|---|---|---|---|---|---|
| | | Bandwidth | Loss | Order | Timing | |
| Internet | best effort | none | no | no | no | no (inferred via loss) |
| ATM | CBR | constant rate | yes | yes | yes | no congestion |
| ATM | VBR | guaranteed rate | yes | yes | yes | no congestion |
| ATM | ABR | guaranteed minimum | no | yes | no | yes |
| ATM | UBR | none | no | yes | no | no |

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# Connection, connection-less service

❖ *datagram* network provides network-layer *connectionless* service

❖ *virtual-circuit* network provides network-layer *connection* service

❖ analogous to TCP/UDP connecton-oriented / connectionless transport-layer services, but:

- *service:* host-to-host
- *no choice:* network provides one or the other
- *implementation:* in network core

# Virtual circuits

> "source-to-dest path behaves much like telephone circuit"
>   - performance-wise
>   - network actions along source-to-dest path

- ❖ call setup, teardown for each call *before* data can flow
- ❖ each packet carries VC identifier (not destination host address)
- ❖ *every* router on source-dest path maintains "state" for each passing connection
- ❖ link, router resources (bandwidth, buffers) may be *allocated* to VC (dedicated resources = predictable service)

# VC implementation

*a VC consists of:*

1. *path* from source to destination
2. *VC numbers*, one number for each link along path
3. *entries in forwarding tables* in routers along path

❖ packet belonging to VC carries VC number (rather than dest address)

❖ VC number can be changed on each link.

■ new VC number comes from forwarding table
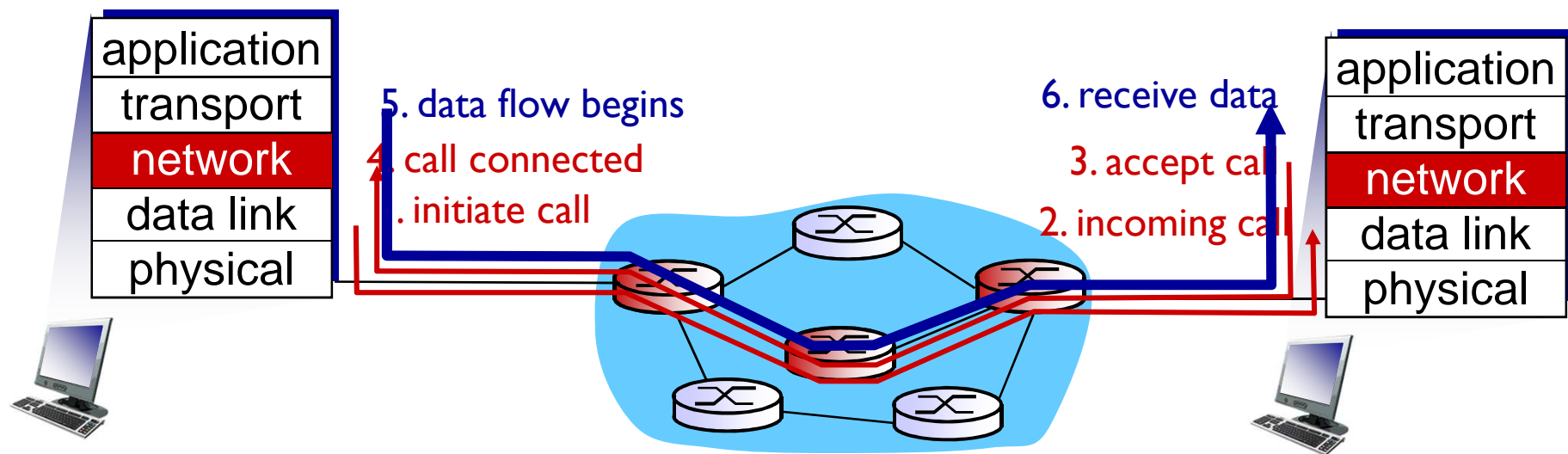
# VC forwarding table
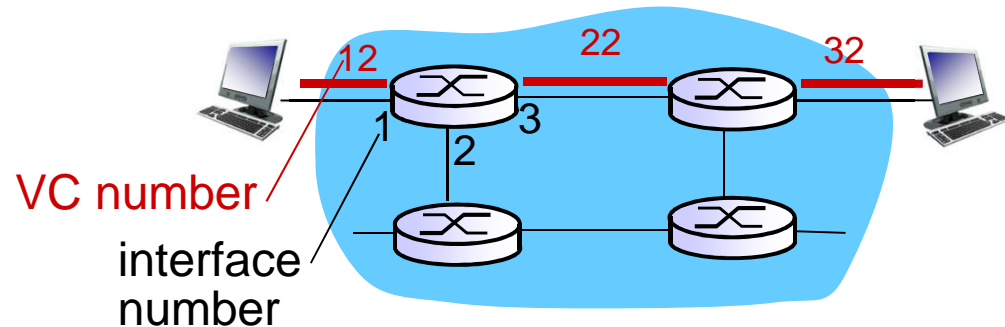


*forwarding table in northwest router:*

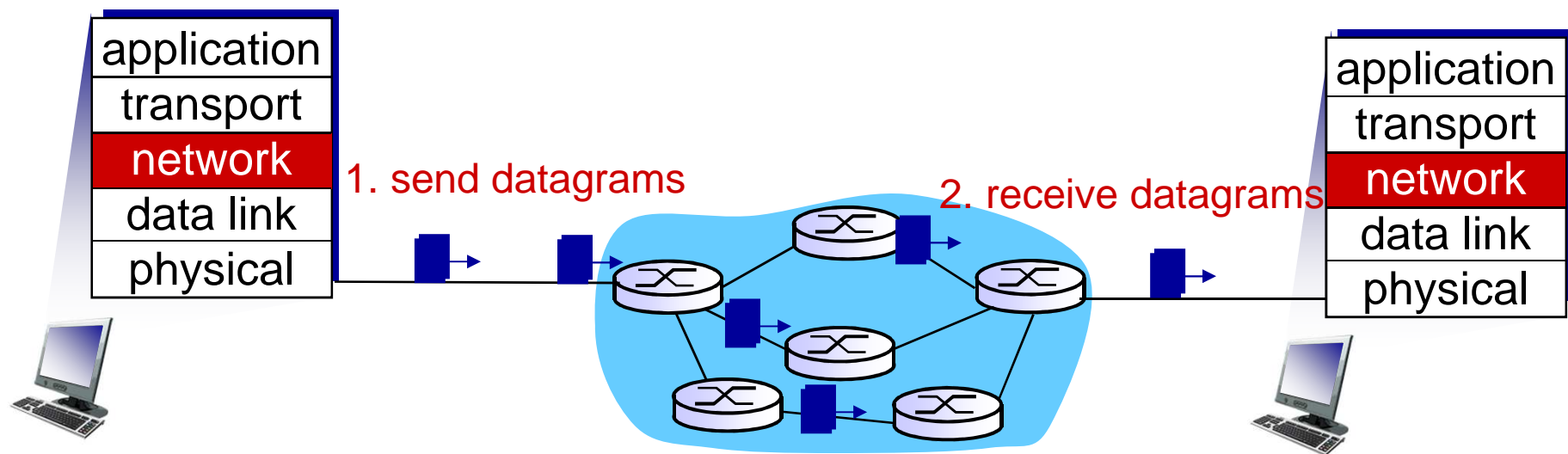| Incoming interface | Incoming VC # | Outgoing interface | Outgoing VC # |
| --- | --- | --- | --- |
| 1 | 12 | 3 | 22 |
| 2 | 63 | 1 | 18 |
| 3 | 7 | 2 | 17 |
| 1 | 97 | 3 | 87 |
| … | … | … | … |

*VC routers maintain connection state information!*

# Virtual circuits: signaling protocols

❖ used to setup, maintain  teardown VC
❖ used in ATM, frame-relay, X.25
❖ not used in today's Internet



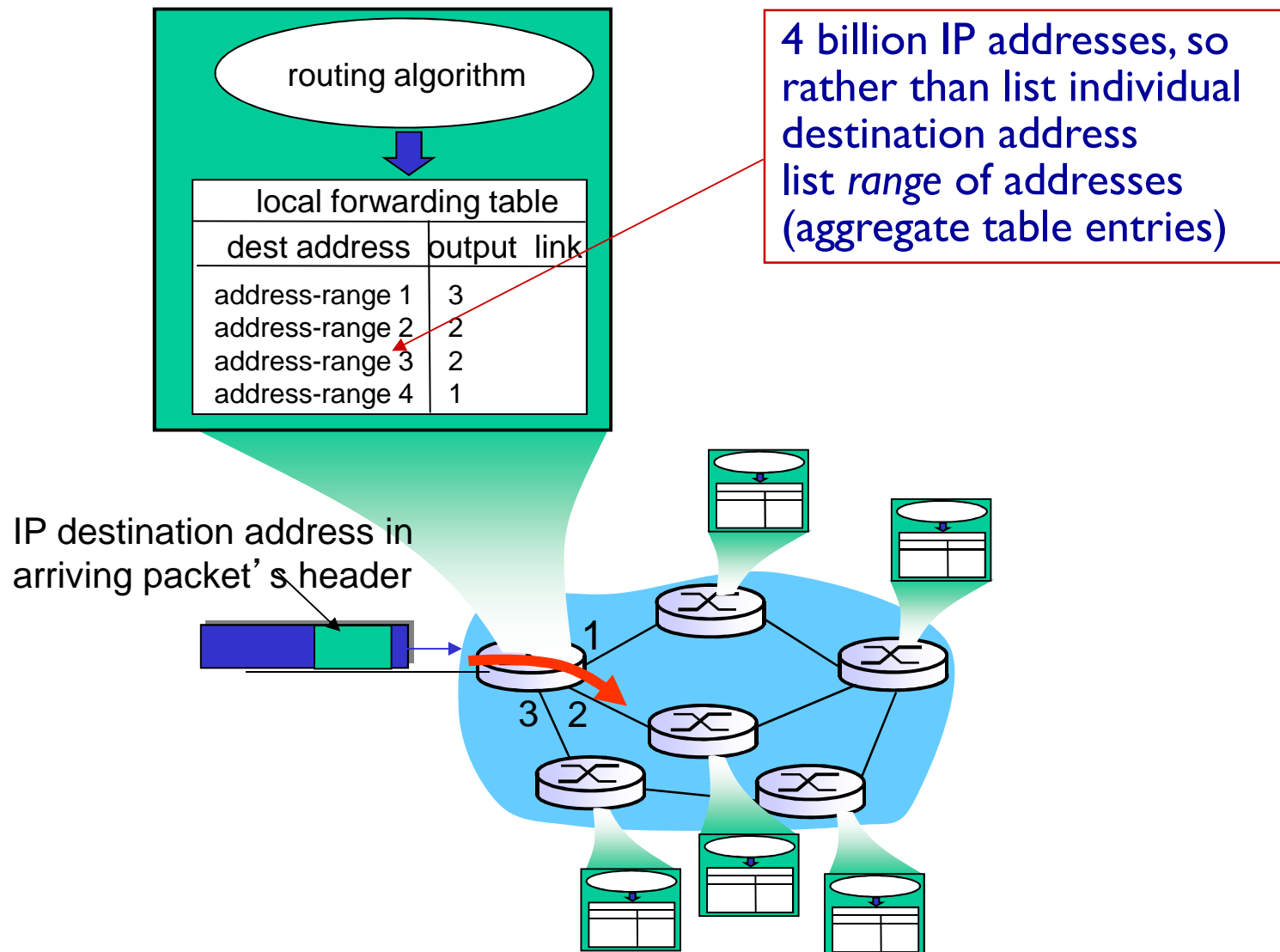| application |
| transport |
| network |
| data link |
| physical |

5. data flow begins

4. call connected

1. initiate call

6. receive data

3. accept call

2. incoming call

| application |
| transport |
| network |
| data link |
| physical |

# Datagram networks

❖ no call setup at network layer

❖ routers: no state about end-to-end connections
   ▪ no network-level concept of "connection"

❖ packets forwarded using destination host address

| application |
| transport |
| network |
| data link |
| physical |

1. send datagrams

2. receive datagrams

| application |
| transport |
| network |
| data link |
| physical |

# Datagram forwarding  table



routing algorithm

| local forwarding table | |
| --- | --- |
| dest address | output  link |
| address-range 1 | 3 |
| address-range 2 | 2 |
| address-range 3 | 2 |
| address-range 4 | 1 |

4 billion IP addresses, so rather than list individual destination address list *range* of addresses (aggregate table entries)

IP destination address in arriving packet's header

1

3  2

# Datagram forwarding  table

| Destination Address Range | Link Interface |
|---|---|
| 11001000 00010111 00010000 00000000<br>through<br>11001000 00010111 00010111 11111111 | 0 |
| 11001000 00010111 00011000 00000000<br>through<br>11001000 00010111 00011000 11111111 | 1 |
| 11001000 00010111 00011001 00000000<br>through<br>11001000 00010111 00011111 11111111 | 2 |
| otherwise | 3 |

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

# Longest prefix matching

*longest prefix matching* ————————————————

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:

DA: 11001000  00010111  00010110  10100001    which interface?

DA: 11001000  00010111  00011000  10101010    which interface?

# Datagram or VC network: why?

## Internet (datagram)

- ❖ data exchange among computers
  - ▪ "elastic" service, no strict timing req.
- ❖ many link types
  - ▪ different characteristics
  - ▪ uniform service difficult
- ❖ "smart" end systems (computers)
  - ▪ can adapt, perform control, error recovery
  - ▪ *simple inside network, complexity at "edge"*

## ATM (VC)

- ❖ evolved from telephony
- ❖ human conversation:
  - ▪ strict timing, reliability requirements
  - ▪ need for guaranteed service
- ❖ "dumb" end systems
  - ▪ telephones
  - ▪ *complexity inside network*

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

<span style="color:red">4.3 what's inside a router</span>

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
- distance vector
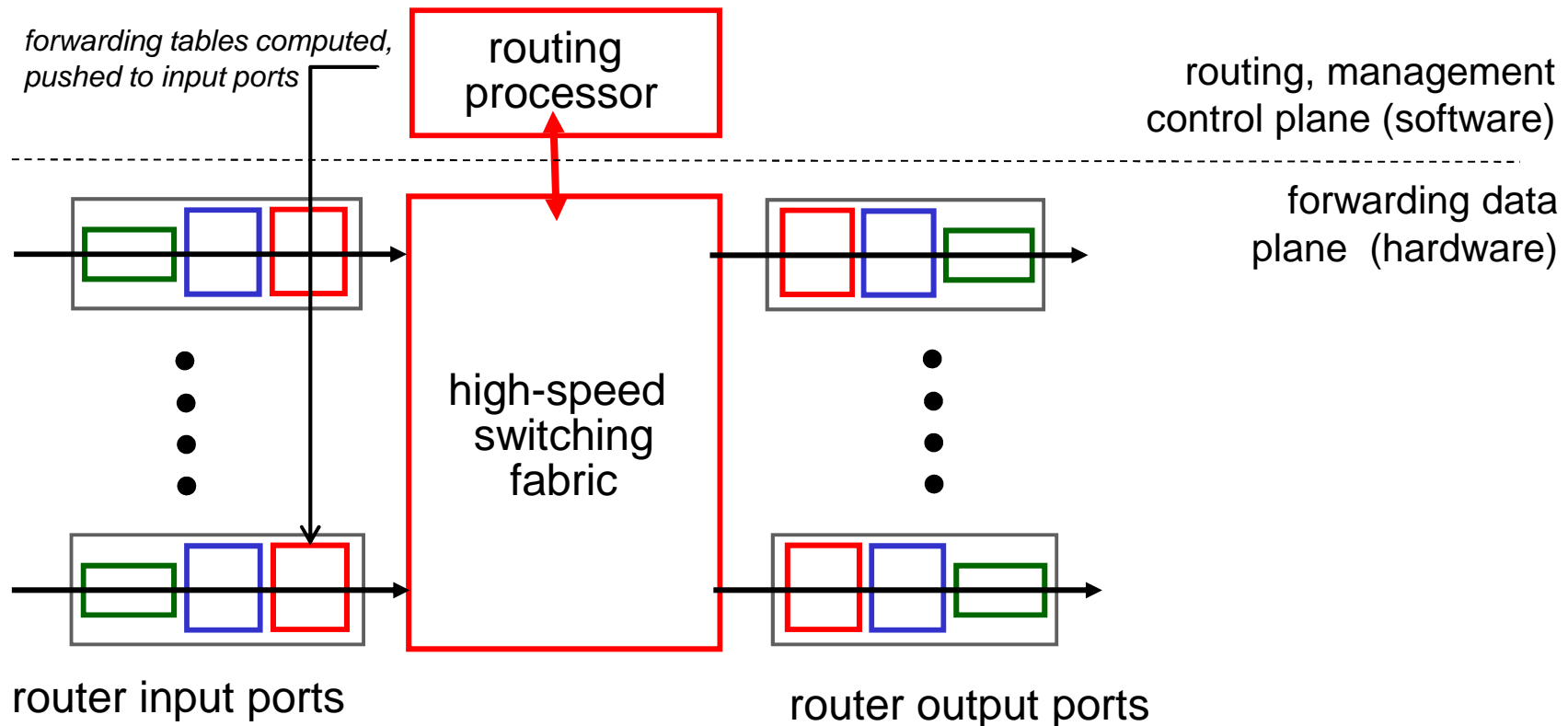- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP
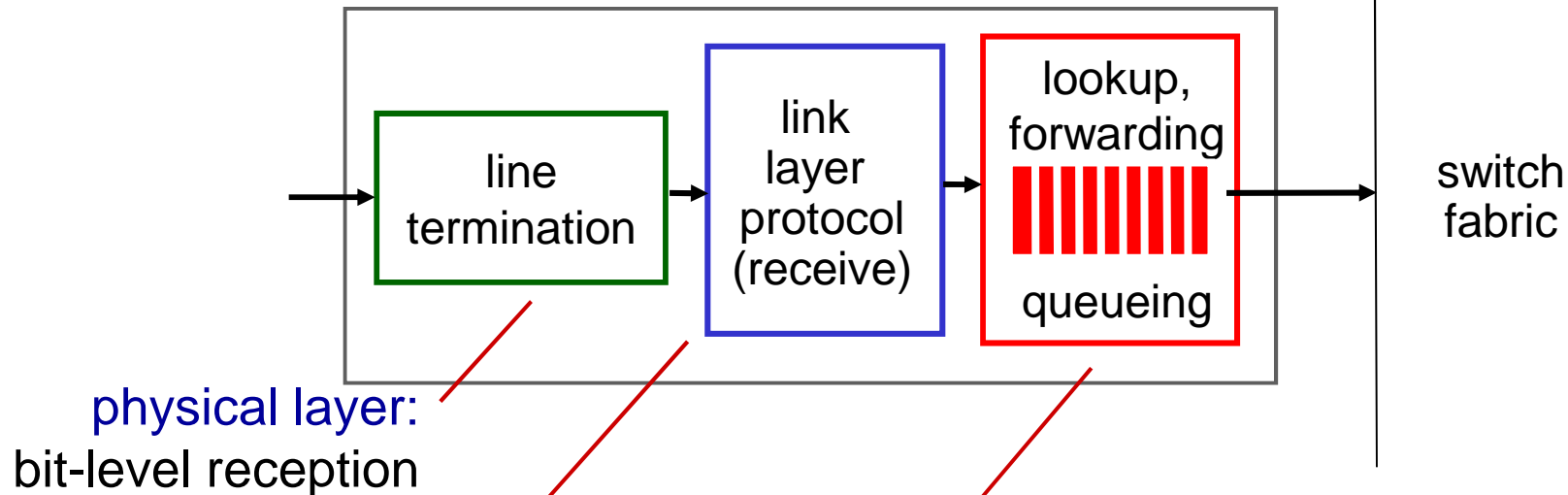
4.7 broadcast and multicast routing

# Router architecture overview

two key router functions:

❖ run routing algorithms/protocol (RIP, OSPF, BGP)

❖ *forwarding* datagrams from incoming to outgoing link

*forwarding tables computed, pushed to input ports*

routing processor

routing, management control plane (software)

high-speed switching fabric

forwarding data plane (hardware)

router input ports

router output ports

# Input port functions



**physical layer:**
bit-level reception

**data link layer:**
   e.g., Ethernet
   see chapter 5

**decentralized switching:**

- given datagram dest., lookup output port using forwarding table in input port memory (*"match plus action"*)
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

# Switching fabrics

❖ **transfer packet from input buffer to appropriate output buffer**

❖ **switching rate: rate at which packets can be transfer from inputs to outputs**

  ▪ often measured as multiple of input/output line rate
  ▪ N inputs: switching rate N times line rate desirable

❖ **three types of switching fabrics**

memory                    bus                    crossbar
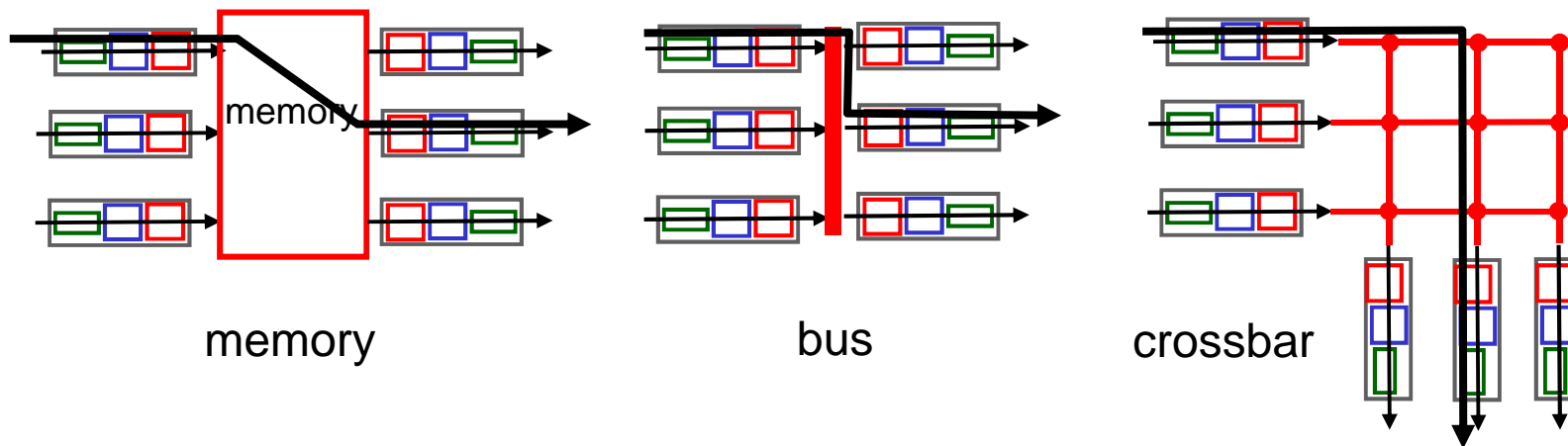
# Switching via memory

*first generation routers:*

- ❖ traditional computers with switching under direct control of CPU

- ❖ packet copied to system's memory

- ❖ speed limited by memory bandwidth (2 bus crossings per datagram)
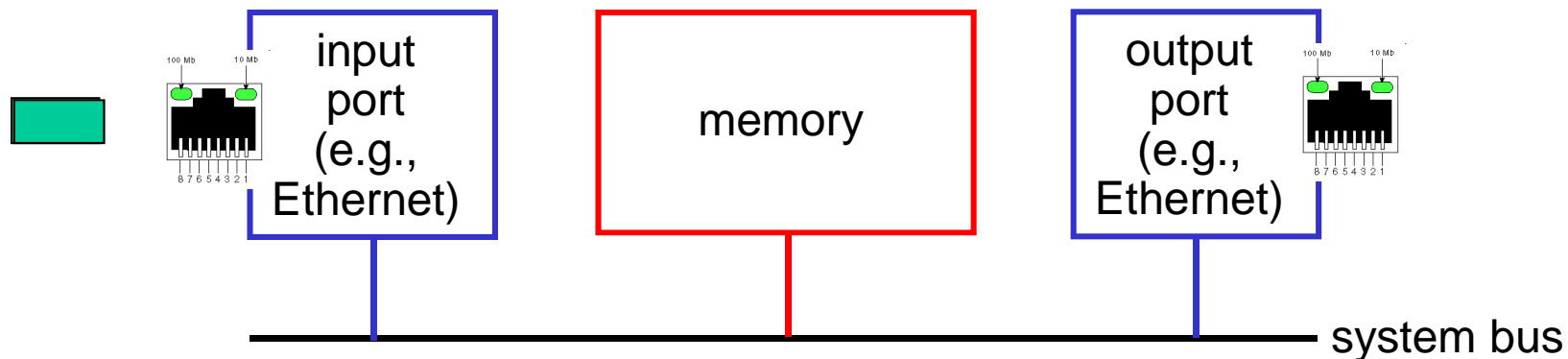
# Switching via a bus

❖ datagram from input port memory
   to output port memory via a
   shared bus

❖ *bus contention:* switching speed
   limited by bus bandwidth

❖ 32 Gbps bus, Cisco 5600: sufficient
   speed for access and enterprise
   routers

bus
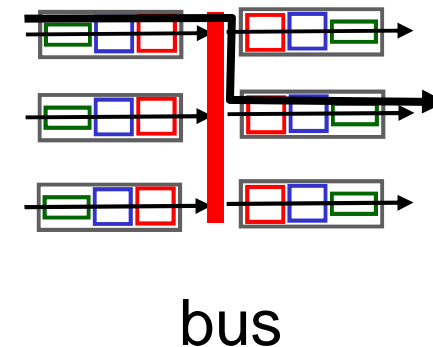
# Switching via interconnection network

❖ overcome  bus bandwidth limitations

❖ banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor

❖ advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.

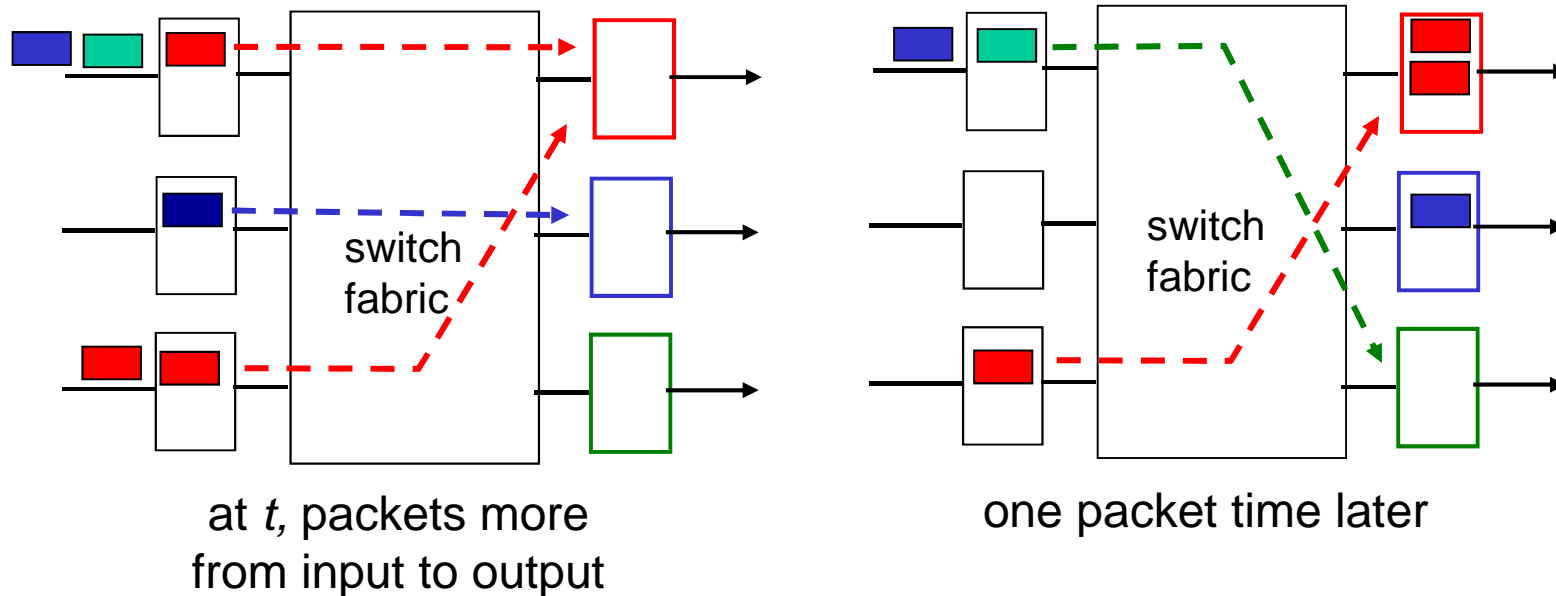❖ Cisco 12000: switches 60 Gbps through the interconnection network

crossbar
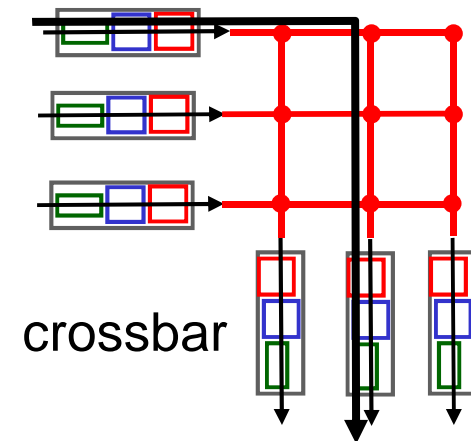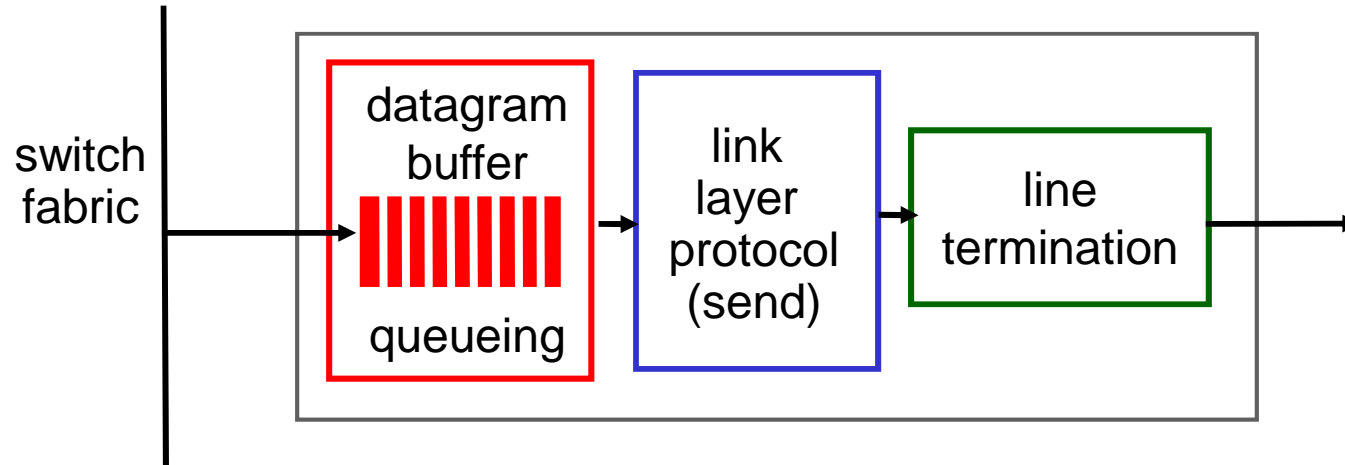
# Output ports



- ❖ *buffering* required when datagrams arrive from fabric faster than the transmission rate
- ❖ *scheduling discipline* chooses among queued datagrams for transmission

# Output port queueing



at *t,* packets more
from input to output

one packet time later

❖ **buffering when arrival rate via switch exceeds
output line speed**

❖ *queueing (delay) and loss due to output port buffer
overflow!*

# Input port queuing

- ❖ 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
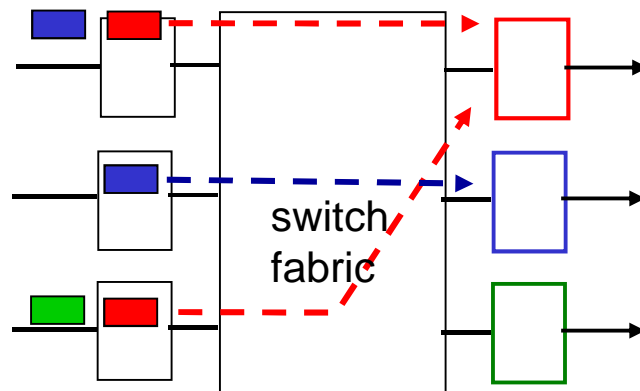


output port contention:
only one red datagram can be transferred.
*lower red packet is blocked*
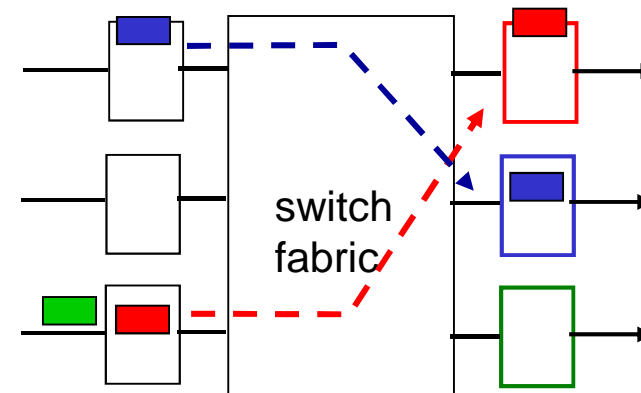
one packet time later:
green packet
experiences HOL
blocking

# 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 Gpbs link: 2.5 Gbit buffer
* ❖ recent recommendation: with $N$ flows, buffering equal to

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

# Chapter 4: outline

# The Internet network layer

host, router network layer functions:

| transport layer: TCP, UDP |
|---|

**network layer**

*routing protocols*
- path selection
- RIP, OSPF, BGP

forwarding table

*IP protocol*
- addressing conventions
- datagram format
- packet handling conventions

*ICMP protocol*
- error reporting
- router "signaling"

| link layer |
|---|

| physical layer |
|---|

# IP datagram format

IP protocol version number

header length (bytes)

"type" of data

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to

total datagram length (bytes)

for fragmentation/ reassembly

e.g. timestamp, record route taken, specify list of routers to visit.

32 bits

| ver | head. len | type of service | length | |
|-----|-----------|-----------------|--------|---|
| 16-bit identifier | | | flgs | fragment offset |
| time to live | upper layer | | header checksum | |
| 32 bit source IP address | | | | |
| 32 bit destination IP address | | | | |
| options (if any) | | | | |
| data (variable length, typically a TCP or UDP segment) | | | | |

*how much overhead?*
- ❖ 20 bytes of TCP
- ❖ 20 bytes of IP
- ❖ = 40 bytes + app layer overhead
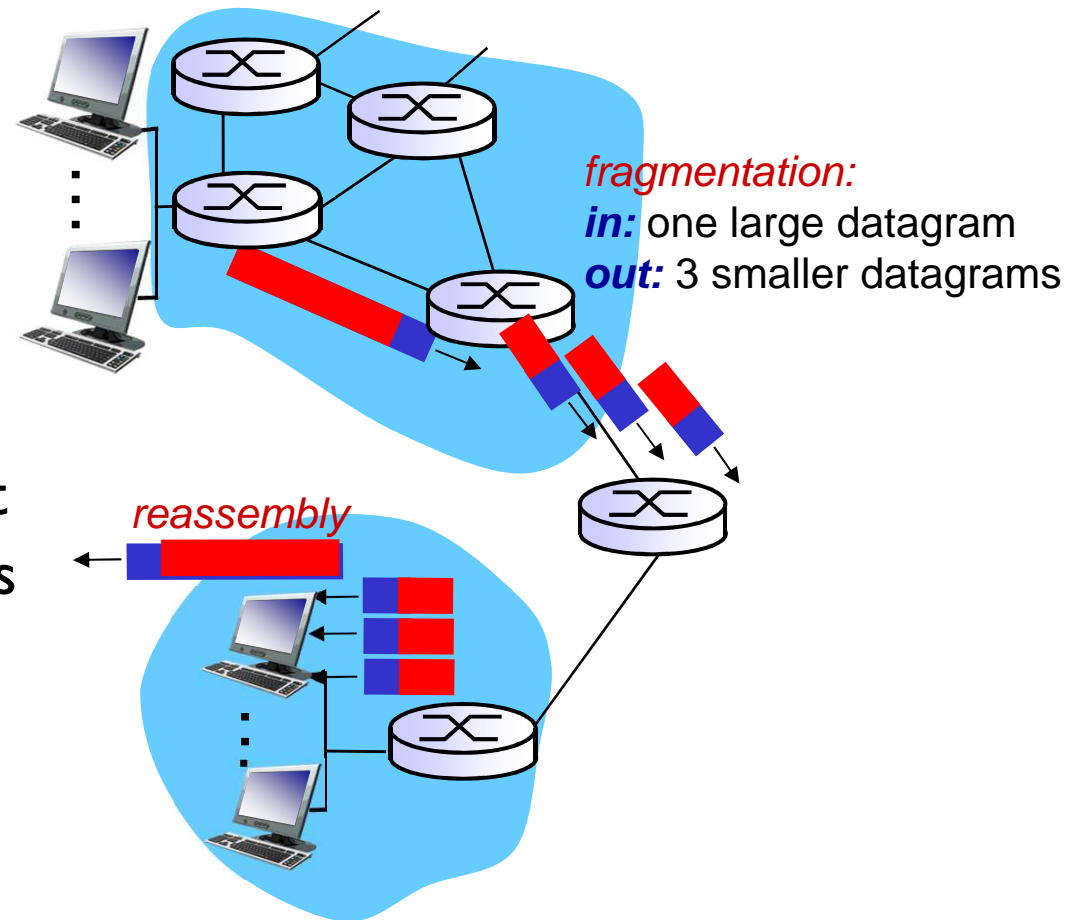
# IP fragmentation, reassembly

❖ 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 final destination
  - IP header bits used to identify, order related fragments

*fragmentation:*
***in:*** one large datagram
***out:*** 3 smaller datagrams

*reassembly*

# IP fragmentation, reassembly

*example:*

- 4000 byte datagram
- MTU = 1500 bytes

| | length =4000 | ID =x | fragflag =0 | offset =0 | |

*one large datagram becomes several smaller datagrams*

1480 bytes in data field

offset = 1480/8

| | length =1500 | ID =x | fragflag =1 | offset =0 | |

| | length =1500 | ID =x | fragflag =1 | offset =185 | |

| | length =1040 | ID =x | fragflag =0 | offset =370 | |

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
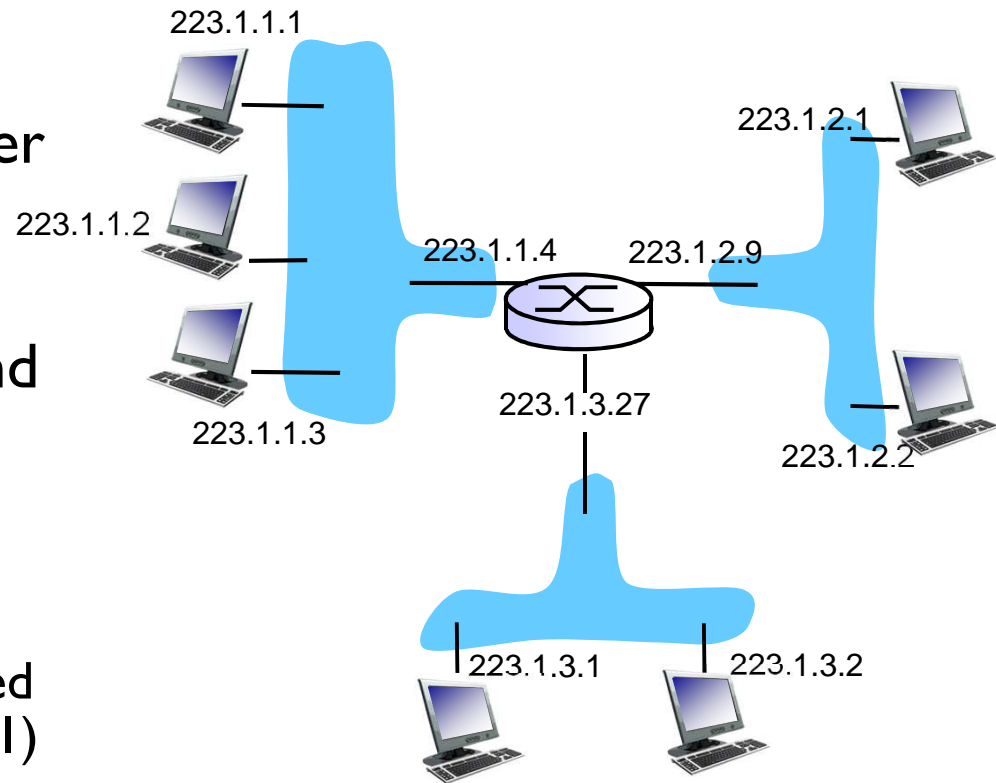- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# IP addressing: introduction

❖ *IP address:* 32-bit identifier for host, 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)

❖ *IP addresses associated with each interface*

223.1.1.1

223.1.1.2

223.1.2.1

223.1.1.4    223.1.2.9

223.1.1.3

223.1.3.27

223.1.2.2

223.1.3.1    223.1.3.2

223.1.1.1 = 11011111 00000001 00000001 00000001

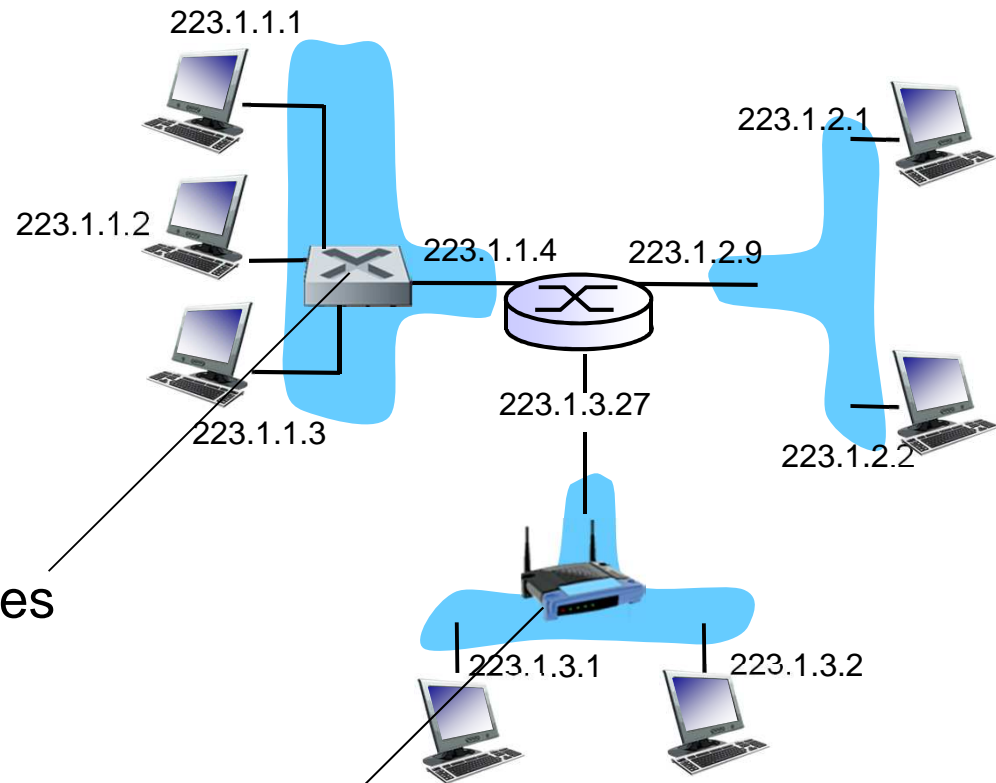          223          1          1          1

# IP addressing: introduction

*Q: how are interfaces actually connected?*

*A: we'll learn about that in chapter 5, 6.*

*A:* wired Ethernet interfaces connected by Ethernet switches

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

223.1.1.1

223.1.1.2

223.1.1.4    223.1.2.9

223.1.1.3

223.1.2.1

223.1.3.27

223.1.2.2

223.1.3.1    223.1.3.2

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

# Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same subnet part of IP address
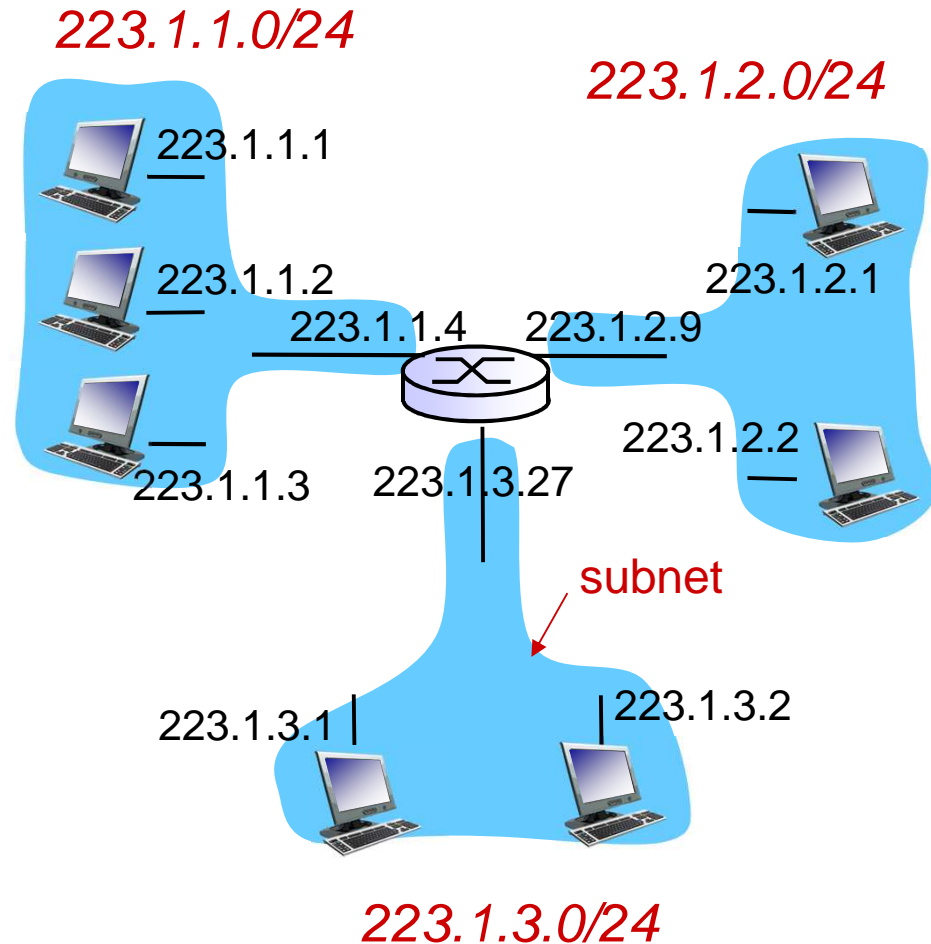  - can physically reach each other *without intervening router*

223.1.1.1

223.1.1.2

223.1.1.4    223.1.2.9

223.1.2.1

223.1.2.2

223.1.1.3    223.1.3.27

subnet

223.1.3.1    223.1.3.2
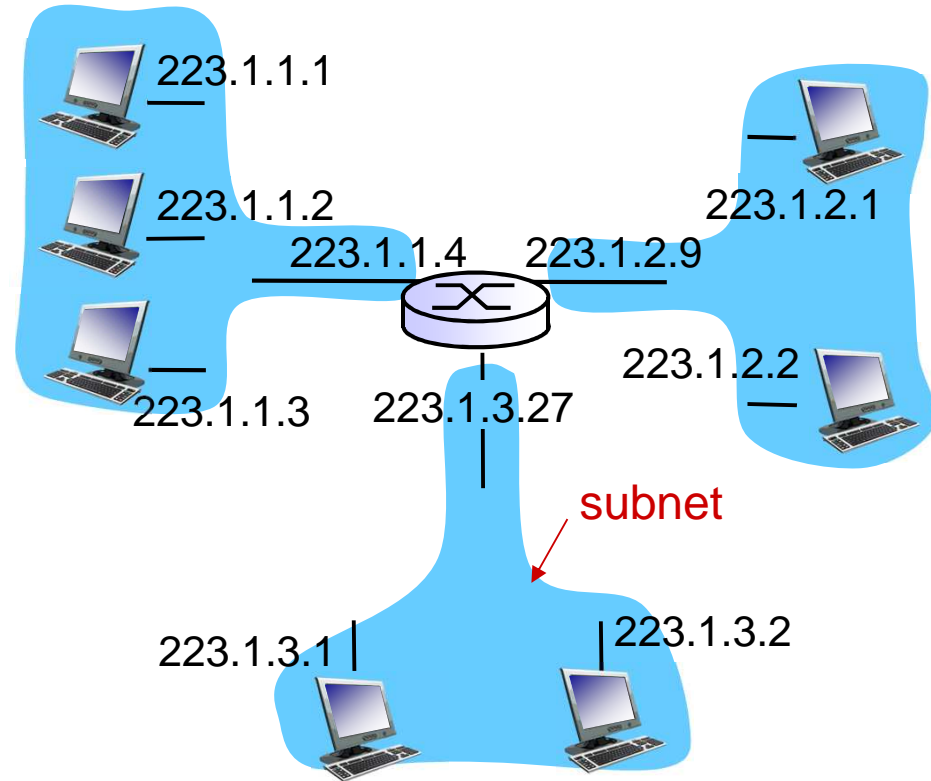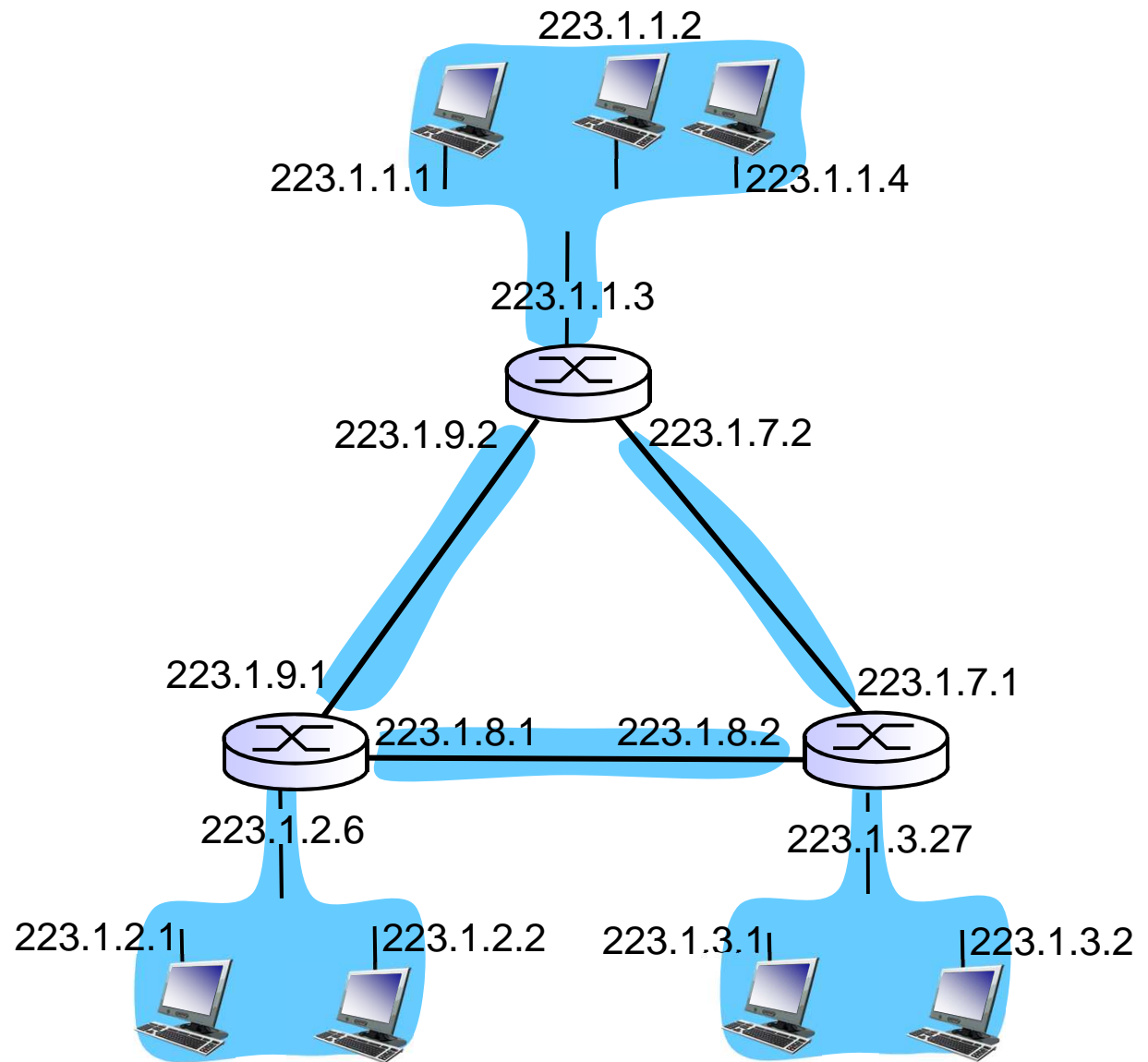
network consisting of 3 subnets

# Subnets

*recipe*

❖ to determine the subnets, detach each interface from its host or router, creating islands of isolated networks

❖ each isolated network is called a *subnet*



223.1.1.0/24

223.1.2.0/24

223.1.1.1

223.1.1.2

223.1.1.4    223.1.2.9

223.1.2.1

223.1.2.2

223.1.1.3    223.1.3.27

subnet

223.1.3.1    223.1.3.2

223.1.3.0/24

subnet mask: /24

# Subnets

how many?



223.1.1.2

223.1.1.1   223.1.1.4

223.1.1.3

223.1.9.2   223.1.7.2

223.1.9.1   223.1.7.1

223.1.8.1   223.1.8.2

223.1.2.6   223.1.3.27

223.1.2.1   223.1.2.2   223.1.3.1   223.1.3.2

# IP addressing: CIDR

**CIDR:** Classless InterDomain Routing

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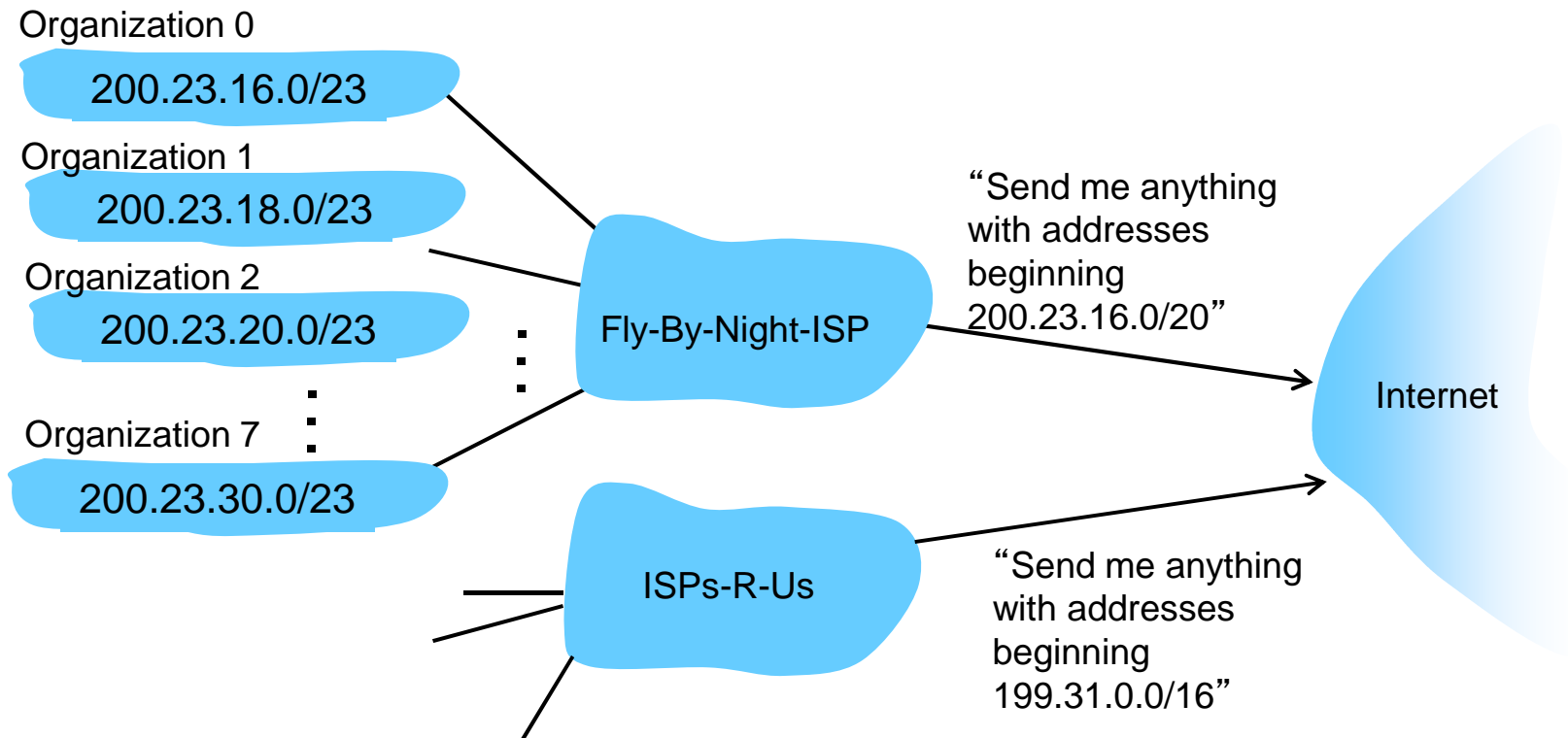- address format: a.b.c.d/x, where x is # bits in subnet portion of address

subnet part ◄─────────────────────────► ◄──► host part

11001000 00010111 00010000 00000000
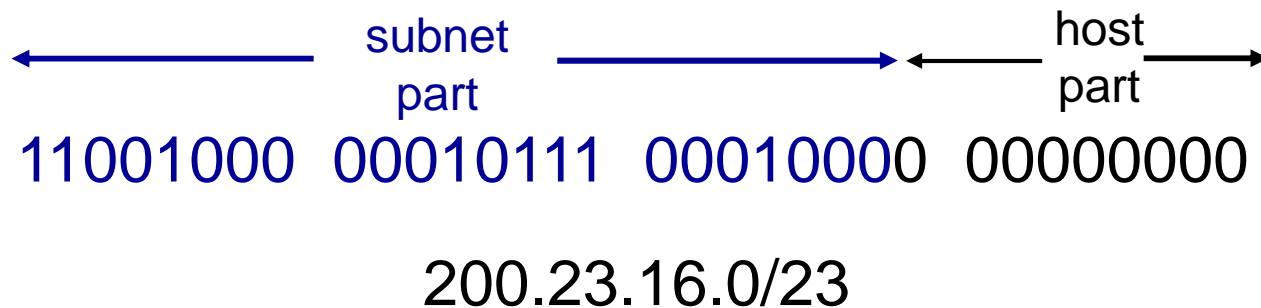
200.23.16.0/23

# IP addresses: how to get one?

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

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

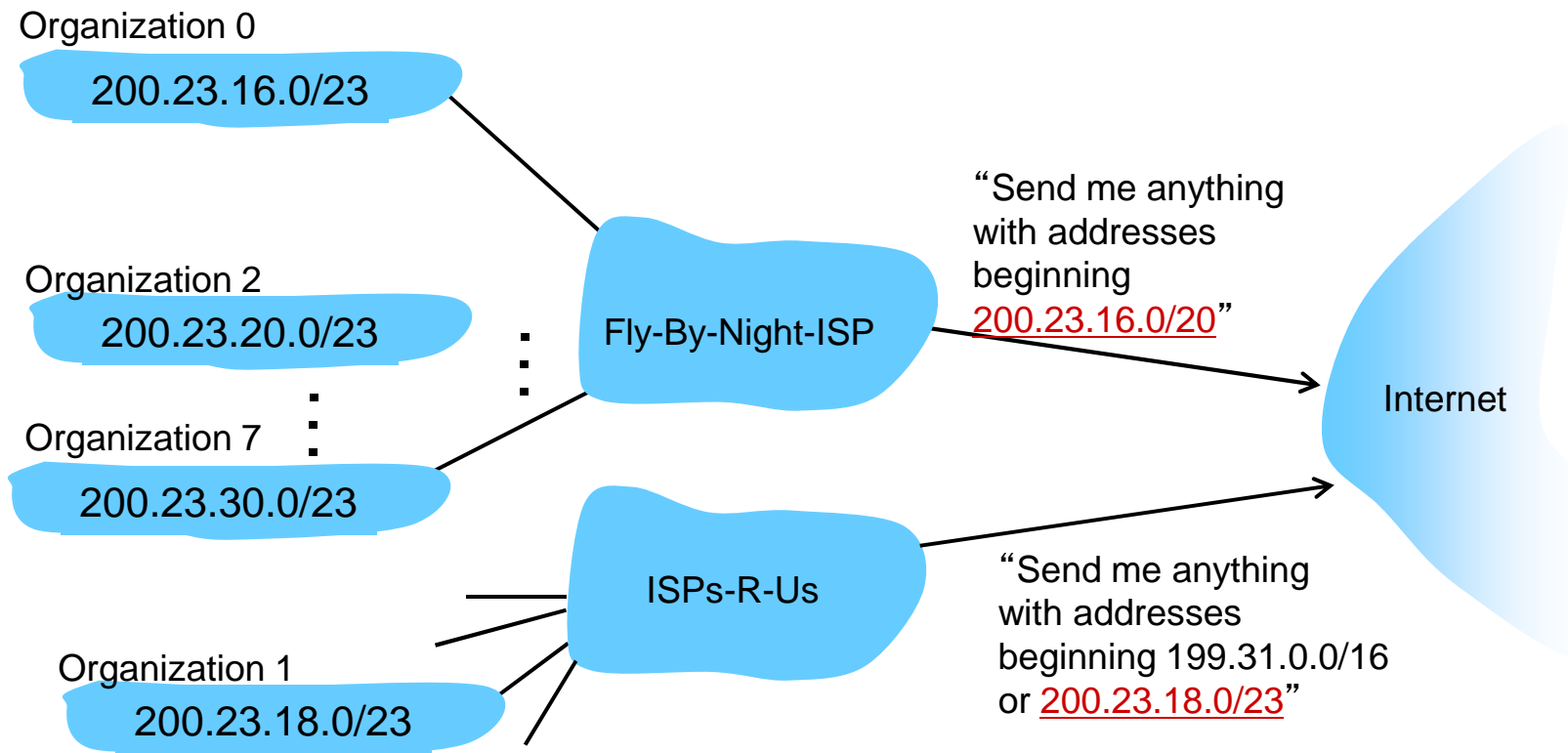| | | | |
|---|---|---|---|
| ISP's block | 11001000  00010111  00010000 | 00000000 | 200.23.16.0/20 |
| | | | |
| Organization 0 | 11001000  00010111  00010000 | 00000000 | 200.23.16.0/23 |
| Organization 1 | 11001000  00010111  00010010 | 00000000 | 200.23.18.0/23 |
| Organization 2 | 11001000  00010111  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:

Organization 0

200.23.16.0/23

Organization 1

200.23.18.0/23

Organization 2

200.23.20.0/23

Organization 7

200.23.30.0/23

Fly-By-Night-ISP

"Send me anything with addresses beginning 200.23.16.0/20"

Internet

ISPs-R-Us

"Send me anything with addresses beginning 199.31.0.0/16"

# Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1

Organization 0
200.23.16.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

Fly-By-Night-ISP

"Send me anything
with addresses
beginning
200.23.16.0/20"

Internet

ISPs-R-Us

Organization 1
200.23.18.0/23

"Send me anything
with addresses
beginning 199.31.0.0/16
or 200.23.18.0/23"

# IP addressing: administration

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

*A:* ICANN: Internet Corporation for Assigned
   Names and Numbers http://www.icann.org/
   - allocates addresses
   - manages DNS
   - assigns domain names, resolves disputes

*Special Addresses*

- broadcast: all host bits set to 1
- private networks: 192.168/16, 172.16/12, 10/8

# IP addresses: how to get one?

Q: How does a *host* get IP address?

❖ hard-coded by system admin in a file
- Windows: control-panel->network->configuration->tcp/ip->properties
- UNIX: /etc/rc.config

❖ DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
- "plug-and-play"

# DHCP: Dynamic Host Configuration Protocol

*goal:* allow host to *dynamically* obtain its 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 want to join network (more shortly)

*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

**223.1.1.0/24**

223.1.1.1

223.1.1.2

223.1.1.3

223.1.1.4

*DHCP server*

223.1.2.1

223.1.2.9

223.1.2.2

*arriving DHCP client needs address in this network*

**223.1.2.0/24**

223.1.3.27

223.1.3.1

223.1.3.2

**223.1.3.0/24**

# DHCP client-server scenario

DHCP server: 223.1.2.5

**DHCP discover**

arriving client

> src : 0.0.0.0, 68
> dest.: 255.255.255.255,67
> yiaddr:   0.0.0.0
> transaction ID: 654

**DHCP offer**

> src: 223.1.2.5, 67
> dest:  255.255.255.255, 68
> yiaddrr: 223.1.2.4
> transaction ID: 654
> lifetime: 3600 secs

**DHCP request**

> src:  0.0.0.0, 68
> dest::  255.255.255.255, 67
> yiaddrr: 223.1.2.4
> transaction ID: 654
> lifetime: 3600 secs

**DHCP ACK**

> src: 223.1.2.5, 67
> dest:  255.255.255.255, 68
> yiaddrr: 223.1.2.4
> transaction ID: 654
> lifetime: 3600 secs

# 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

DHCP
UDP
IP
Eth
Phy

DHCP
UDP
IP
Eth
Phy

168.1.1.1

*router with DHCP server built into router*
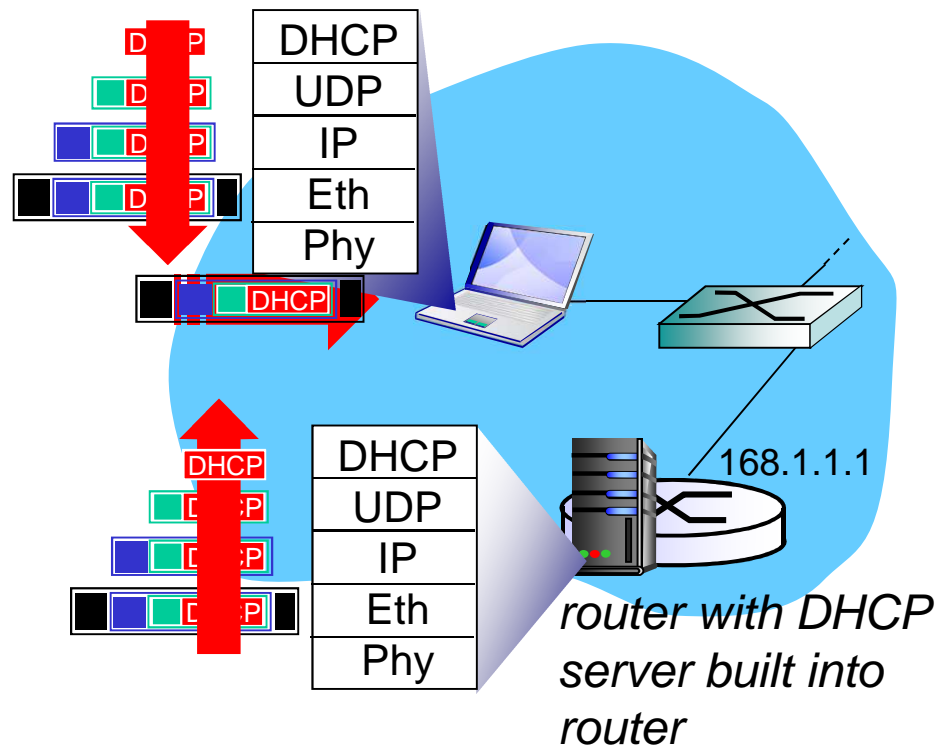
- ❖ connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP

- ❖ DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet

- ❖ Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server

- ❖ Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

# DHCP: example



router with DHCP server built into router
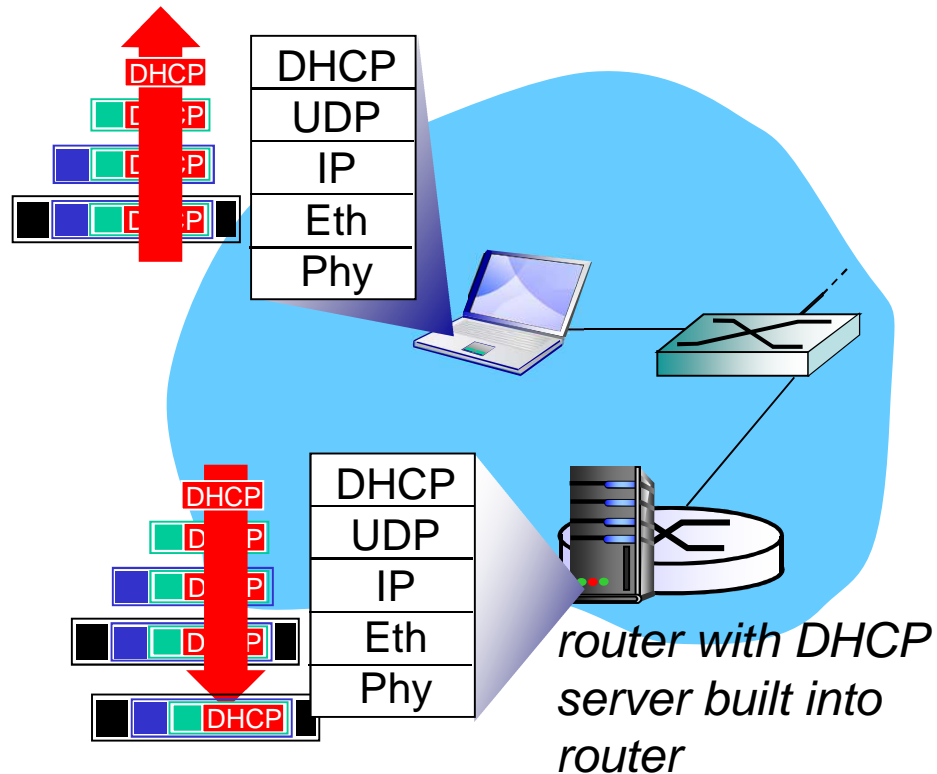
- ❖ DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server

- ❖ encapsulation of DHCP server, frame forwarded to client, demuxing 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

# NAT: network address translation

rest of Internet ← → | ← local network (e.g., home network) 10.0.0/24 →

10.0.0.4

138.76.29.7

10.0.0.1

10.0.0.2

10.0.0.3
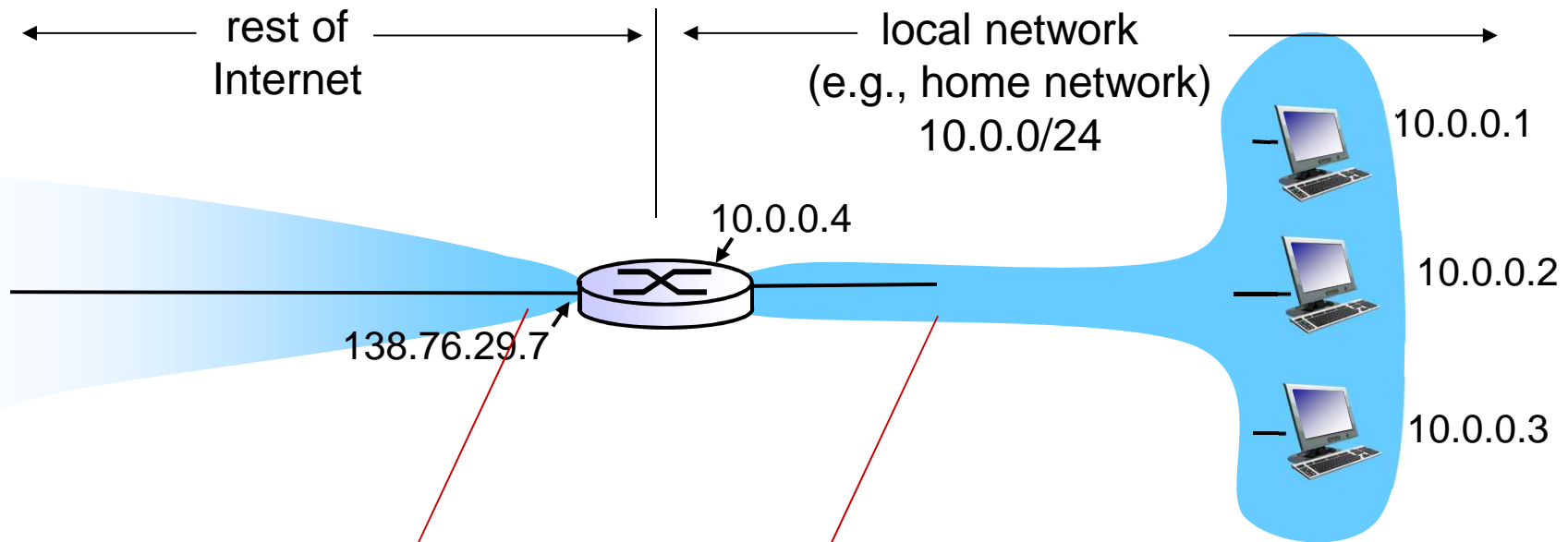
*all* datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7,different source port numbers

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

# NAT: network address translation

*motivation:* local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices

- can change addresses of devices in local network without notifying outside world

- can change ISP without changing addresses of devices in local network

- devices inside local net not explicitly addressable, visible by outside world (a security plus)

# NAT: network address translation

*implementation:* NAT router must:

- *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 addr

- *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 dest 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 addr 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

S: 10.0.0.1, 3345
D: 128.119.40.186, 80

1

10.0.0.1

S: 138.76.29.7, 5001
D: 128.119.40.186, 80

2

10.0.0.4

10.0.0.2

138.76.29.7

S: 128.119.40.186, 80
D: 10.0.0.1, 3345

4

S: 128.119.40.186, 80
D: 138.76.29.7, 5001

3

10.0.0.3

**3:** reply arrives dest. address: 138.76.29.7, 5001

**4:** NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

# NAT: network address translation

❖ 16-bit port-number field:
  ▪ 60,000 simultaneous connections with a single LAN-side address!

❖ NAT is controversial:
  ▪ routers should only process up to layer 3
  ▪ violates end-to-end argument
    • NAT possibility must be taken into account by app designers, e.g., P2P applications
  ▪ address shortage should instead be solved by IPv6

# NAT traversal problem

❖ **client wants to connect to server with address 10.0.0.1**

  ▪ server address 10.0.0.1 local to LAN (client can't use it as destination addr)

  ▪ only one externally visible NATed address: 138.76.29.7

❖ *solution1:* statically configure NAT to forward incoming connection requests at given port to server

  ▪ e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000

client

?

138.76.29.7    NAT
               router

10.0.0.1

10.0.0.4
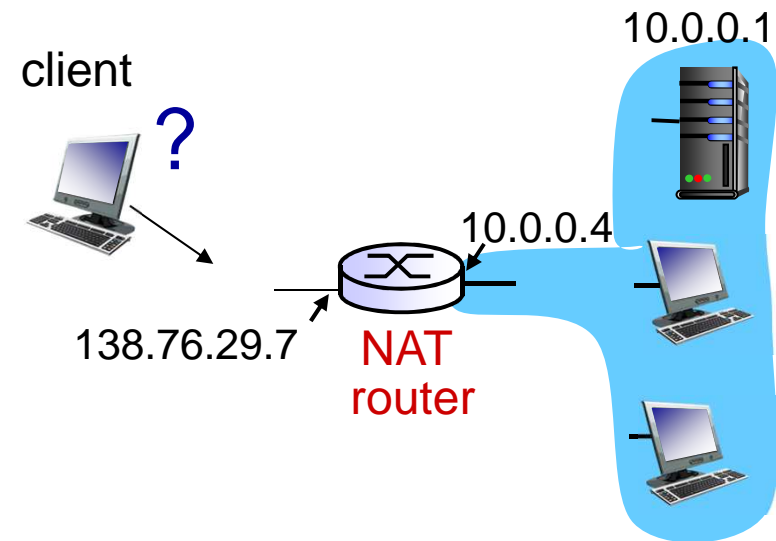
# NAT traversal problem

❖ *solution 2:* Universal Plug and Play
(UPnP) Internet Gateway Device
(IGD) Protocol.  Allows NATed
host to:
  ❖ learn public IP address
    (138.76.29.7)
  ❖ add/remove port mappings
    (with lease times)

i.e., automate static NAT port
    map configuration

10.0.0.1

IGD

NAT
router
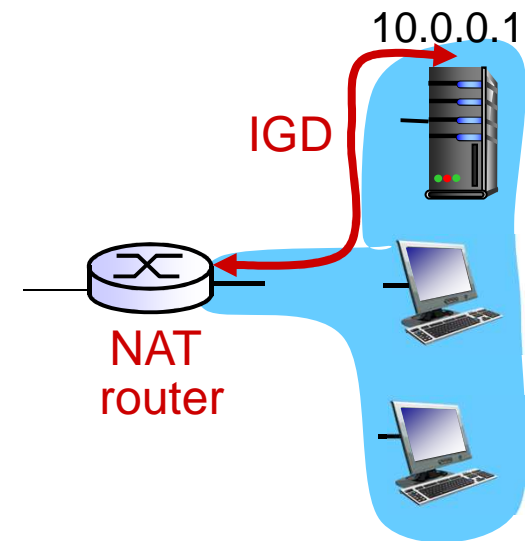
# NAT traversal problem

- *solution 3:* relaying (used in Skype)
  - NATed client establishes connection to relay
  - external client connects to relay
  - relay bridges packets between to connections
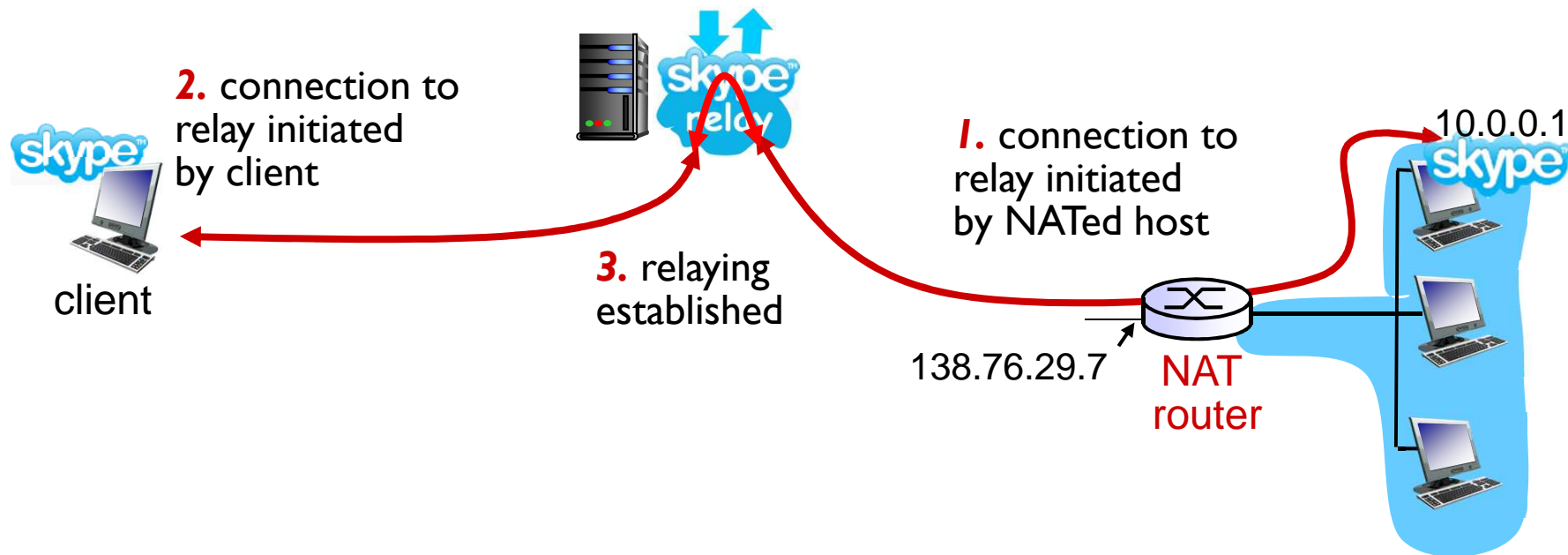
**2.** connection to relay initiated by client

**1.** connection to relay initiated by NATed host

**3.** relaying established

client

10.0.0.1

138.76.29.7

NAT router

# Chapter 4: outline

# 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 plus first 8 bytes of IP datagram causing error

| Type | Code | description |
|------|------|-------------|
| 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 |
| 4 | 0 | source quench (congestion control - not used) |
| 8 | 0 | echo request (ping) |
| 9 | 0 | route advertisement |
| 10 | 0 | router discovery |
| 11 | 0 | TTL expired |
| 12 | 0 | bad IP header |

# Traceroute and ICMP

❖ source sends series of UDP segments to dest
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number

❖ when *n*th set of datagrams arrives to nth router:
  - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes name of router & IP address

❖ when ICMP messages arrives, source records RTTs

*stopping criteria:*

❖ UDP segment eventually arrives at destination host

❖ destination returns ICMP "port unreachable" message (type 3, code 3)

❖ source stops

3 probes    3 probes

3 probes
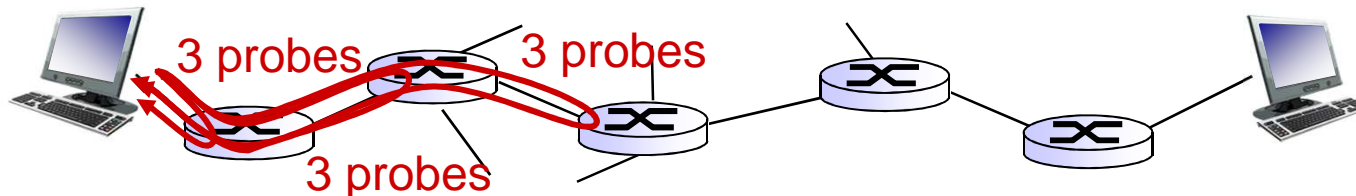
# IPv6: motivation

❖ *initial motivation:* 32-bit address space soon to be completely allocated.
❖ additional motivation:
  ▪ header format helps speed processing/forwarding
  ▪ header changes to facilitate QoS

*IPv6 datagram format:*
  ▪ fixed-length 40 byte header
  ▪ no fragmentation allowed

# IPv6 datagram format

*priority:* identify priority among datagrams in flow
*flow label:* identify datagrams in same "flow."
        (concept of "flow" not well defined).
*next header:* identify upper layer protocol for data

| ver | pri | flow label | | |
|---|---|---|---|---|
| payload len | | | next hdr | hop limit |
| source address (128 bits) | | | | |
| destination address (128 bits) | | | | |
| data | | | | |

⟵——————— 32 bits ———————⟶
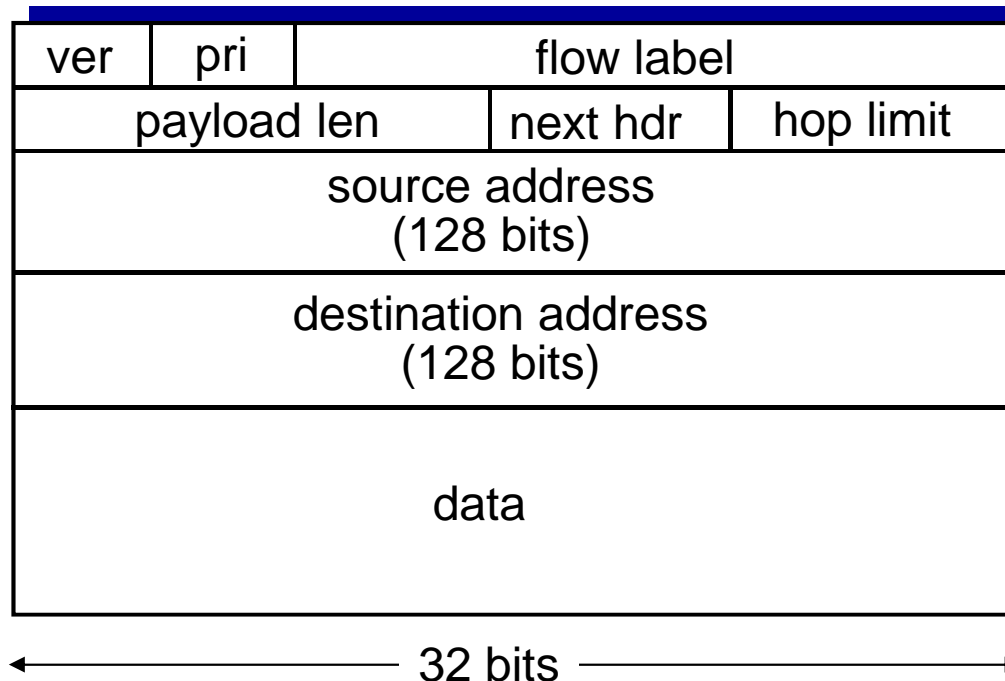
# Other changes from IPv4

❖ *checksum:* removed entirely to reduce processing time at each hop

❖ *options:* allowed, but outside of header, indicated by "Next Header" field

❖ *ICMPv6:* new version of ICMP
  ▪ additional message types, e.g. "Packet Too Big"
  ▪ multicast group management functions

# 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
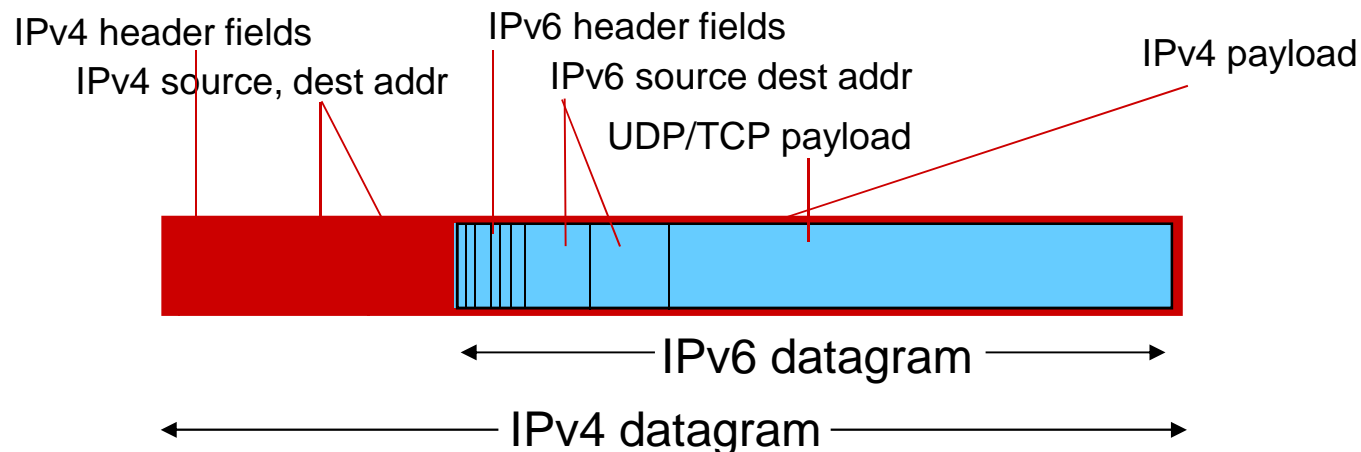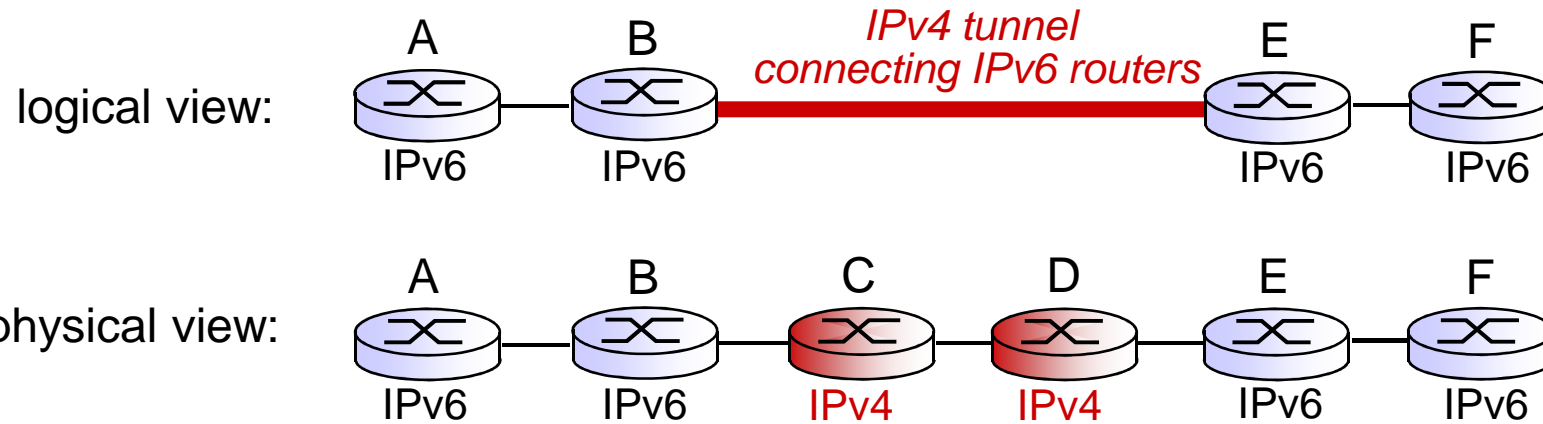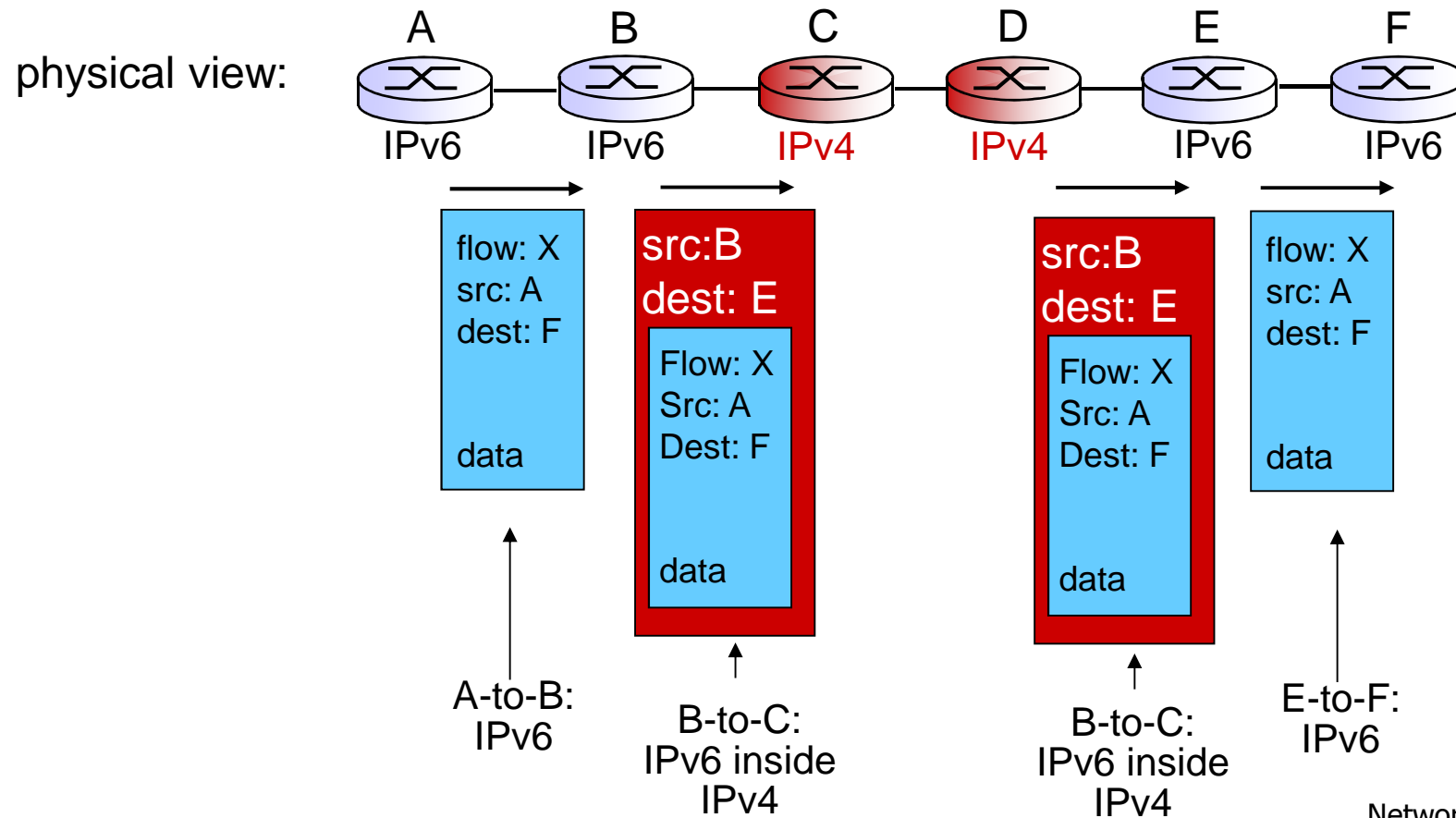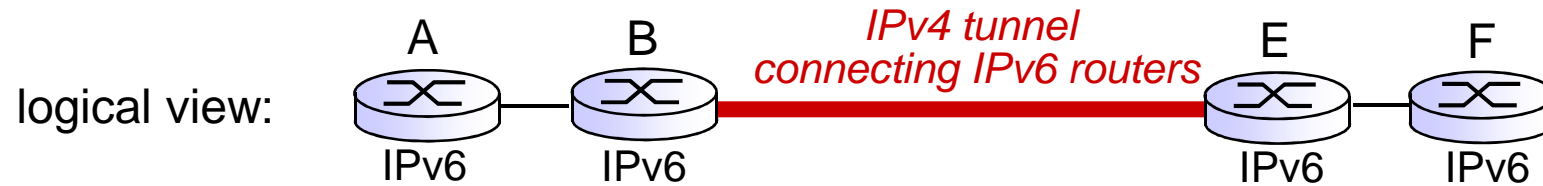
IPv4 header fields

IPv4 source, dest addr

IPv6 header fields

IPv6 source dest addr

UDP/TCP payload

IPv4 payload

IPv6 datagram

IPv4 datagram

# Tunneling

logical view:

A       B       *IPv4 tunnel*     E       F
*connecting IPv6 routers*

IPv6     IPv6             IPv6     IPv6

physical view:

A     B     C     D     E     F

IPv6    IPv6    IPv4    IPv4    IPv6    IPv6

# Tunneling

logical view:

A — B ———— IPv4 tunnel connecting IPv6 routers ———— E — F

IPv6   IPv6                                          IPv6   IPv6

physical view:

A — B — C — D — E — F

IPv6   IPv6   IPv4   IPv4   IPv6   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

flow: X
src: A
dest: F

data

A-to-B:
IPv6

B-to-C:
IPv6 inside
IPv4

B-to-C:
IPv6 inside
IPv4

E-to-F:
IPv6

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

# Interplay between routing, forwarding

routing algorithm

local forwarding table

| dest address | output link |
|---|---|
| address-range 1 | 3 |
| address-range 2 | 2 |
| address-range 3 | 2 |
| address-range 4 | 1 |

routing algorithm determines end-end-path through network

forwarding table determines local forwarding at this router

IP destination address in arriving packet's header

1

3 2
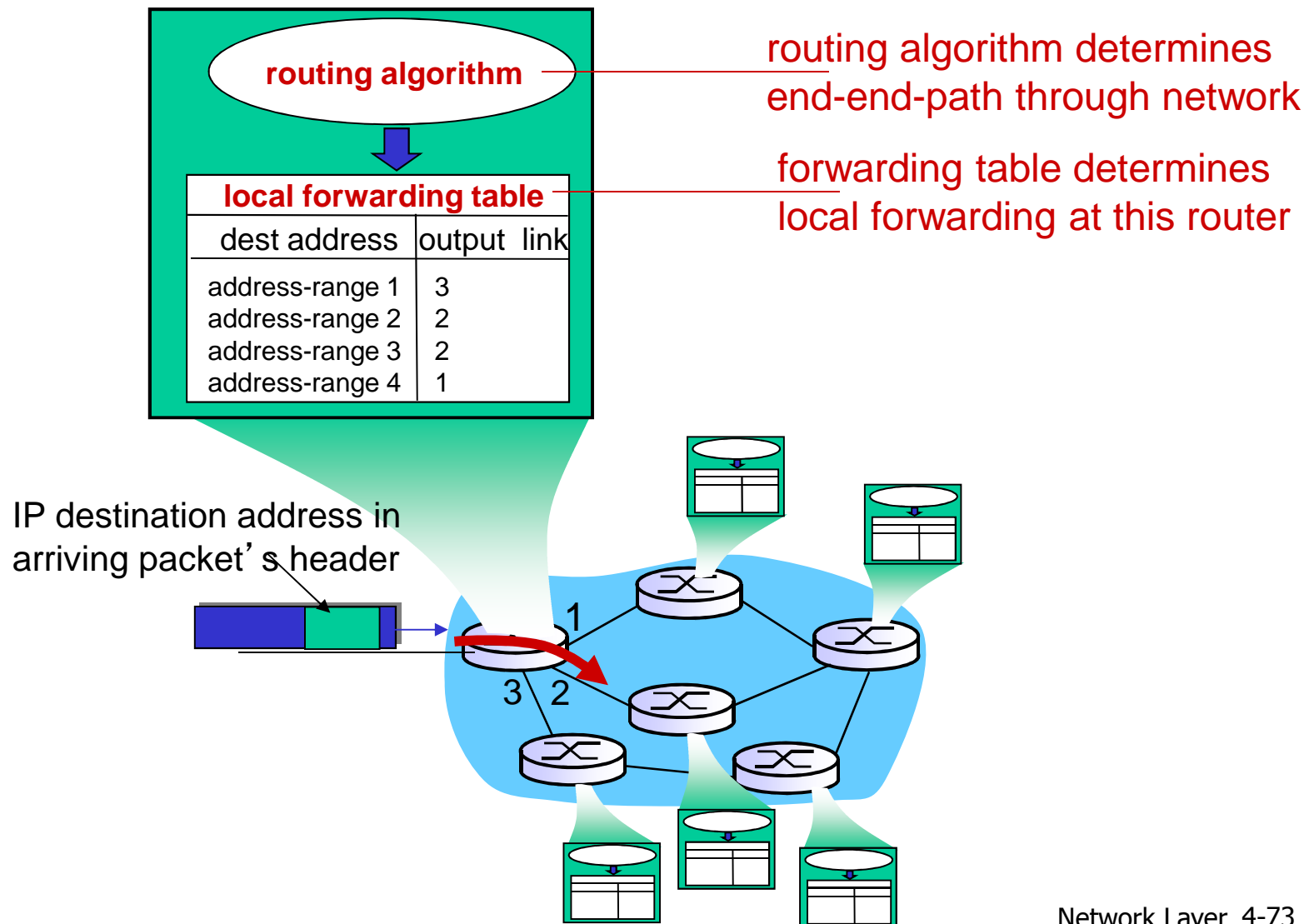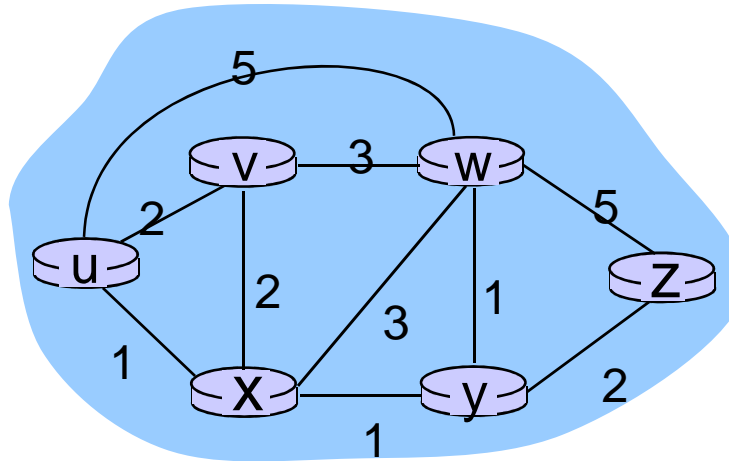
# Graph abstraction



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), (x,w), (x,y), (w,y), (w,z), (y,z) }

*aside:* graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

# Graph abstraction: costs
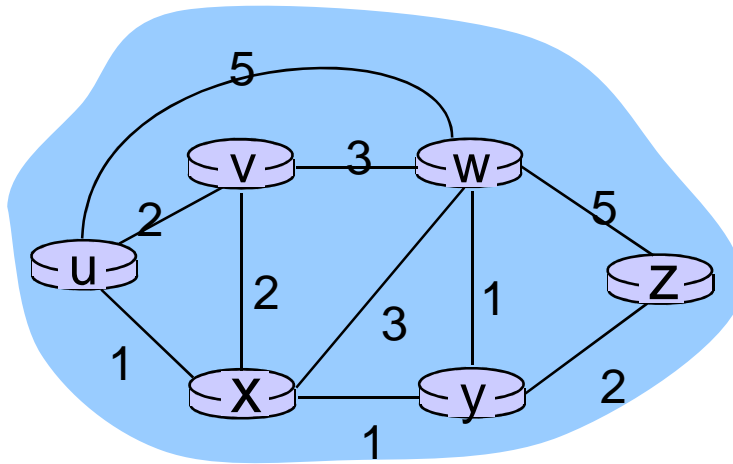


$c(x,x') = $ cost of link $(x,x')$
  e.g., $c(w,z) = 5$

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

cost of path $(x_1, x_2, x_3,…, x_p) = c(x_1,x_2) + c(x_2,x_3) + … + c(x_{p-1},x_p)$

*key question:* what is the least-cost path between u and z ?
*routing algorithm:* algorithm that finds that least cost path

# Routing algorithm classification

## Q: static or dynamic?

*static:*

❖ routes change slowly over time

*dynamic:*

❖ routes change more quickly
- periodic update
- in response to link cost changes

## Q: global or decentralized information?

*global:*

❖ all routers have complete topology, link cost info
❖ "link state" algorithms

*decentralized:*

❖ router knows physically-connected neighbors, link costs to neighbors
❖ iterative process of computation, exchange of info with neighbors
❖ "distance vector" algorithms

# Chapter 4: outline

# A Link-State Routing Algorithm

## *Dijkstra's algorithm*

- ❖ net topology, link costs known to all nodes
  - ▪ accomplished via "link state broadcast"
  - ▪ all nodes have same info
- ❖ computes least cost paths from one node ('source") to all other nodes
  - ▪ gives *forwarding table* for that node
- ❖ iterative: after k iterations, know least cost path to k dest.'s

## *notation:*

- ❖ c(x,y): link cost from node x to y;  = ∞ if not direct neighbors
- ❖ D(v): current value of cost of path from source to dest. v
- ❖ p(v): predecessor node along path from source to v
- ❖ N': set of nodes whose least cost path definitively known

# Dijsktra's Algorithm

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5        then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12     D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15 until all nodes in N'
```

# Dijkstra's algorithm: example

| | | D(**v**) | D(**w**) | D(**x**) | D(**y**) | D(**z**) |
|---|---|---|---|---|---|---|
| Step | N' | p(v) | p(w) | p(x) | p(y) | p(z) |
| 0 | u | 7,u | 3,u | 5,u | ∞ | ∞ |
| 1 | uw | 6,w | | 5,u | 11,w | ∞ |
| 2 | uwx | 6,w | | | 11,w | 14,x |
| 3 | uwxv | | | | 10,v | 14,x |
| 4 | uwxvy | | | | | 12,y |
| 5 | uwxvyz | | | | | |

## *notes:*

- ❖ construct shortest path tree by tracing predecessor nodes
- ❖ ties can exist (can be broken arbitrarily)
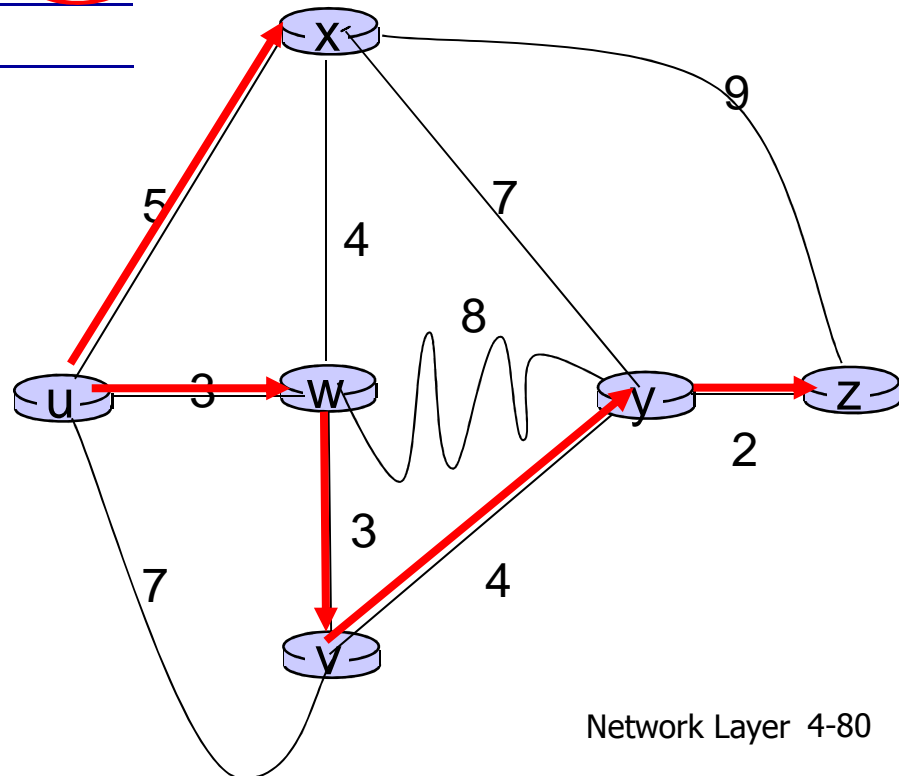
# Dijkstra's algorithm: another example

| Step | N' | D(v),p(v) | D(w),p(w) | D(x),p(x) | D(y),p(y) | D(z),p(z) |
|------|-------|-----------|-----------|-----------|-----------|-----------|
| 0 | u | 2,u | 5,u | 1,u | ∞ | ∞ |
| 1 | ux | 2,u | 4,x | | 2,x | ∞ |
| 2 | uxy | 2,u | 3,y | | | 4,y |
| 3 | uxyv | | 3,y | | | 4,y |
| 4 | uxyvw | | | | | 4,y |
| 5 | uxyvwz | | | | | |
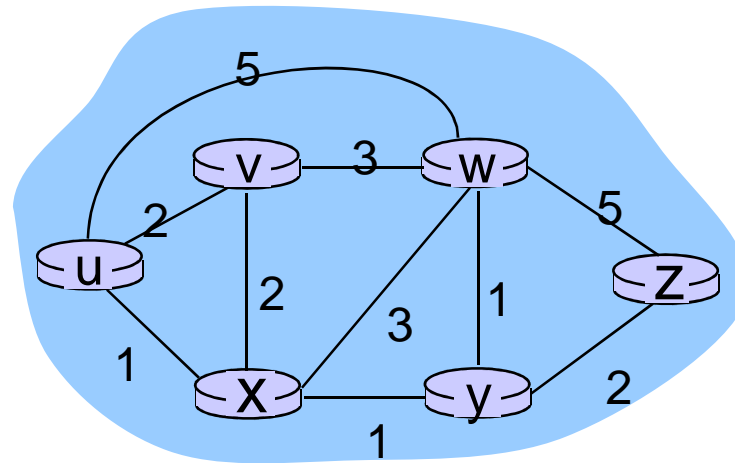
# Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:
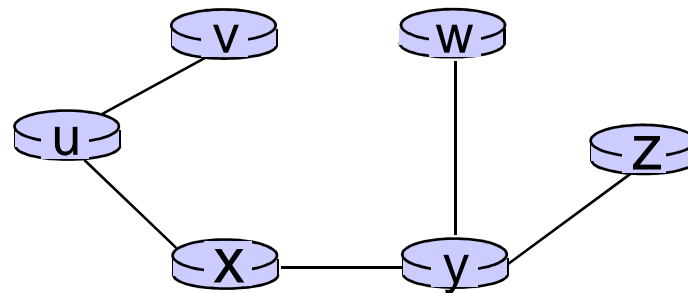


resulting forwarding table in u:

| destination | link |
|:-----------:|:-----:|
| v | (u,v) |
| x | (u,x) |
| y | (u,x) |
| w | (u,x) |
| z | (u,x) |

# Dijkstra's algorithm, discussion

*algorithm complexity:* n nodes

- ❖ each iteration: need to check all nodes, w, not in N
- ❖ $n(n+1)/2$ comparisons: $O(n^2)$
- ❖ more efficient implementations possible: $O(n\log n)$

*oscillations possible:*

- ❖ e.g., support link cost equals amount of carried traffic:
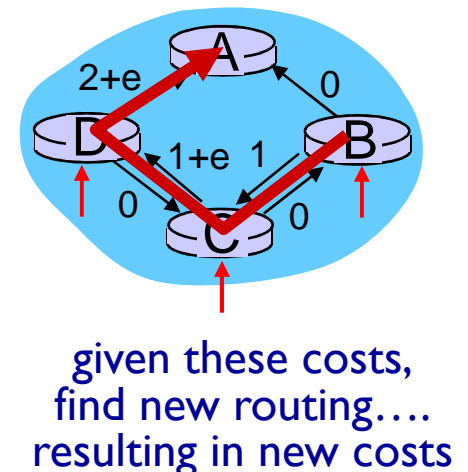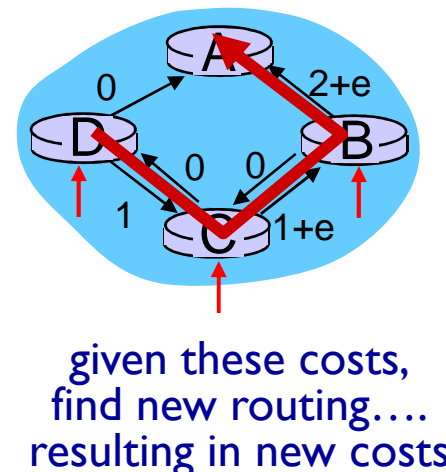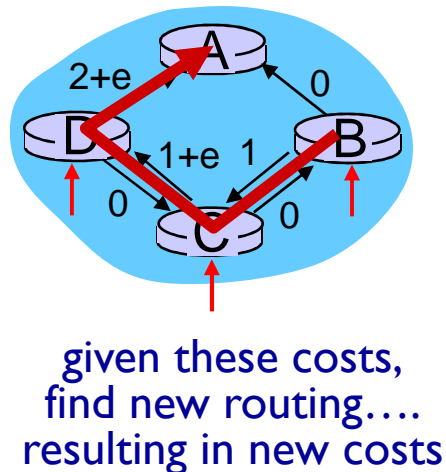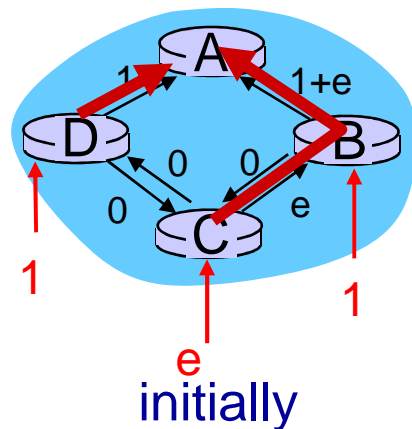


initially

given these costs,
find new routing….
resulting in new costs

given these costs,
find new routing….
resulting in new costs

given these costs,
find new routing….
resulting in new costs

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# Distance vector algorithm

*Bellman equation (dynamic programming)*

let

   $d_x(y) :=$ cost of least-cost path from x to y

then

$$d_x(y) = \min_{v} \{c(x,v) + d_v(y) \}$$

cost from neighbor v to destination y

cost to neighbor v

*min* taken over all neighbors v of x

# Example
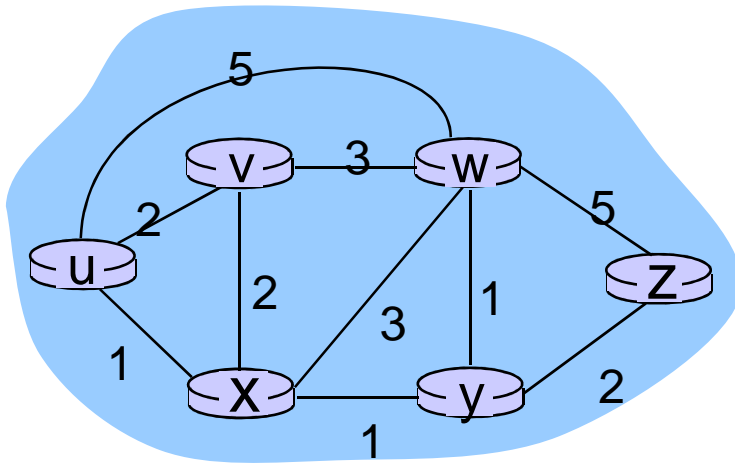


clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

Bellman equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z),$$
$$c(u,x) + d_x(z),$$
$$c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5,$$
$$1 + 3,$$
$$5 + 3\} = 4$$

node achieving minimum is next hop in shortest path, used in forwarding table

# Distance vector algorithm

❖ Bellman-Ford algorithm

❖ $D_x(y)$ = estimate of least cost from x to y
  ▪ x maintains distance vector $\mathbf{D_x}$ = [$D_x(y)$: y ∈ N ]

❖ node x:
  ▪ knows cost to each neighbor v: $c(x,v)$
  ▪ maintains its neighbors' distance vectors. For each neighbor v, x maintains
    $\mathbf{D_v}$ = [$D_v(y)$: y ∈ N ]

# Distance vector algorithm

*key idea:*

- ❖ from time-to-time, each node sends its own distance vector estimate to neighbors

- ❖ when x receives new DV estimate from neighbor, it updates its own DV using Bellman equation:

$$D_x(y) \leftarrow min_v\{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- ❖ under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

# Distance vector algorithm

*iterative, asynchronous:*
each local iteration
caused by:

❖ local link cost change

❖ DV update message from
neighbor

*distributed:*

❖ each node notifies *only*
neighbors when its DV
changes

  ▪ neighbors then notify their
    neighbors if necessary

*each node:*
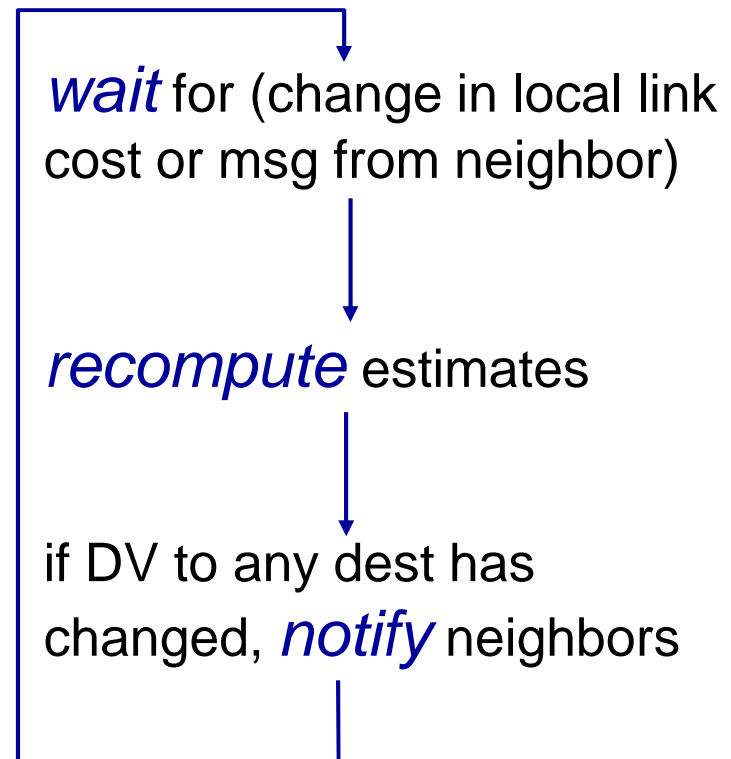
*wait* for (change in local link
cost or msg from neighbor)

*recompute* estimates

if DV to any dest has
changed, *notify* neighbors

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$
$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$
$$= \min\{2+1, 7+0\} = 3$$

**node x**
**table**

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

**node y**
**table**

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

**node z**
**table**

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | ∞ | ∞ | ∞ |
| z | 7 | 1 | 0 |

time

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0, 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1, 7+0\} = 3$

**node x table**

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

**node y table**

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

**node z table**

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | ∞ | ∞ | ∞ |
| z | 7 | 1 | 0 |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

*cost to*

| from | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

time

# Distance vector: link cost changes

*link cost changes:*

- ❖ node detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors
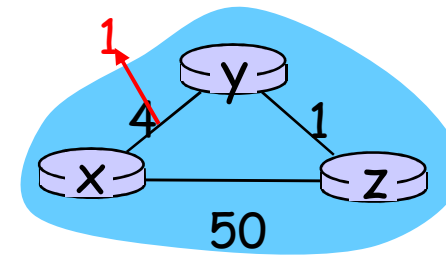


"good news travels fast"

$t_0$ : $y$ detects link-cost change, updates its DV, informs its neighbors.

$t_1$ : $z$ receives update from $y$, updates its table, computes new least cost to $x$, sends its neighbors its DV.

$t_2$ : $y$ receives $z$'s update, updates its distance table. $y$'s least costs do *not* change, so $y$ does *not* send a message to $z$.

# Distance vector: link cost changes

*link cost changes:*

❖ node detects local link cost change

❖ *bad news travels slow* - "count to infinity" problem!

❖ 44 iterations before algorithm stabilizes: see text
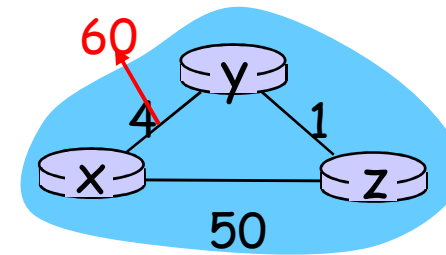
*poisoned reverse:*

❖ If Z routes through Y to get to X :

  ▪ Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)

❖ will this completely solve count to infinity problem?

# Comparison of LS and DV algorithms

## message complexity

❖ **LS:** with n nodes, E links, O(nE) msgs sent

❖ **DV:** exchange between neighbors only
  - convergence time varies

## speed of convergence

❖ **LS:** $O(n^2)$ algorithm requires O(nE) msgs
  - may have oscillations

❖ **DV:** convergence time varies
  - may be routing loops
  - count-to-infinity problem
  - oscillations even worse

## robustness: what happens if router malfunctions?

### LS:
  - node can advertise incorrect *link* cost
  - each node computes only its *own* table

### DV:
  - DV node can advertise incorrect *path* cost
  - each node's table used by others
    - error propagates through network

# Chapter 4: outline

# Hierarchical routing

our routing study thus far - idealization
- ❖ all routers identical
- ❖ network "flat"
- … *not* true in practice

*scale:* ~ 1 billion hosts:
- ❖ can't store all dest's in routing tables!
- ❖ routing table exchange would swamp links!

*administrative autonomy*
- ❖ internet = network of networks
- ❖ each network admin may want to control routing in its own network

# Hierarchical routing

❖ **aggregate routers into regions,** "autonomous systems" (AS)

❖ **routers in same AS run same routing protocol**
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol

*gateway router:*

❖ at "edge" of its own AS

❖ has link to router in another AS

# Interconnected ASes



Intra-AS Routing algorithm

Inter-AS Routing algorithm
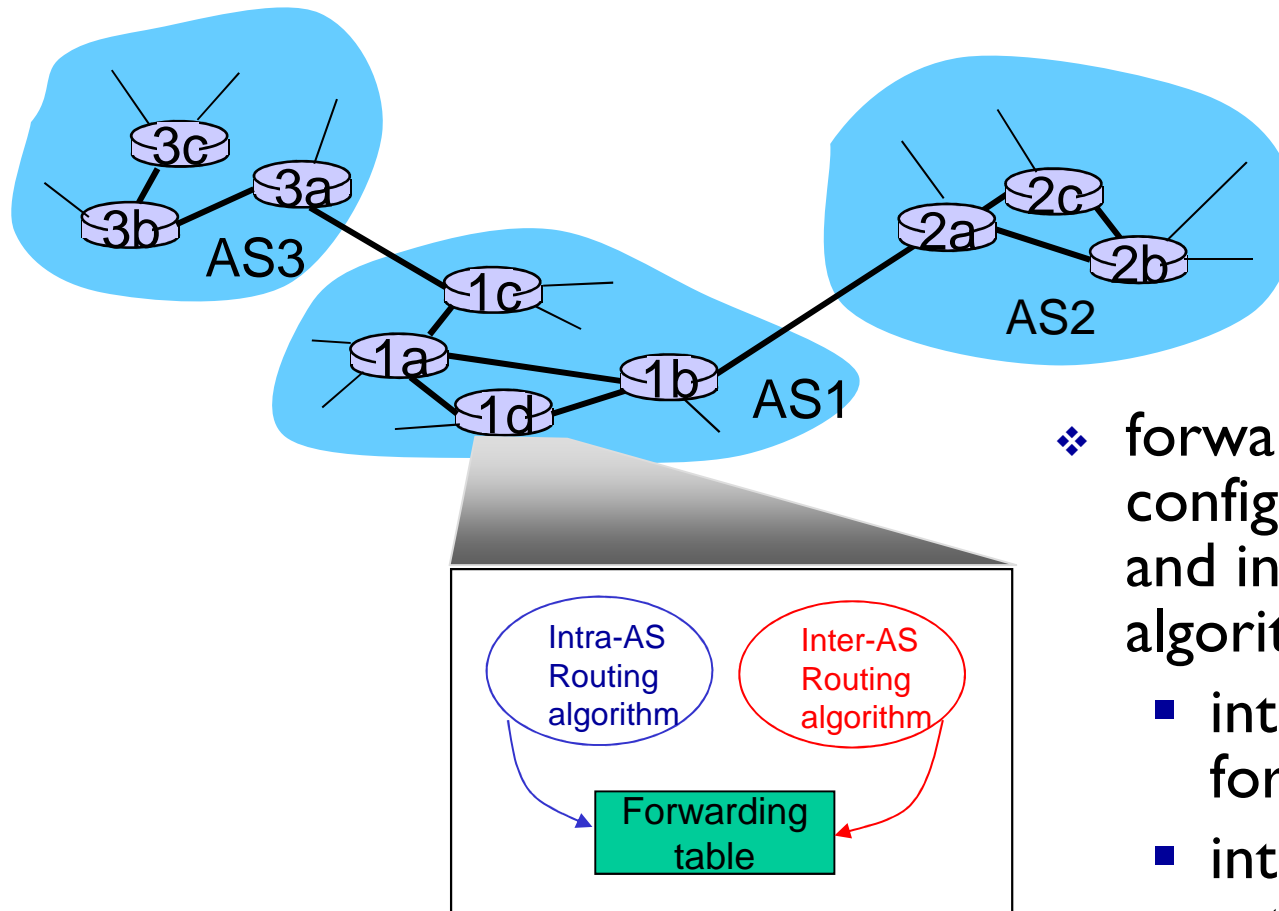
Forwarding table

❖ **forwarding table configured by both intra- and inter-AS routing algorithm**

- intra-AS sets entries for internal dests
- inter-AS & intra-AS sets entries for external dests

# Chapter 4: outline

# Intra-AS Routing

❖ also known as *interior gateway protocols (IGP)*

❖ most common intra-AS routing protocols:

- RIP: Routing Information Protocol

- OSPF: Open Shortest Path First

- IGRP: Interior Gateway Routing Protocol (Cisco proprietary)

# RIP ( Routing Information Protocol)

❖ included in BSD-UNIX distribution in 1982

❖ distance vector algorithm

▪ distance metric: # hops (max = 15 hops), each link has cost 1

▪ DVs exchanged with neighbors every 30 sec in response message (aka advertisement)

▪ each advertisement: list of up to 25 destination *subnets (in IP addressing sense)*
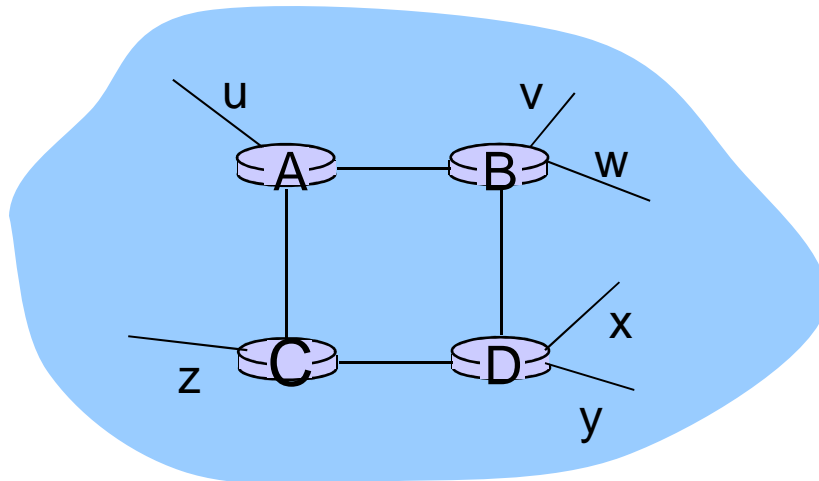


from router A to destination *subnets:*

| subnet | hops |
|--------|------|
| u | 1 |
| v | 2 |
| w | 2 |
| x | 3 |
| y | 3 |
| z | 2 |

# RIP: link failure, recovery

if no advertisement heard after 180 sec -->
neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly (?) propagates to entire net
- *poison reverse* used to prevent ping-pong loops (infinite distance = 16 hops)

# OSPF (Open Shortest Path First)

❖ "open": publicly available

❖ uses link state algorithm
  - LS packet dissemination
  - topology map at each node
  - route computation using Dijkstra's algorithm

❖ OSPF advertisement carries one entry per neighbor

❖ advertisements flooded to *entire* AS
  - carried in OSPF messages directly over IP (rather than TCP or UDP

❖ *IS-IS routing* protocol: nearly identical to OSPF

# OSPF "advanced" features (not in RIP)

❖ *security:* all OSPF messages authenticated (to prevent malicious intrusion)

❖ multiple same-cost paths allowed (only one path in RIP)

❖ for each link, multiple cost metrics for different TOS (e.g., satellite link cost set "low" for best effort ToS; high for real time ToS)

❖ integrated uni- and multicast support:
  ▪ Multicast OSPF (MOSPF) uses same topology data base as OSPF

❖ hierarchical OSPF in large domains.

# Hierarchical OSPF

boundary router

backbone router

backbone

area
border
routers

internal
routers

area 1

area 2

area 3
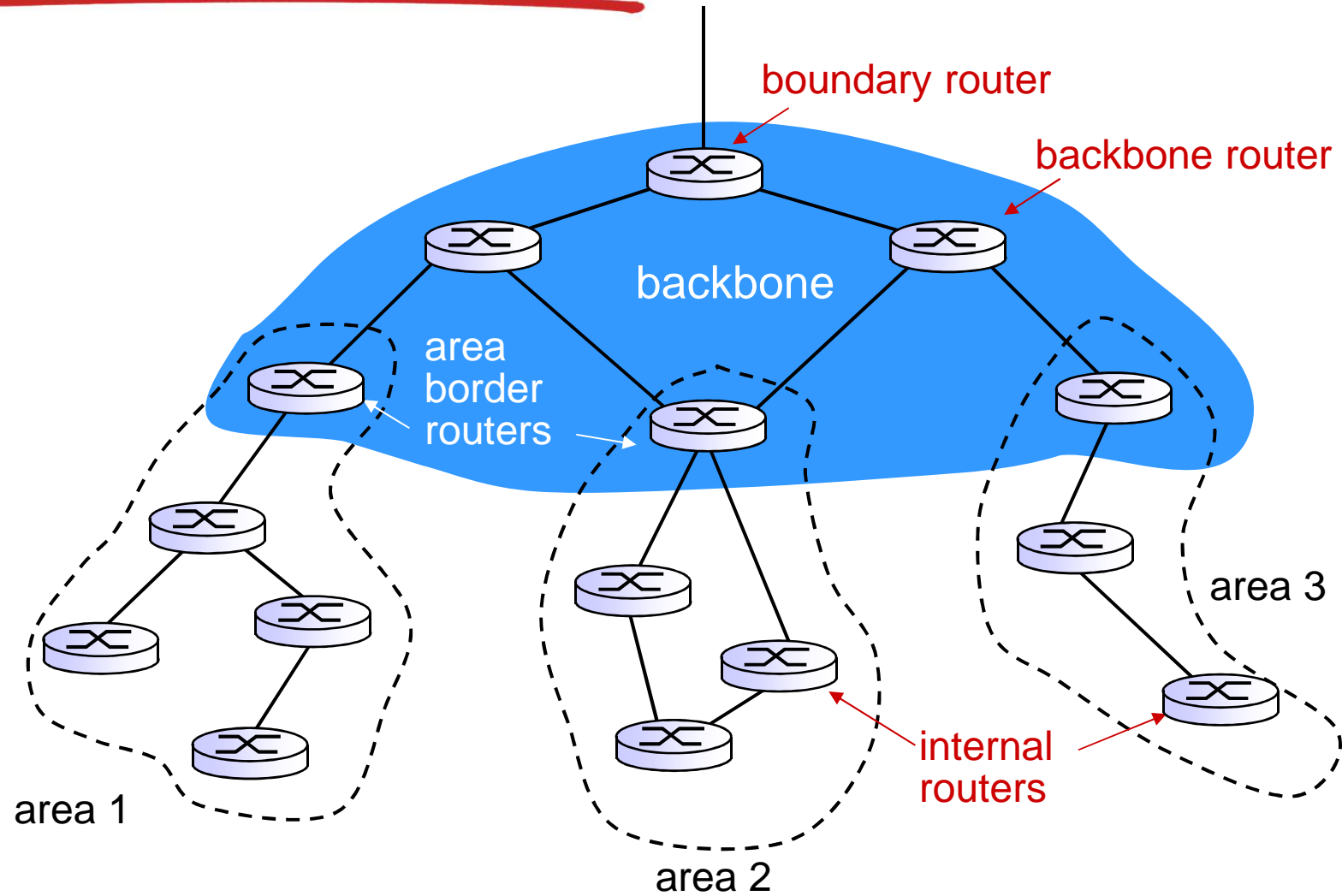
# Internet inter-AS routing: BGP

❖ **BGP (Border Gateway Protocol):** *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
❖ BGP provides each AS a means to:
  - **eBGP:** obtain subnet reachability information from neighboring ASs.
  - **iBGP:** propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and policy.
❖ allows subnet to advertise its existence to rest of Internet: *"I am here"*

# Inter-AS tasks

* suppose router in AS1 receives datagram destined outside of AS1:
    * router should forward packet to gateway router, but which one?

*AS1 must:*

1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

*job of inter-AS routing!*
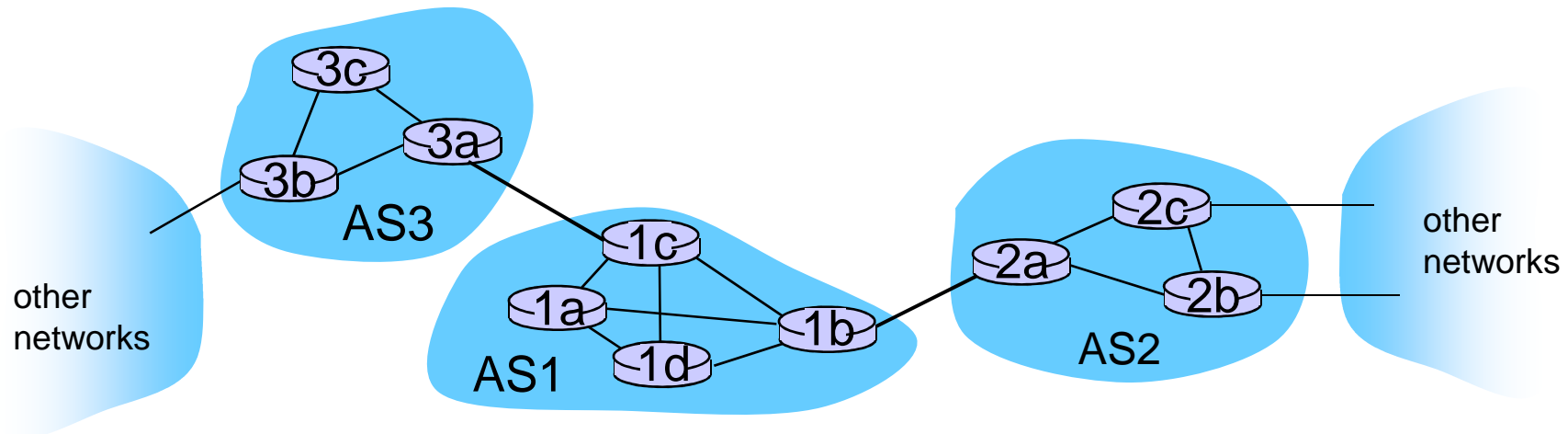
# Example: setting forwarding table in router 1d

❖ suppose AS1 learns (via inter-AS protocol) that subnet *x* reachable via AS3 (gateway 1c), but not via AS2

   ▪ inter-AS protocol propagates reachability info to all internal routers

❖ router 1d determines from intra-AS routing info that its interface *I* is on the least cost path to 1c
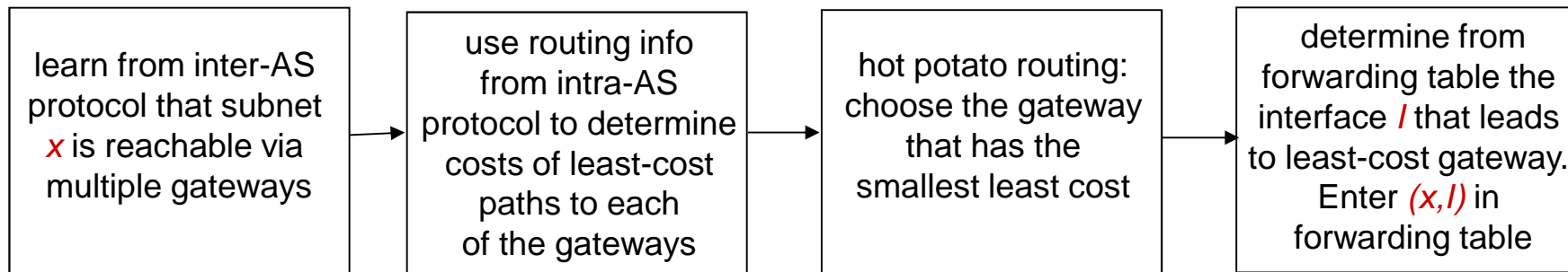
   ▪ installs forwarding table entry *(x,I)*
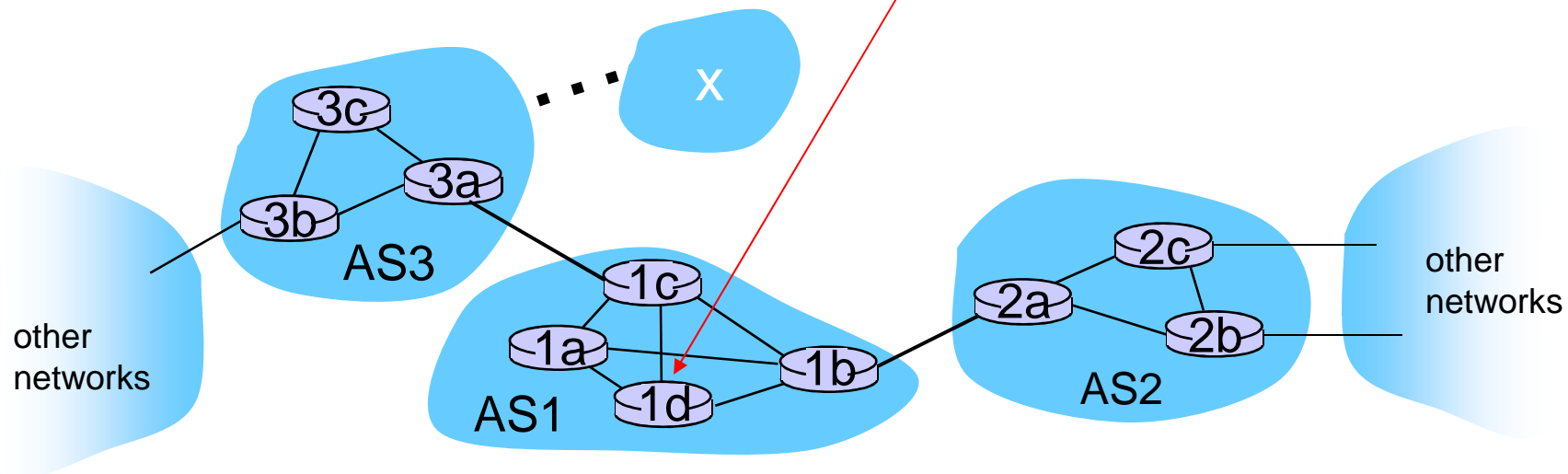
# Example: choosing among multiple ASes

❖ now suppose AS1 learns from inter-AS protocol that subnet x is reachable from AS3 *and* from AS2.

❖ to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest x

  ▪ this is also job of inter-AS routing protocol!
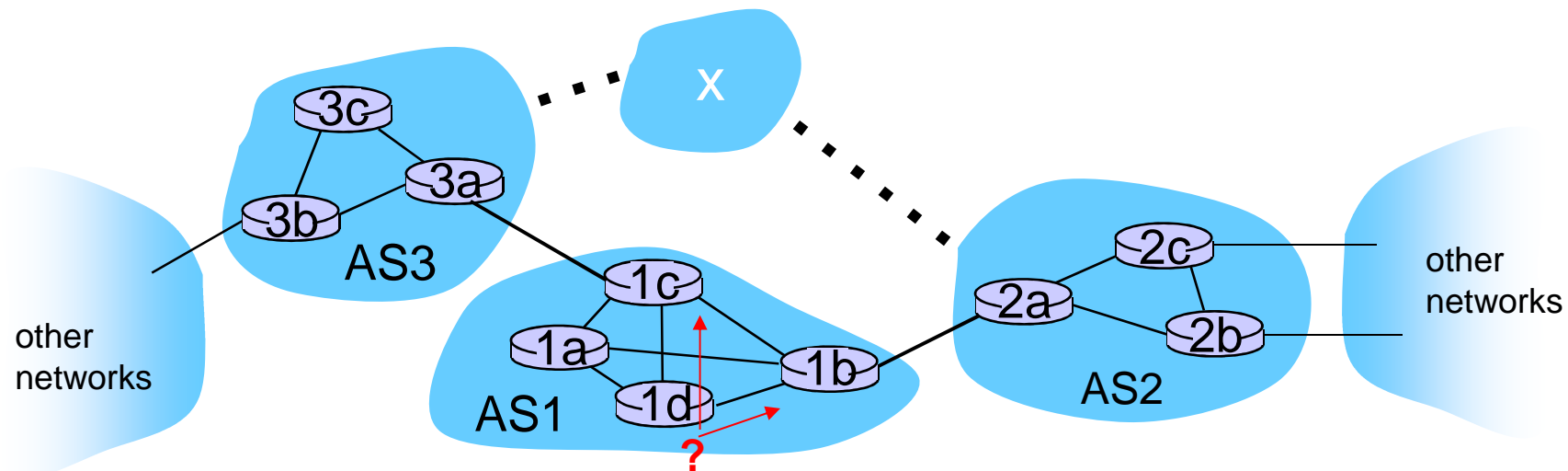
# Example: choosing among multiple ASes

- ❖ now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- ❖ to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest *x*
  - ▪ this is also job of inter-AS routing protocol!
- ❖ *hot potato routing: send* packet towards closest of two routers.

| learn from inter-AS protocol that subnet *x* is reachable via multiple gateways | → | use routing info from intra-AS protocol to determine costs of least-cost paths to each of the gateways | → | hot potato routing: choose the gateway that has the smallest least cost | → | determine from forwarding table the interface *I* that leads to least-cost gateway. Enter *(x,I)* in forwarding table |
|---|---|---|---|---|---|---|

# BGP basics

❖ **BGP session:** two BGP routers ("peers") exchange BGP messages:
  - advertising *paths* to different destination network prefixes ("path vector" protocol)
  - exchanged over semi-permanent TCP connections

❖ when AS3 advertises a prefix to AS1:
  - AS3 *promises* it will forward datagrams towards that prefix
  - AS3 can aggregate prefixes in its advertisement
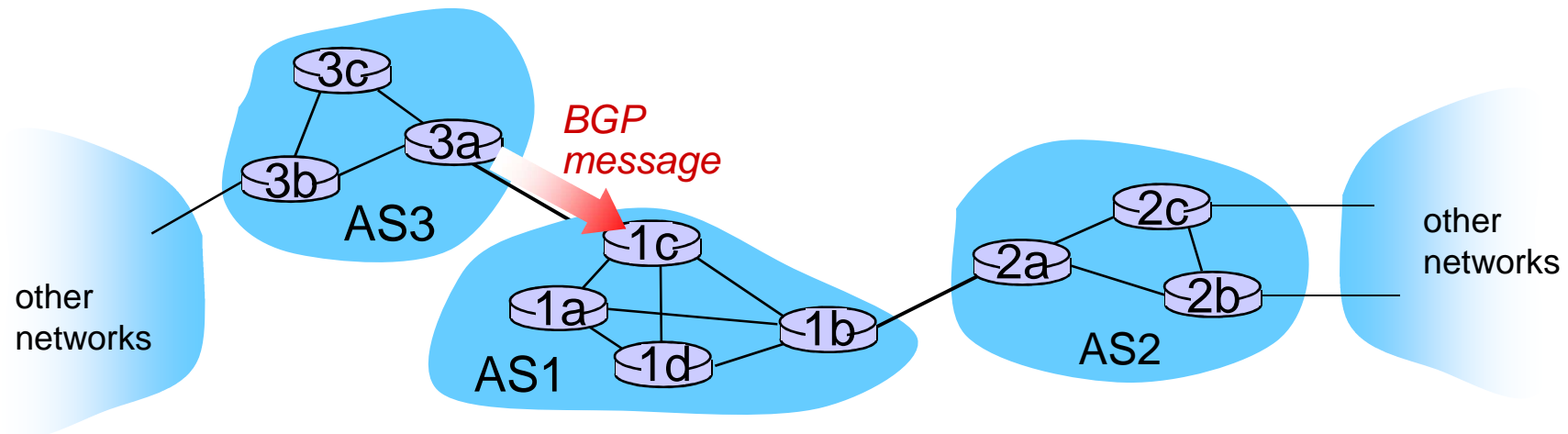
# BGP basics: distributing path information

- ❖ using eBGP session between 3a and 1c, AS3 sends prefix reachability info to AS1.
    - ▪ 1c can then use iBGP do distribute new prefix info to all routers in AS1
    - ▪ 1b can then re-advertise new reachability info to AS2 over 1b-to-2a eBGP session
- ❖ when router learns of new prefix, it creates entry for prefix in its forwarding table.
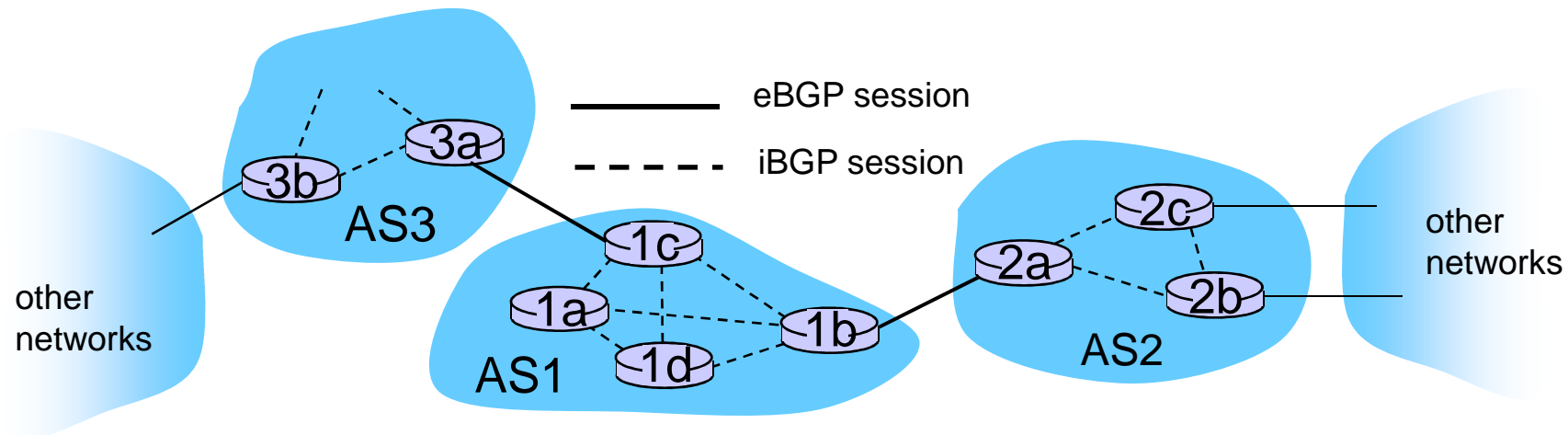
# Path attributes and BGP routes

- ❖ advertised prefix includes BGP attributes
  - ▪ prefix + attributes = "route"
- ❖ two important attributes:
  - ▪ AS-PATH: contains ASs through which prefix advertisement has passed: e.g., AS 67, AS 17
  - ▪ NEXT-HOP: indicates specific internal-AS router to next-hop AS. (may be multiple links from current AS to next-hop-AS)
- ❖ gateway router receiving route advertisement uses import policy to accept/decline
  - ▪ e.g., never route through AS x
  - ▪ *policy-based* routing

# BGP route selection

❖ router may learn about more than 1 route to destination AS, selects route based on:

1. local preference value attribute: policy decision
2. shortest AS-PATH
3. closest NEXT-HOP router: hot potato routing
4. additional criteria

# BGP routing policy
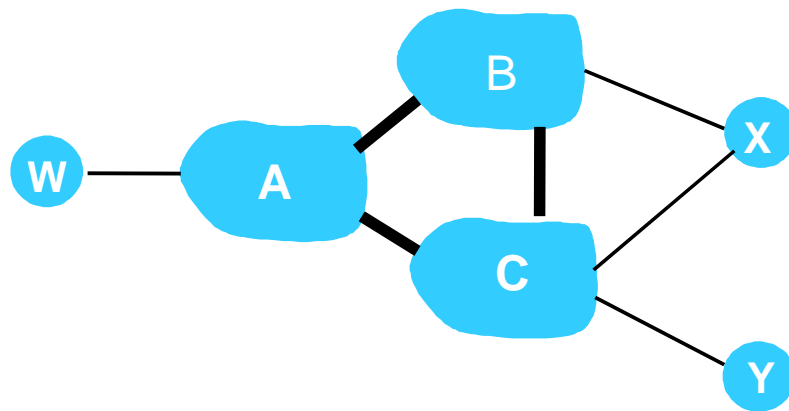


legend:

provider
network

customer
network:

❖ A,B,C are *provider networks*
❖ X,W,Y are customer (of provider networks)
❖ X is *dual-homed:* attached to two networks
  ▪ X does not want to route from B via X to C
  ▪ .. so X will not advertise to B a route to C

# BGP routing policy (2)



legend:

provider network

customer network:
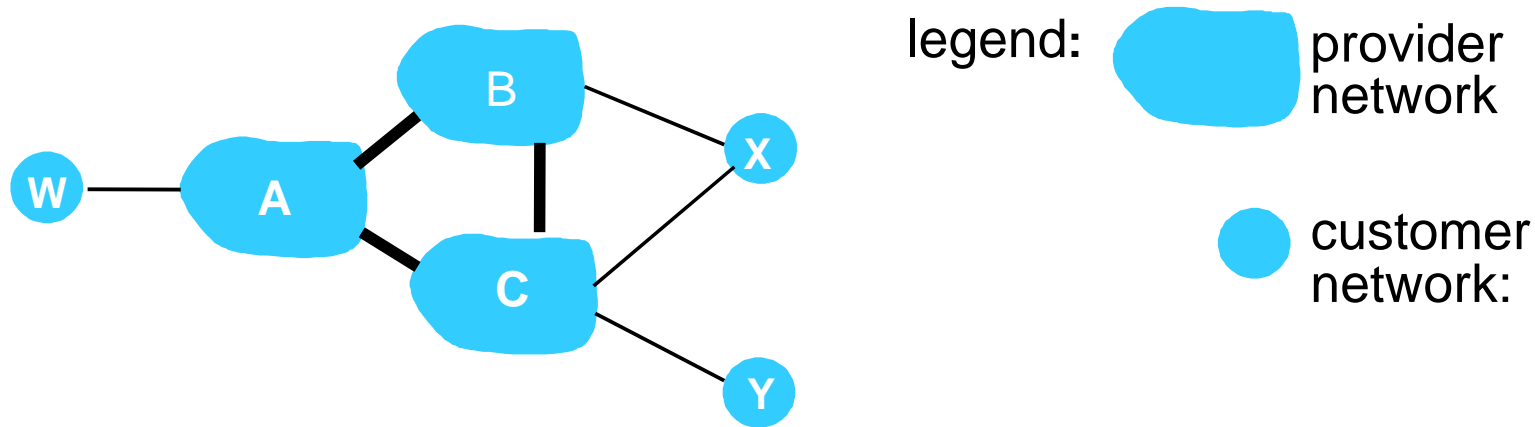
❖ A advertises path AW  to B

❖ B advertises path BAW to X

❖ Should B advertise path BAW to C?

- No way! B gets no "revenue" for routing CBAW since neither W nor C are B's customers
- B wants to force C to route to w via A
- B wants to route *only* to/from its customers!

# Why different Intra-, Inter-AS routing ?

*policy:*

- ❖ inter-AS: admin wants control over how its traffic routed, who routes through its net.
- ❖ intra-AS: single admin, so no policy decisions needed

*scale:*

- ❖ hierarchical routing saves table size, reduced update traffic

*performance:*

- ❖ intra-AS: can focus on performance
- ❖ inter-AS: policy may dominate over performance

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
 datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
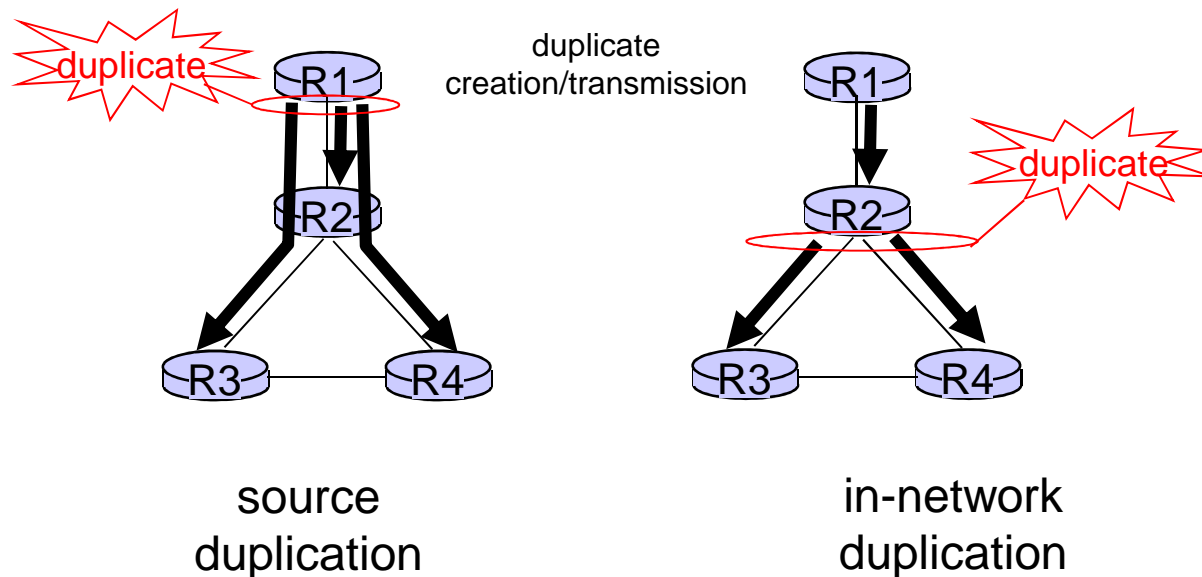- distance vector
- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP

4.7 broadcast and multicast
 routing

# Broadcast routing

❖ deliver packets from source to all other nodes

❖ source duplication is inefficient:



duplicate creation/transmission

source duplication

in-network duplication

❖ source duplication: how does source determine recipient addresses?

# In-network duplication

- ❖ *flooding:* when node receives broadcast packet, sends copy to all neighbors
  - ▪ problems: cycles & broadcast storm
- ❖ *controlled flooding:* node only broadcasts pkt if it hasn't broadcast same packet before
  - ▪ node keeps track of packet ids already broadcasted
  - ▪ or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- ❖ *spanning tree:*
  - ▪ no redundant packets received by any node

# Spanning tree

- ❖ first construct a spanning tree
- ❖ nodes then forward/make copies only along spanning tree
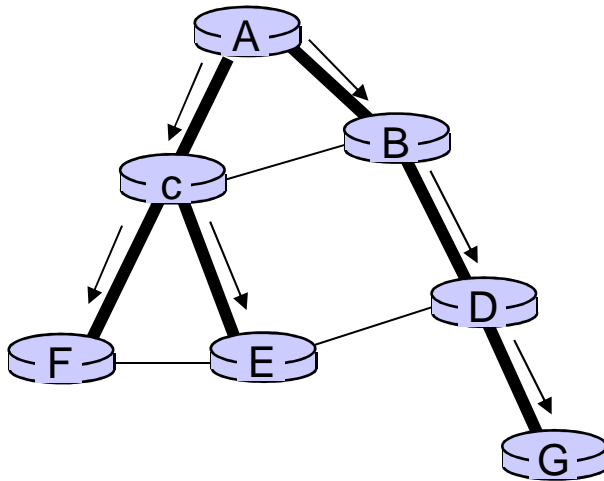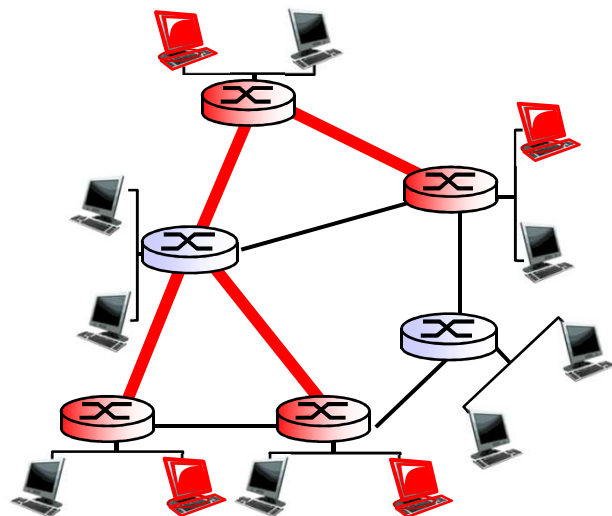


(a) broadcast initiated at A
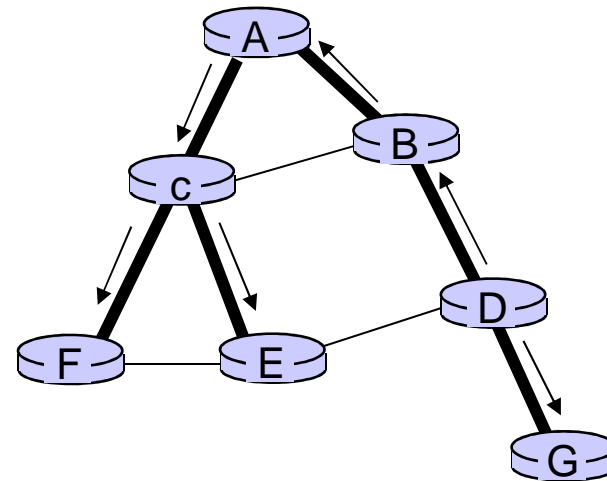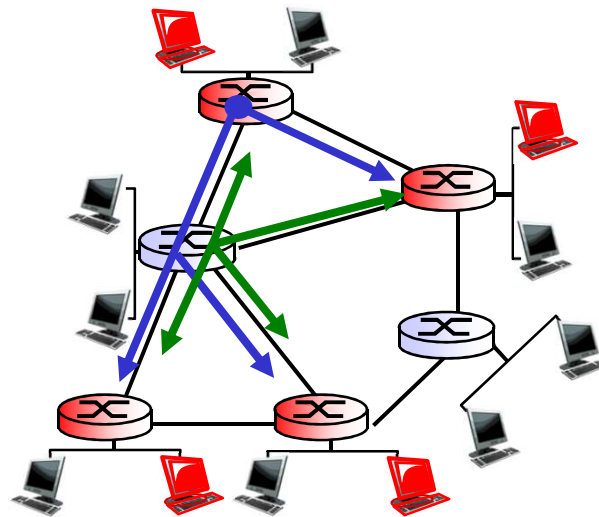
(b) broadcast initiated at D

# Multicast routing: problem statement

*goal:* find a tree (or trees) connecting routers having local mcast group members

- ❖ *tree:* not all paths between routers used
- ❖ *shared-tree:* same tree used by all group members
- ❖ *source-based:* different tree from each sender to rcvrs



legend

group member

not group member

router with a group member

router without group member

shared tree

source-based trees

# Approaches for building mcast trees

approaches:

❖ *source-based tree:* one tree per source
  - shortest path trees
  - reverse path forwarding

❖ *group-shared tree:* group uses one tree
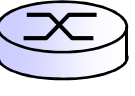  - minimal spanning (Steiner)
  - center-based trees

group management:

❖ dedicated addressing range: 224.0.0.0/4

❖ tree ⟷ group membership?

# Shortest path tree

❖ **mcast forwarding tree: tree of shortest path routes from source to all receivers**

  ▪ Dijkstra's algorithm

s: source

R1

2

R4

1

R2

5

3

4

R3

R5

6

R6

R7
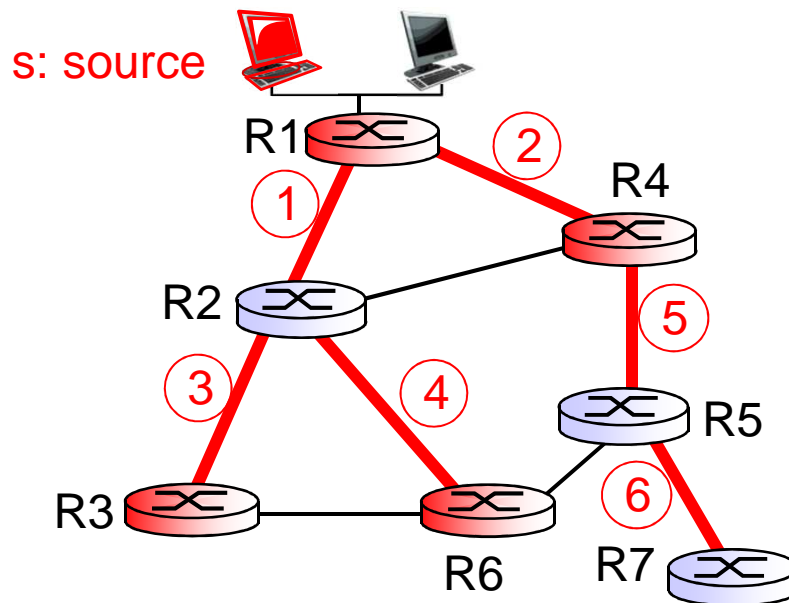
LEGEND

router with attached group member

router with no attached group member

ⓘ link used for forwarding, i indicates order link added by algorithm

# Reverse path forwarding

❖ rely on router's knowledge of unicast shortest path from it  to sender

❖ each router has simple forwarding behavior:

*if* (mcast datagram received on incoming link on
  shortest path back to center)
  *then* flood datagram onto all outgoing links
  *else* ignore datagram

# Reverse path forwarding: example



s: source

LEGEND

router with attached group member

router with no attached group member

→ datagram will be forwarded

↦ datagram will not be forwarded
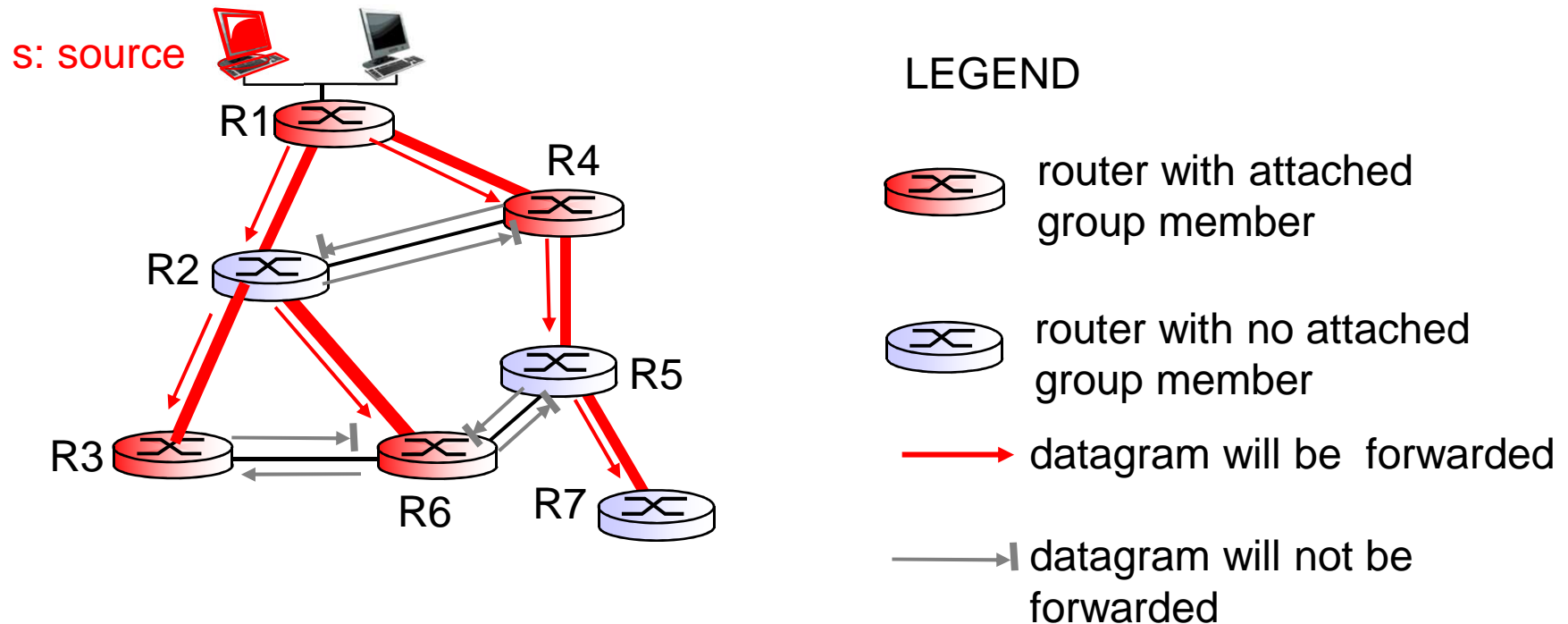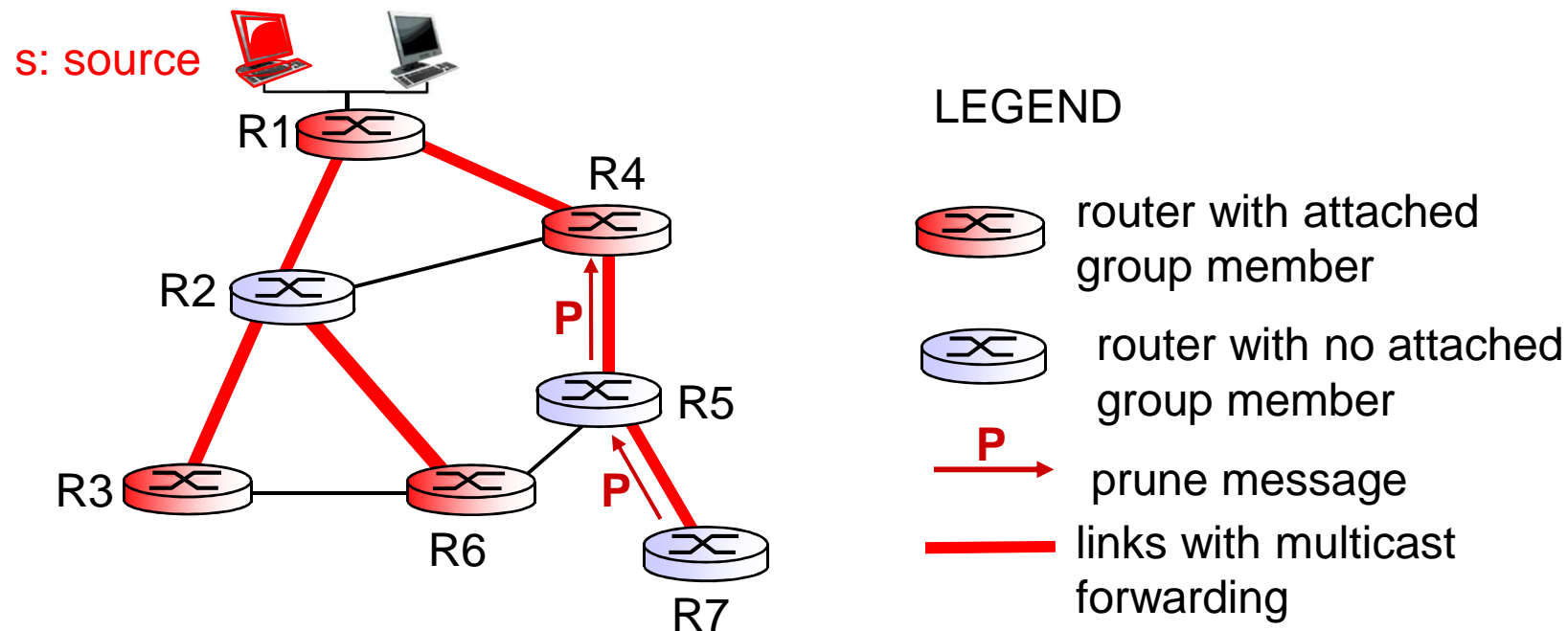
❖ result is a source-specific *reverse* SPT
- may be a bad choice with asymmetric links

# Group management: pruning

❖ forwarding tree contains subtrees with no mcast group members

  ▪ no need to forward datagrams down subtree
  ▪ "prune" msgs sent upstream by router with no downstream group members

s: source

R1

R2

R3

R4

R5

R6

R7

P

P

P

LEGEND

router with attached group member

router with no attached group member

P →  prune message

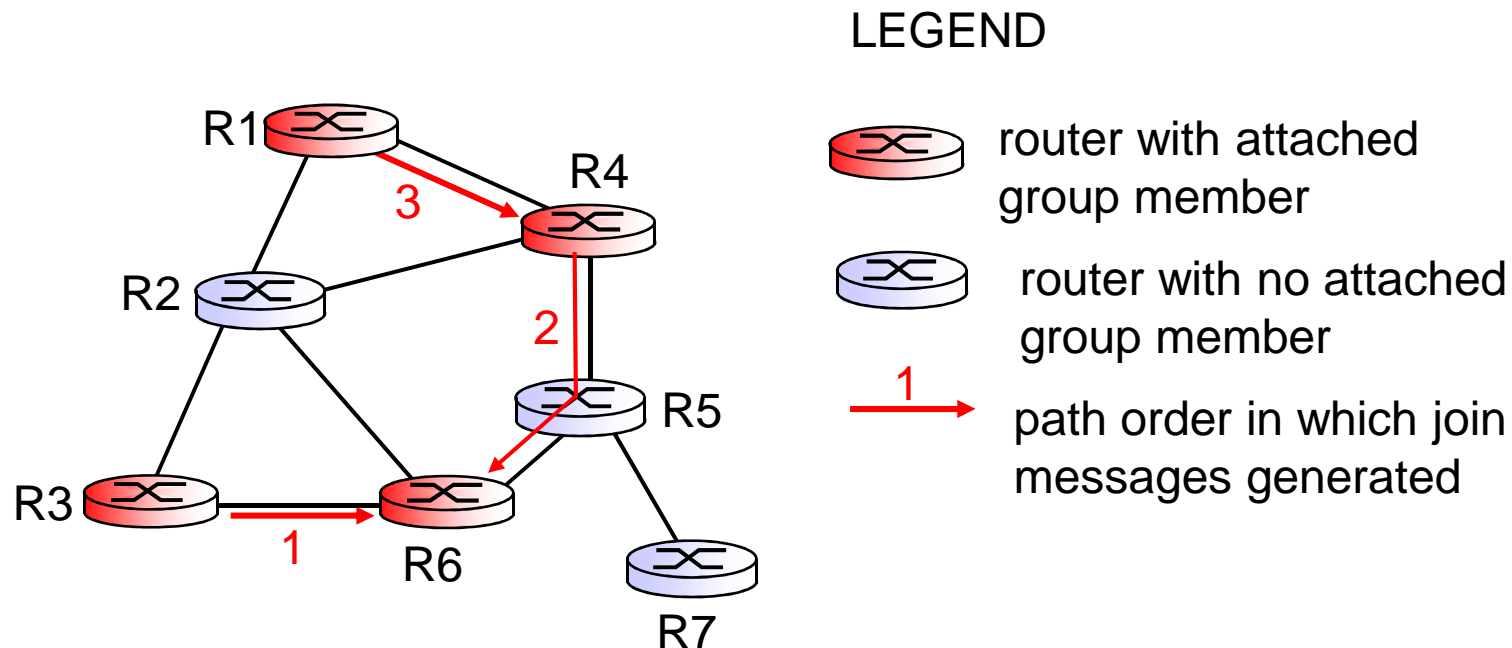——— links with multicast forwarding

# Shared-tree: Steiner tree

❖ *Steiner tree:* minimum cost tree connecting all routers with attached group members
❖ problem is NP-complete
❖ excellent heuristics exists
❖ not used in practice:
  ▪ computational complexity
  ▪ information about entire network needed
  ▪ monolithic: rerun whenever a router needs to join/leave

# Center-based trees

❖ single delivery tree shared by all

❖ one router identified as *"center"* of tree

❖ to join:

- edge router sends unicast *join-msg* addressed to center router

- *join-msg* "processed" by intermediate routers and forwarded towards center

- *join-msg* either hits existing tree branch for this center, or arrives at center

- path taken by *join-msg* becomes new branch of tree for this router

# Center-based trees: example

suppose R6 chosen as center:

LEGEND

router with attached group member

router with no attached group member

1 → path order in which join messages generated

R1

R2

R3

R4

R5

R6

R7

3

2

1

# Internet Multicasting Routing: DVMRP

- ❖ **DVMRP:** distance vector multicast routing protocol, RFC1075

- ❖ *flood and prune:* reverse path forwarding, source-based tree

  - ▪ RPF tree based on DVMRP's own routing tables constructed by communicating DVMRP routers

  - ▪ no assumptions about underlying unicast

  - ▪ initial datagram to mcast group flooded everywhere via RPF

  - ▪ routers not wanting group: send upstream prune msgs

# DVMRP: continued...

❖ *soft state:* DVMRP router periodically (1 min.) "forgets" branches are pruned:

  ▪ mcast data again flows down unpruned branch
  ▪ downstream router: reprune or else continue to receive data

❖ routers can quickly regraft to tree

  ▪ following IGMP join at leaf

❖ odds and ends

  ▪ commonly implemented in commercial router

# PIM: Protocol Independent Multicast

❖ not dependent on any specific underlying unicast routing algorithm (works with all)

❖ two different multicast distribution scenarios :

## dense:

❖ group members densely packed, in "close" proximity.

❖ bandwidth more plentiful

## sparse:

❖ # networks with group members small wrt # interconnected networks

❖ group members "widely dispersed"

❖ bandwidth not plentiful

# Consequences of sparse-dense dichotomy:

## dense

- ❖ group membership by routers *assumed* until routers explicitly prune
- ❖ *data-driven* construction on mcast tree (e.g., RPF)
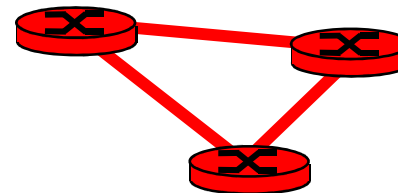- ❖ bandwidth and non-group-router processing *profligate*

## sparse:

- ❖ no membership until routers explicitly join
- ❖ *receiver- driven* construction of mcast tree (e.g., center-based)
- ❖ bandwidth and non-group-router processing *conservative*

# Tunneling

*Q:* how to connect "islands" of multicast routers in a "sea" of unicast routers?
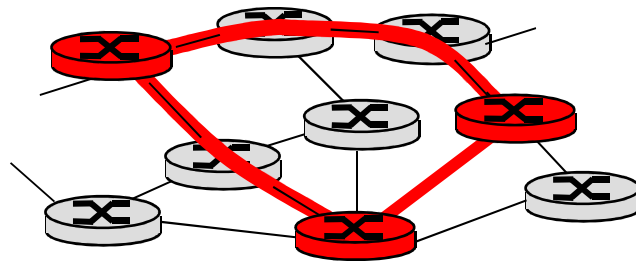


physical topology                    logical topology

❖ mcast datagram encapsulated inside "normal" (non-multicast-addressed) datagram

❖ normal IP datagram sent thru "tunnel" via regular IP unicast to receiving mcast router (recall IPv6 inside IPv4 tunneling)

❖ receiving mcast router unencapsulates to get mcast datagram

# Chapter 4: *done!*

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format, IPv4 addressing, ICMP, IPv6

4.5 routing algorithms
- link state, distance vector, hierarchical routing

4.6 routing in the Internet
- RIP, OSPF, BGP

4.7 broadcast and multicast routing

❖ understand principles behind network layer services:
- network layer service models, forwarding versus routing how a router works, routing (path selection), broadcast, multicast

❖ instantiation, implementation in the Internet