Network layer
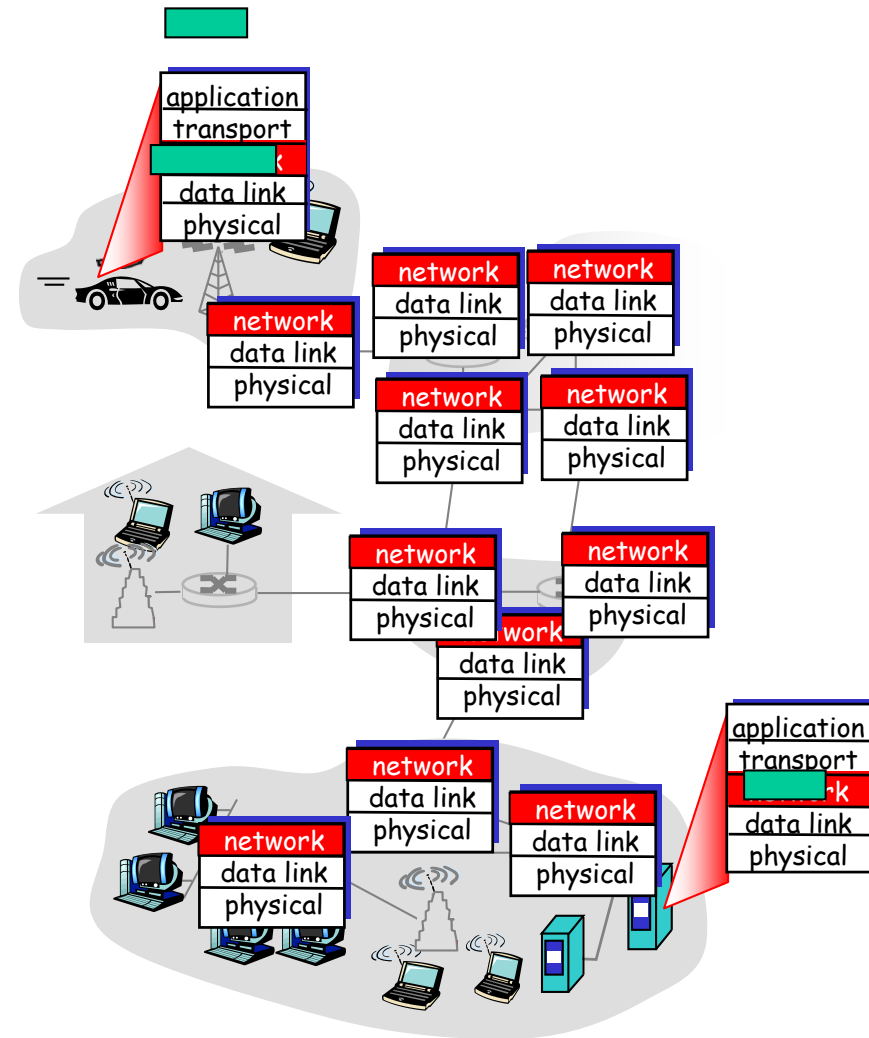
❖ transport segment from sending to receiving host

❖ on sending side encapsulates segments into datagrams

❖ on receiving side, delivers segments to transport layer

❖ network layer protocols in *every* host, router
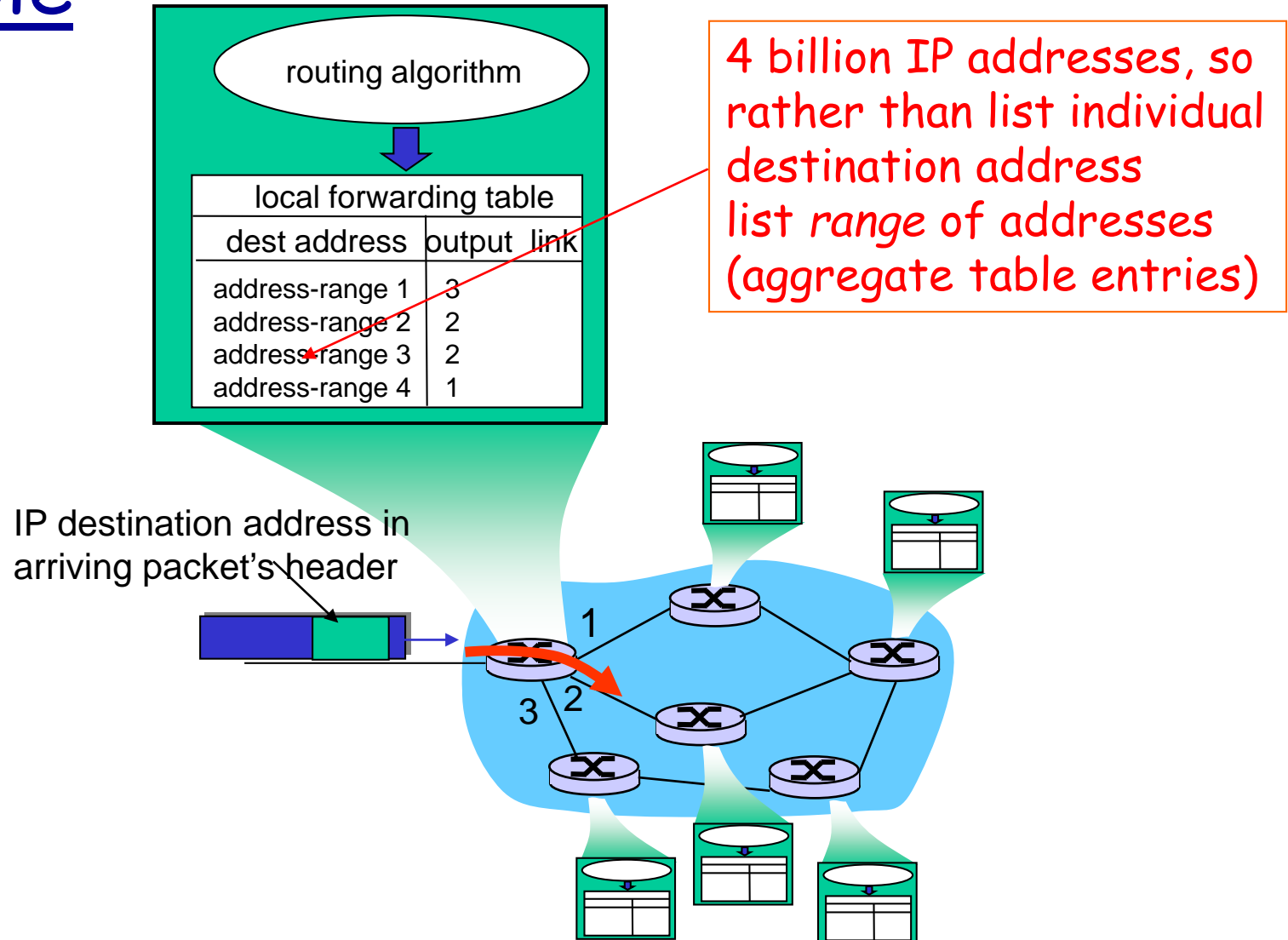
❖ router examines header fields in all IP datagrams

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 destination

  ▪ *routing algorithms*

Analogy (driving):

❖ routing: process of planning trip from source to destimation

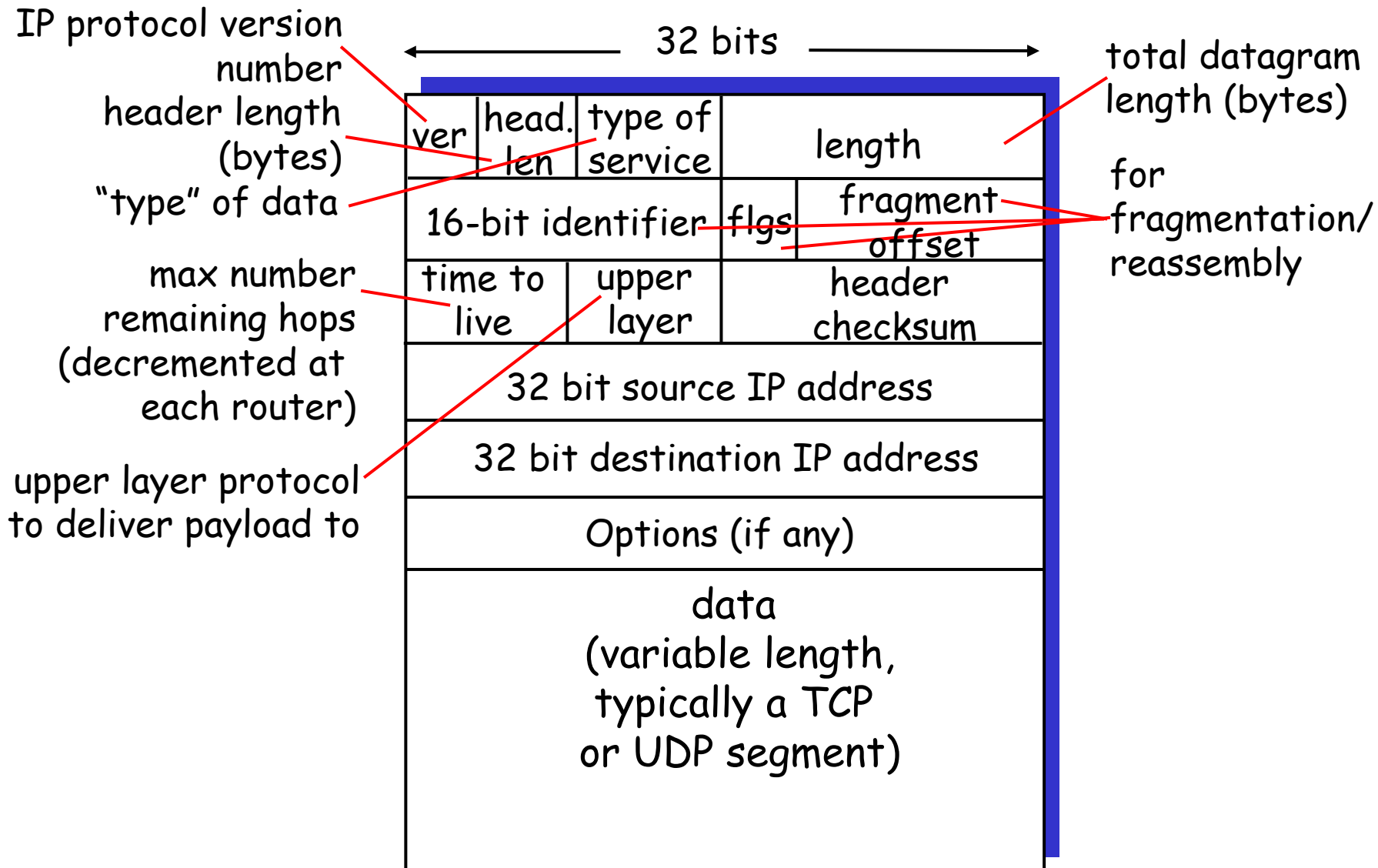❖ forwarding: process of getting through single interchange

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
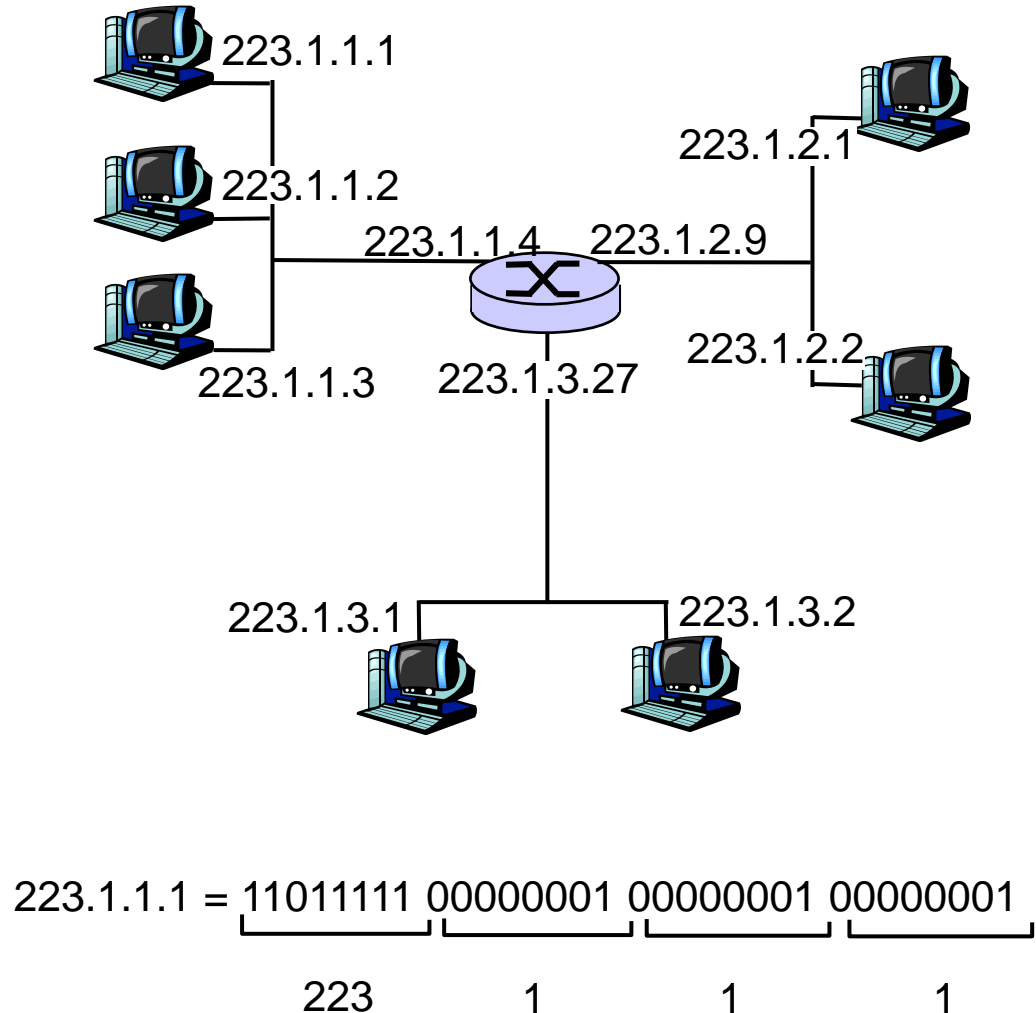
3   2

Datagram Forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

IPv4 datagram format

IP protocol version number

header length (bytes)

"type" of data

32 bits

total datagram length (bytes)

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)

for fragmentation/ reassembly

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to

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 interface
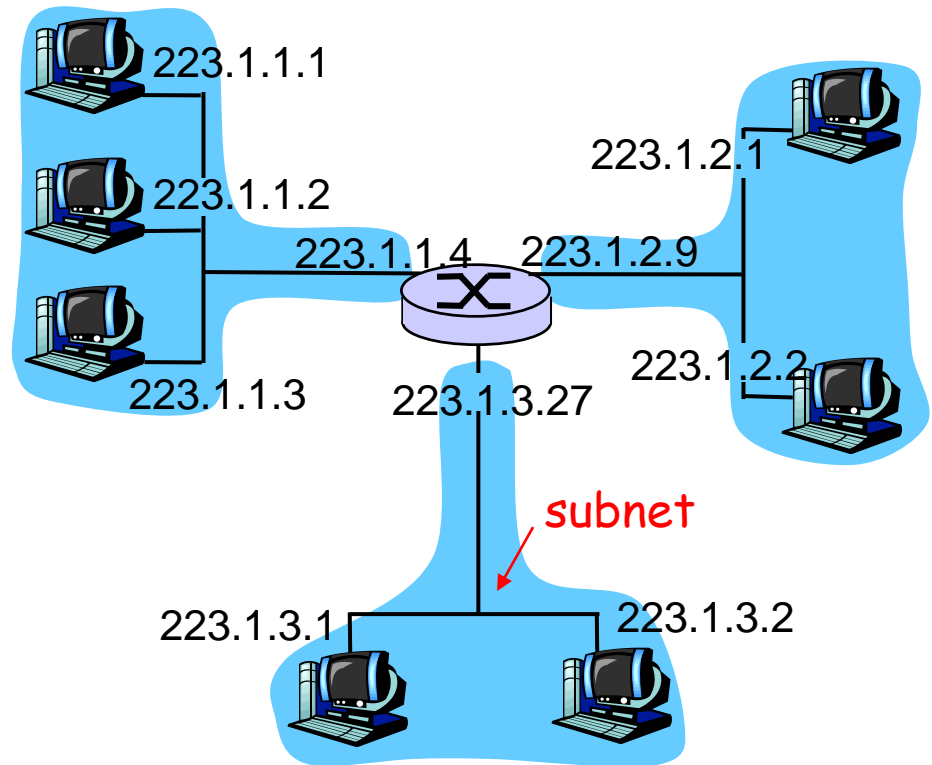 ▪ IP addresses associated with each interface

223.1.1.1

223.1.1.2

223.1.1.4   223.1.2.9

223.1.2.1

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

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

223.1.2.1

223.1.1.4   223.1.2.9

223.1.1.3   223.1.3.27

223.1.2.2

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
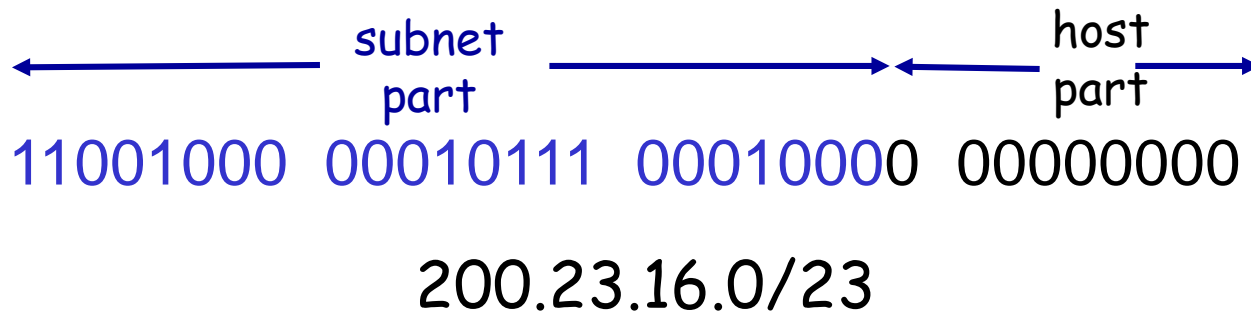
223.1.2.0/24

223.1.3.0/24
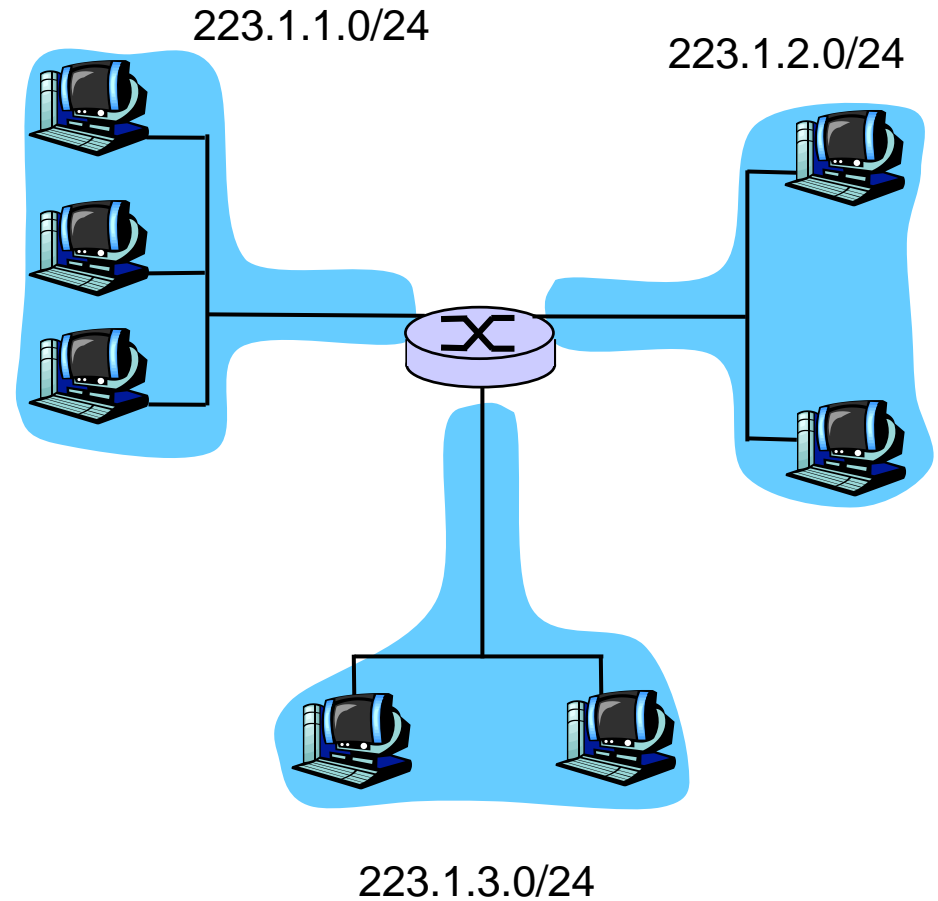
Subnet mask: /24

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 a *host* get an 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 a server
  ▪ "plug-and-play"

IP addressing: the last word...

Q: How does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

IPv6

❖ **Initial motivation:** 32-bit address space soon to be completely allocated.

❖ Additional motivation:

- header format helps speed processing/forwarding
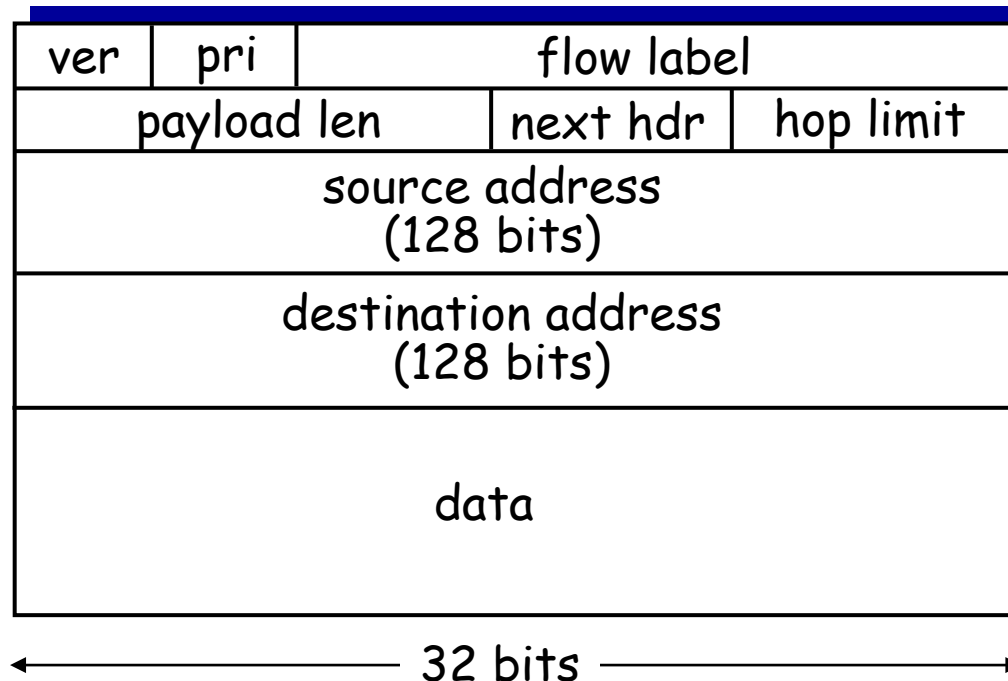- header changes to facilitate QoS

  IPv6 datagram format:

- fixed-length 40 byte header
- no fragmentation allowed

IPv6 Header

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 ————————→

IPv6 Addresses

(IPv4 addresses, 32 bits long, written in decimal, separated by periods)
IPv6 addresses, 128 bits long, written in hexadecimal, separated by colons.

3ffe:1900:4545:3:200:f8ff:fe21:67cf

Leading zeros can be omitted in each field, *:0003: is written :3:*. A double colon (::) can be used once in an address to replace multiple fields of zeros.

fe80:0000:0000:0000:0200:f8ff:fe21:67cf

can be written

fe80::200:f8ff:fe21:67cf

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 (Internet Control Message Protocol) :* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
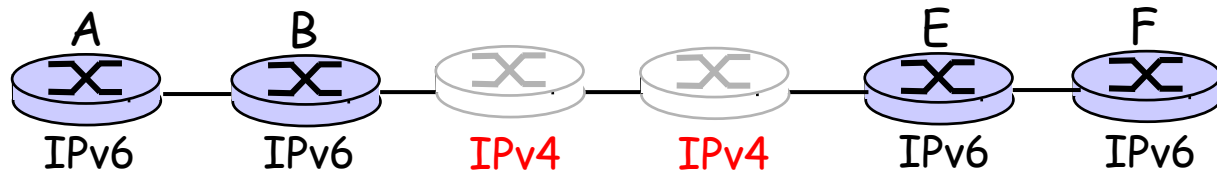 - multicast group management functions

Transition From IPv4 To IPv6

❖ **Not all routers can be upgraded simultaneous**

  ▪ How will the network operate with mixed IPv4 and IPv6 routers?

❖ *Tunneling:* IPv6 carried as payload in IPv4 datagram among IPv4 routers

❖ *Dual stack:* Both IPv4 and IPv6 protocol implemented in the routers

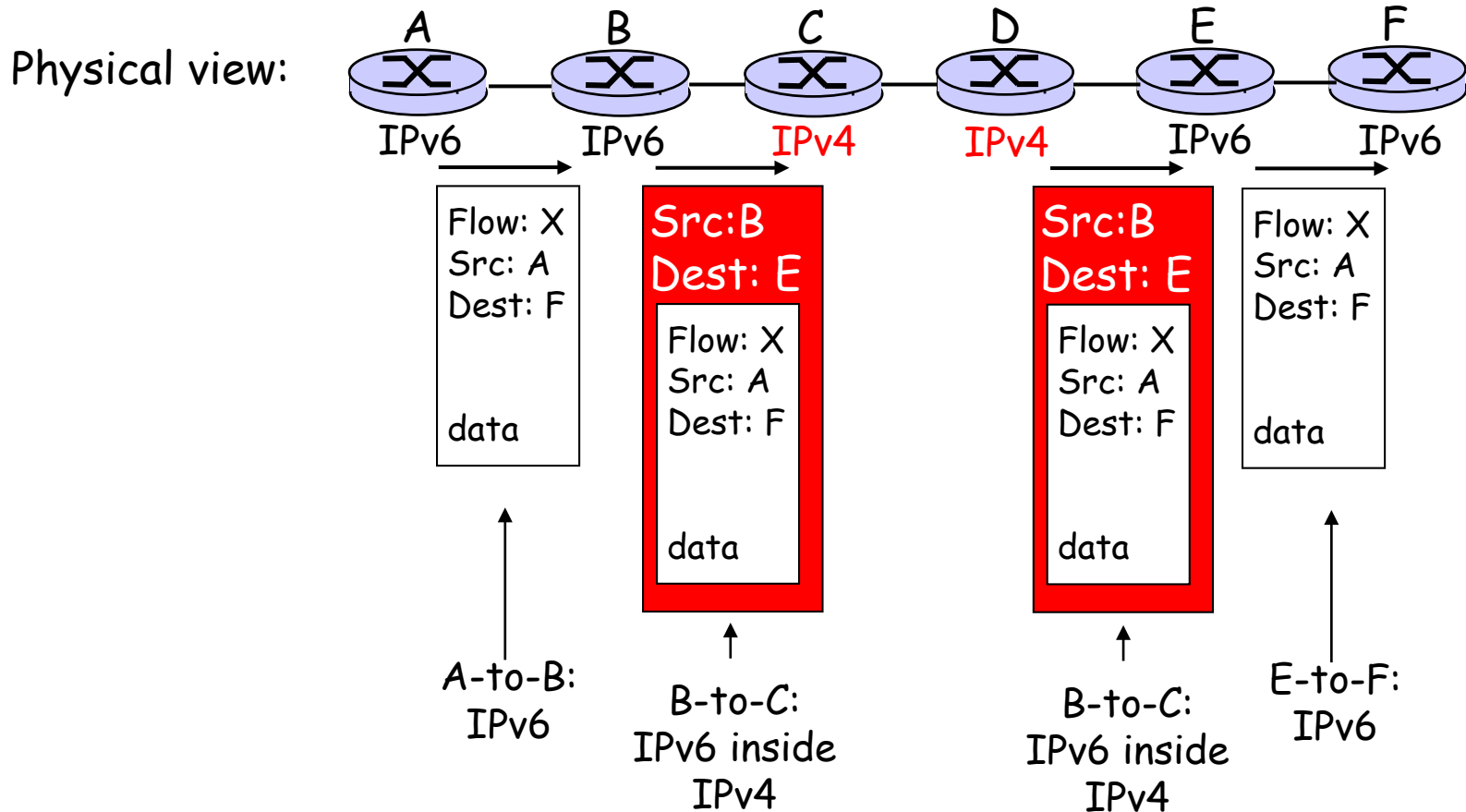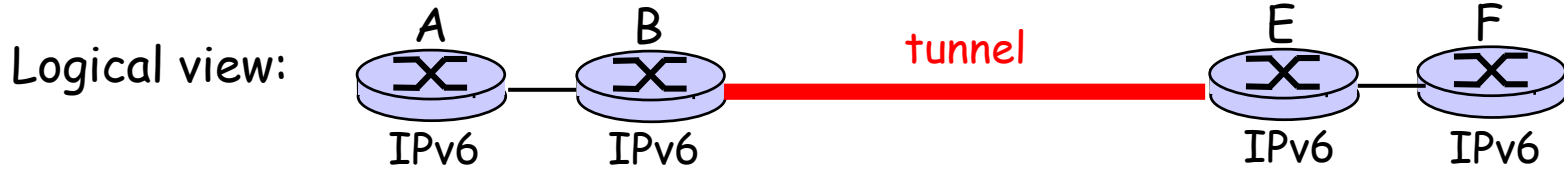❖ *Translation:* When transiting, translate between protocols (information lost)
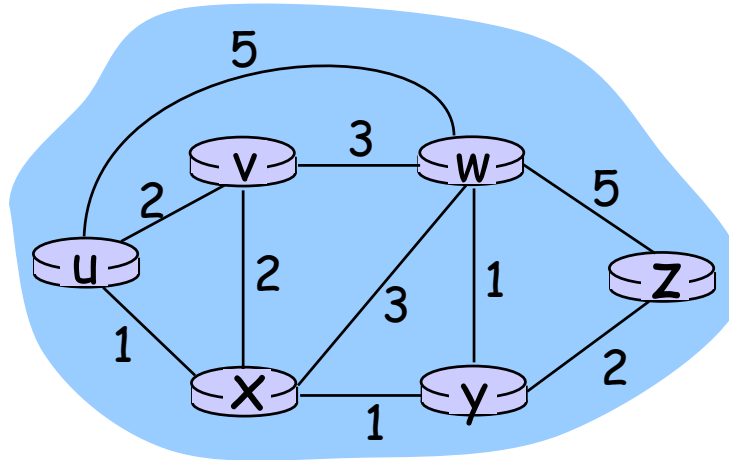
Tunneling

Logical view:

A
IPv6

B
IPv6

tunnel

E
IPv6

F
IPv6

Physical view:

A
IPv6

B
IPv6

IPv4

IPv4

E
IPv6

F
IPv6

Tunneling

Logical view:

A   B   tunnel   E   F

IPv6   IPv6   IPv6   IPv6

Physical view:

A   B   C   D   E   F

IPv6   IPv6   IPv4   IPv4   IPv6   IPv6

Flow: X	Src:B	Src:B	Flow: X
Src: A	Dest: E	Dest: E	Src: A
Dest: F			Dest: F
	Flow: X	Flow: X	
	Src: A	Src: A	
	Dest: F	Dest: F	
data			data
	data	data	

A-to-B:
IPv6

B-to-C:
IPv6 inside
IPv4

B-to-C:
IPv6 inside
IPv4

E-to-F:
IPv6

Routingalgoritmer

Hur skapas innehållet i routingtabellerna??

Graph abstraction

Graph: G = (N,E)
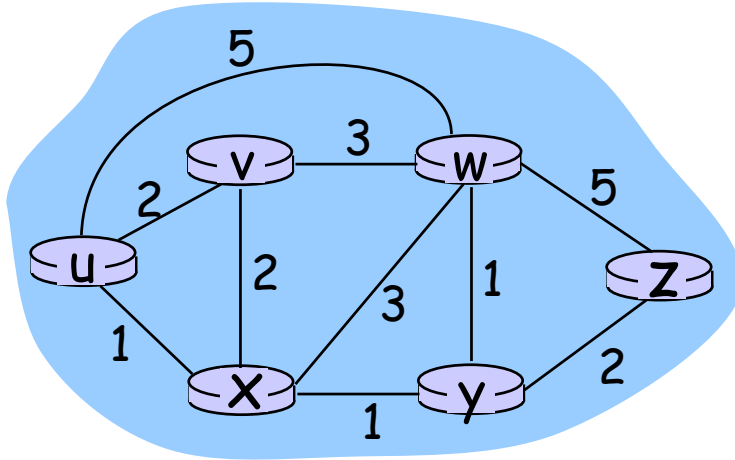
N = set of routers = { u, v, w, x, y, z }

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

Remark: Graph abstraction is useful in other network contexts

Example: 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 $(x_1, x_2, x_3,..., x_p) = c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p)$

Question: What's the least-cost path between u and z ?

Routing algorithm: algorithm that finds least-cost path

Routing Algorithm classification

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

Static or dynamic?

Static:

- ❖ routes change slowly over time

Dynamic:

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

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 destinations

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 destination 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'
```
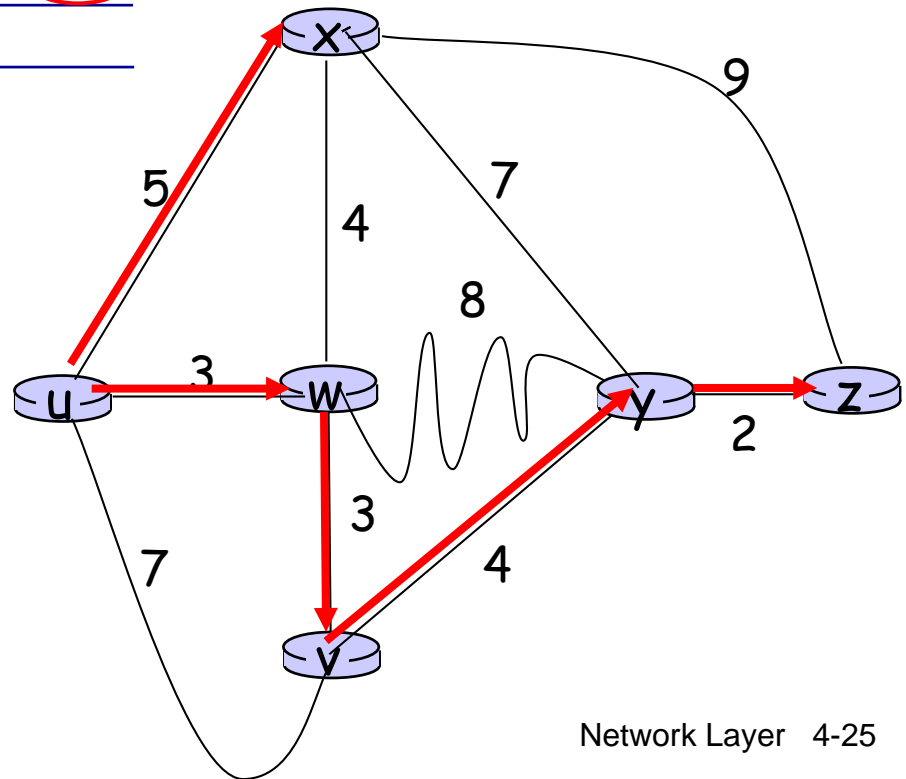
Dijkstra's algorithm: example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) 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)

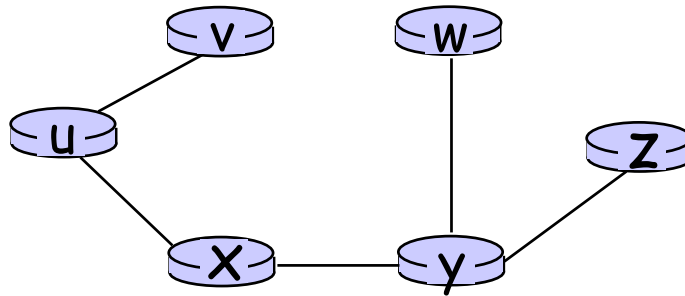# Dijkstra's algorithm: another example

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					
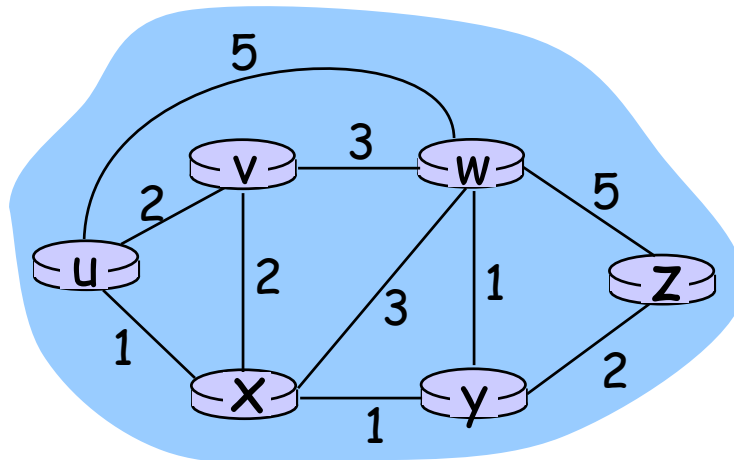
Dijkstra's algorithm: example (2)

Resulting shortest-path tree from u:

Resulting forwarding table in u:

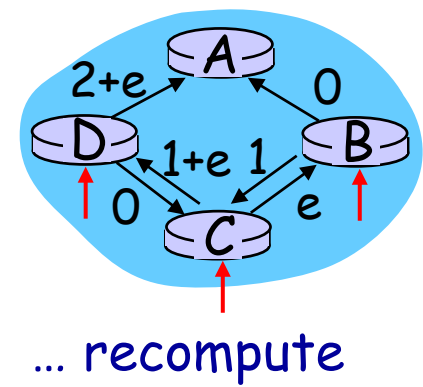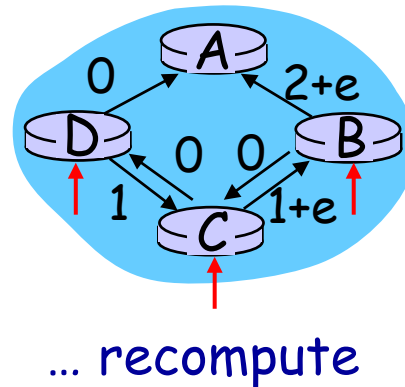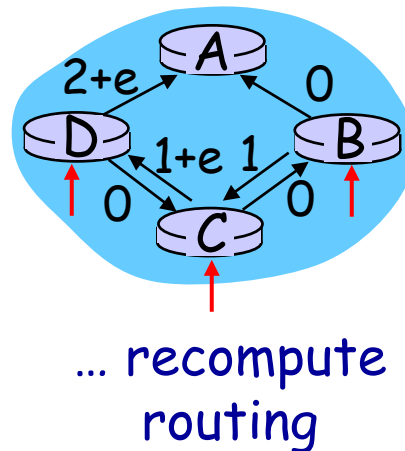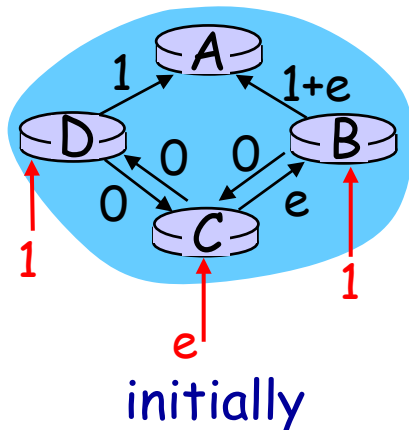| destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

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., link cost = amount of carried traffic

initially

... recompute routing

... recompute

... recompute

Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)
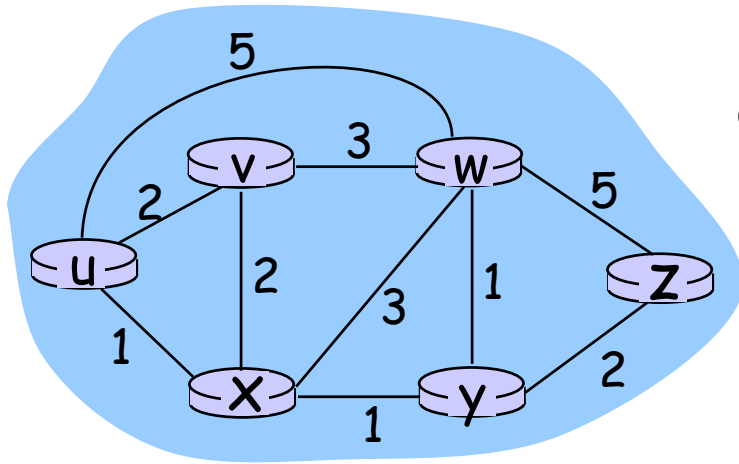Define
$d_x(y)$:= cost of least-cost path from x to y

Then

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

where *min* is taken over all neighbors v of x

Bellman-Ford example

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

B-F 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 that achieves minimum is next
hop in shortest path ➜ forwarding table

Distance Vector Algorithm

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

❖ node x:

  ▪ knows cost to each neighbor v: c(x,v)

  ▪ maintains its neighbors' distance vectors. For each neighbor v, x maintains
    $D_v = [D_v(y): y \in N]$

Distance vector algorithm

Basic 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 B-F equation:

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

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

Distance Vector Algorithm
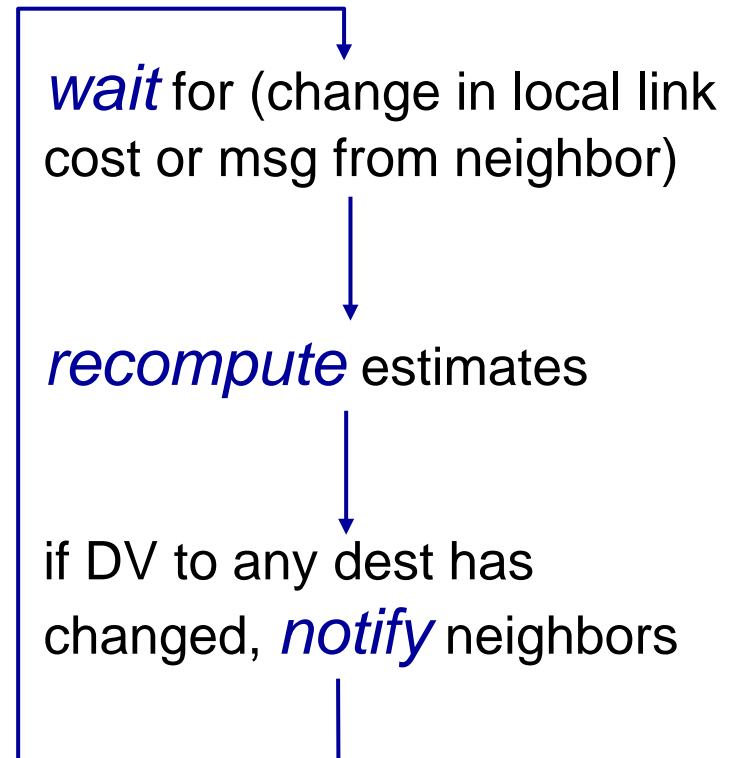
Iterative, asynchronous:

each local iteration caused by:

- ❖ local link cost change
- ❖ DV update message from neighbor

Distributed:

- ❖ each node notifies neighbors *only* 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

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from

cost to

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from

node y table

cost to

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from

node z table

cost to

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from

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

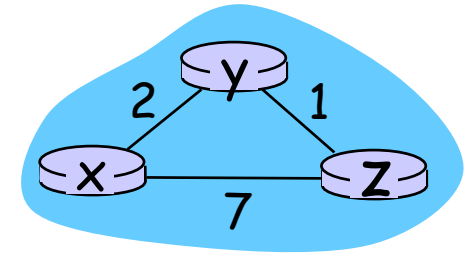| cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from

cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from

cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

from

node y table

cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from

cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

from

cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

from

node z table

cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from

cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from

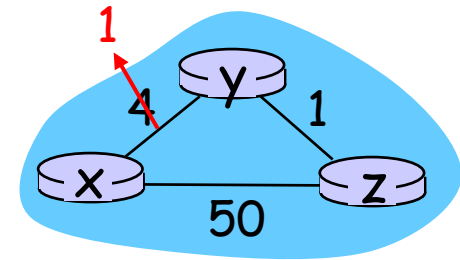| cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

from

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:

❖ good news travels fast

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

❖ 44 iterations before algorithm stabilizes: see text
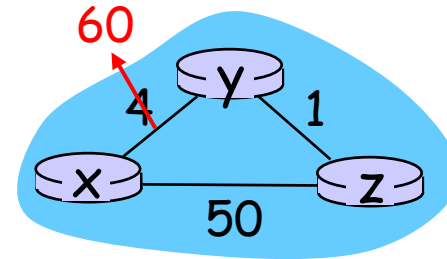
Poisoned reverse:

❖ 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

❖ <u>LS:</u> with n nodes, E links, O(nE) msgs sent

❖ <u>DV:</u> exchange between neighbors only
  ▪ convergence time varies

Speed of Convergence

❖ <u>LS:</u> $O(n^2)$ algorithm requires O(nE) msgs
  ▪ may have oscillations

❖ <u>DV</u>: convergence time varies
  ▪ may be routing loops
  ▪ count-to-infinity problem

Robustness: what happens if router malfunctions?

<u>LS:</u>
  ▪ node can advertise incorrect *link* cost
  ▪ each node computes only its *own* table

<u>DV:</u>
  ▪ DV node can advertise incorrect *path* cost
  ▪ each node's table used by others
    • error propagate thru network