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

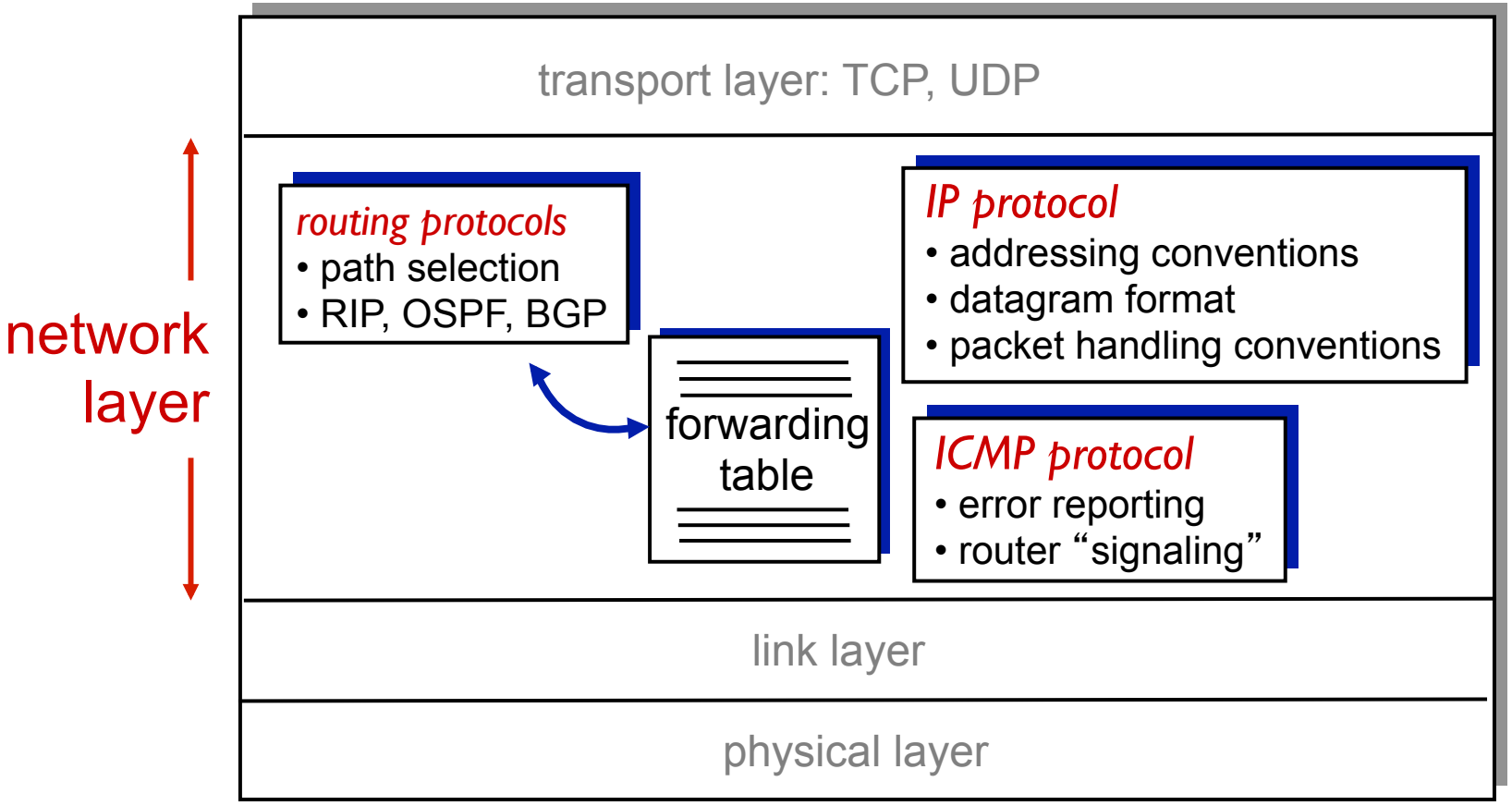
4.6 routing in the Internet

- RIP
- OSPF
- BGP

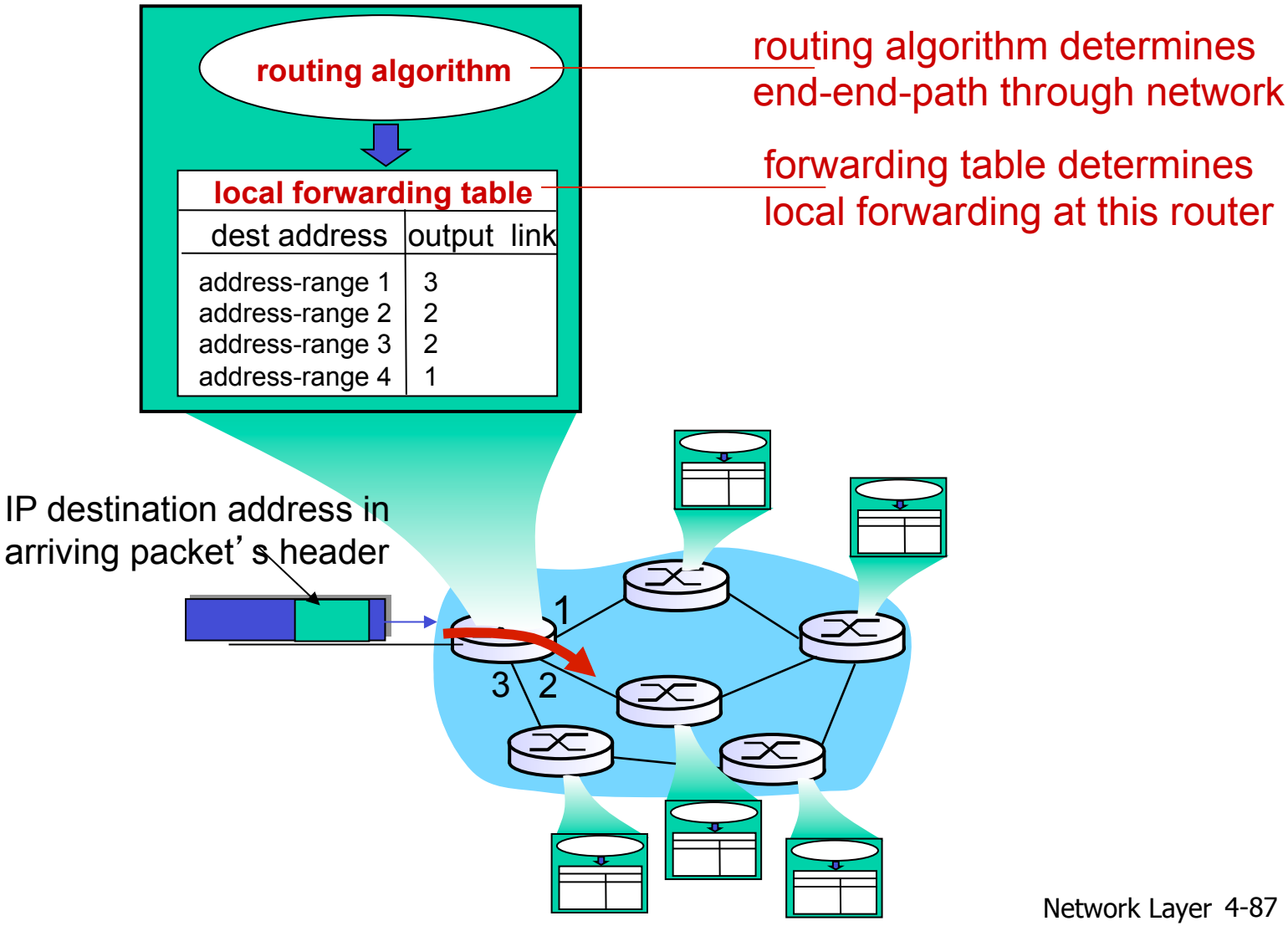
4.7 broadcast and multicast routing

The Internet network layer

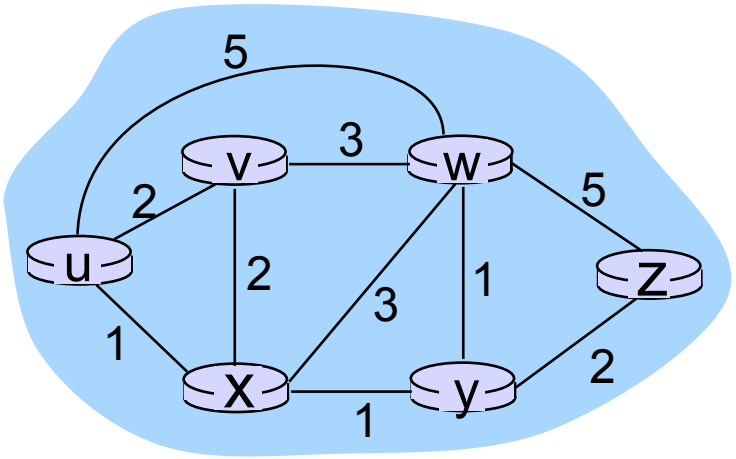
host, router network layer functions:



Interplay between routing, forwarding



Graph abstraction



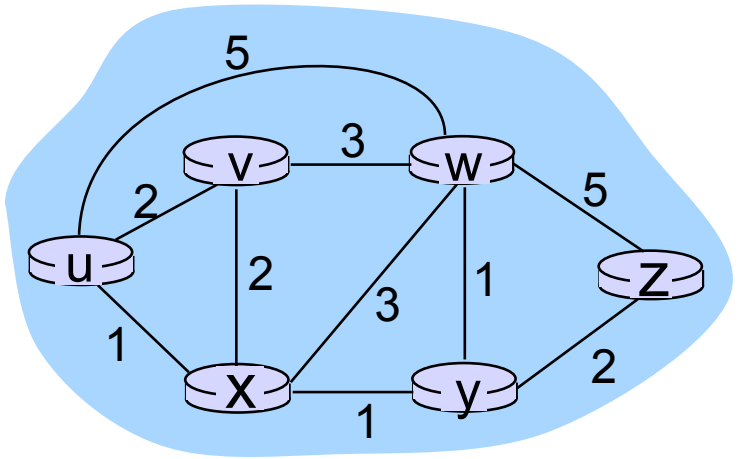
graph: G = (N,E)

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 $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

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

Routing algorithm classifications

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

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

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

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

Detour:

See Supplemental Materials
on Dijkstra Algorithm

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 dest. v
- p(v): predecessor node along path from source to v
- S: set of nodes whose least cost path definitively known

Dijsktra's Algorithm

```
1  Initialization:
2    S = {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 S such that d(w) is a minimum
10     add w to S
11     Update d(v) for all v adjacent to w and not in S :
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 S
```

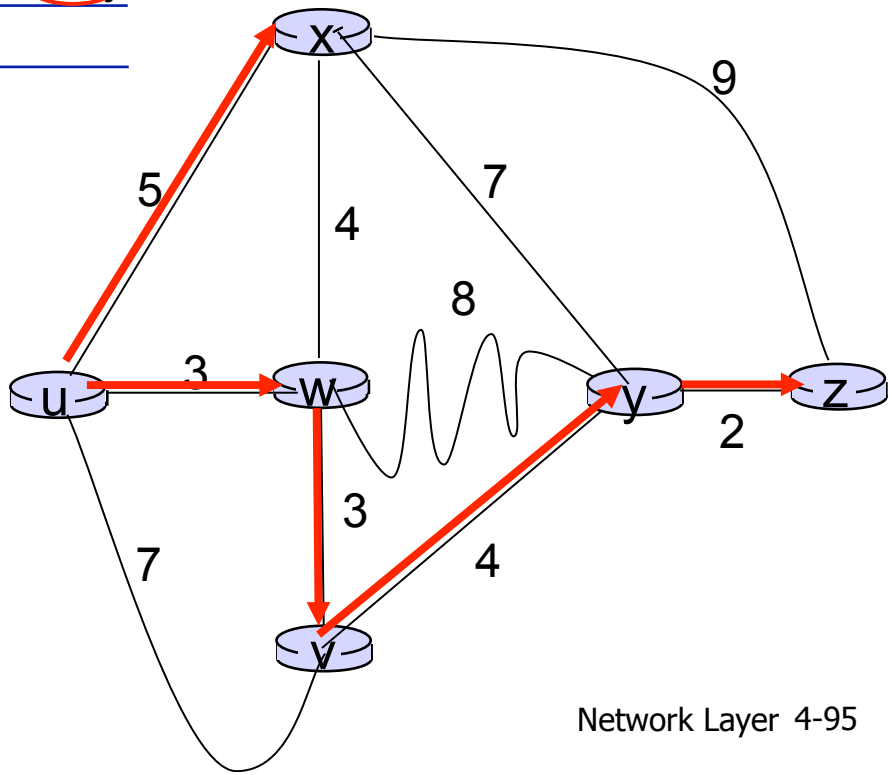
Dijkstra's algorithm: example

Step	S	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					

```
8  Loop
9   find w not in S such that d(w) is a minimum
10   add w to S
11   Update d(v) for all v adjacent to w and not in S :
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 S
```

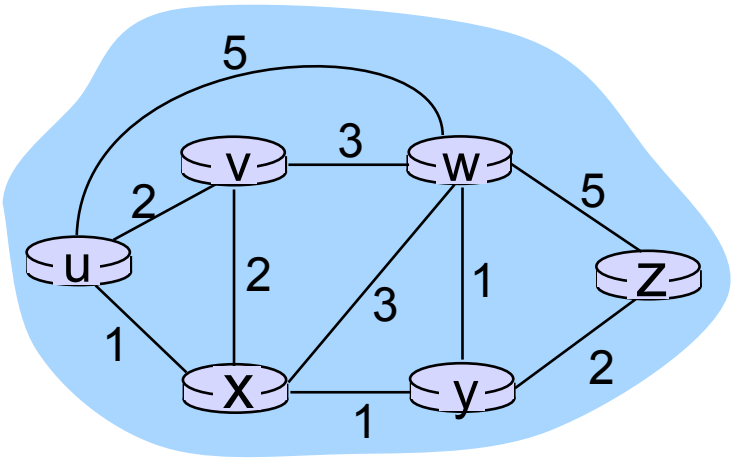
notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



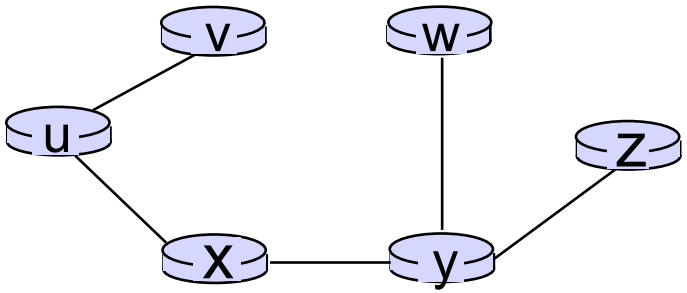
Dijkstra's algorithm: another example

Step	S	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:



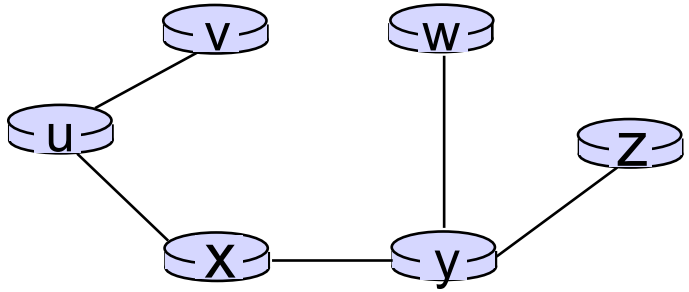
resulting forwarding table in u:

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

Dijkstra's algorithm "reversed":

compute distances from all nodes to a destination

Resulting shortest-path tree to u:



More natural for forwarding Tables (to destination u):

from node	next hop	distance
v	u	2
x	u	1
y	x	2
w	y	3
z	y	4

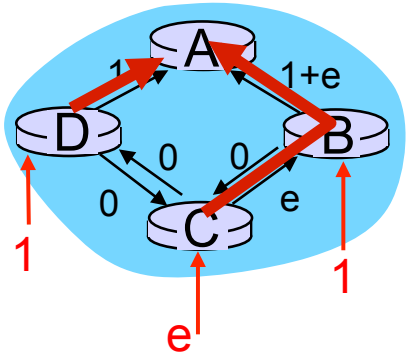
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

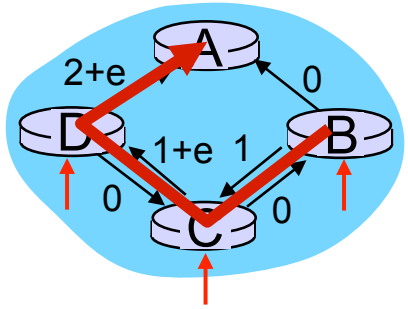
- each iteration: need to check all nodes, w, not in S
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(mlogn)

oscillations possible:

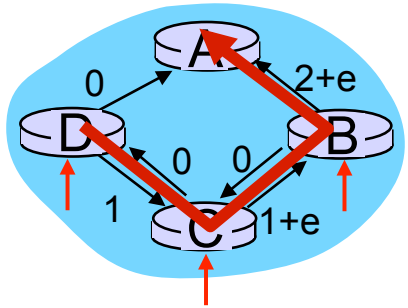
- e.g., suppose link cost = amount of carried traffic:



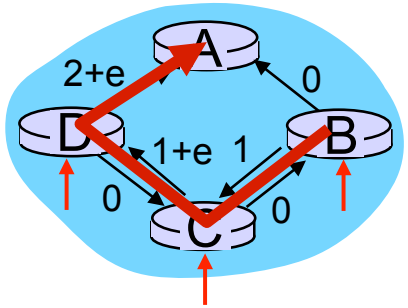
initially



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

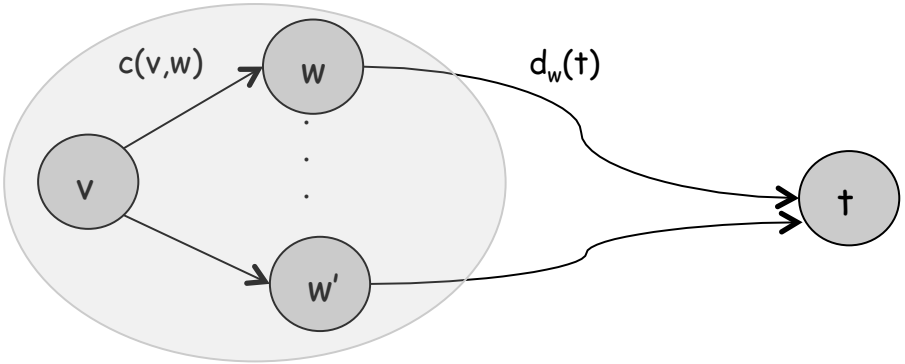
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

Bellman-Ford Algorithm (Dynamic Programming)



Let $d_v(t)$:= cost of shortest path from v to t

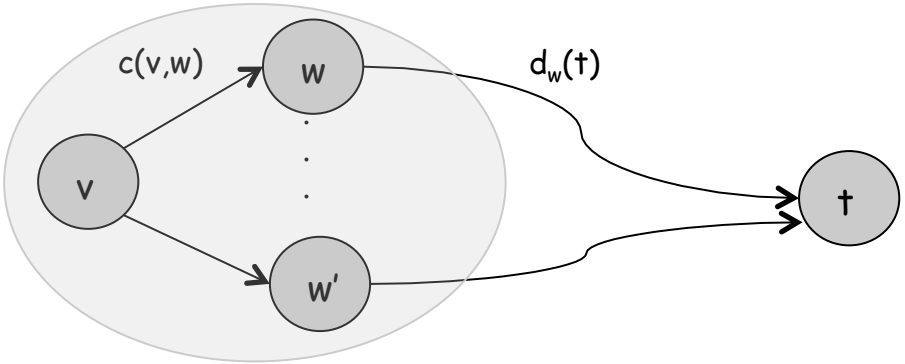
Then $d_v(t) = \min_w \{c(v,w) + d_w(t)\}$

cost from neighbor w to destination t

cost from v to neighbor w

min taken over all neighbors w of v

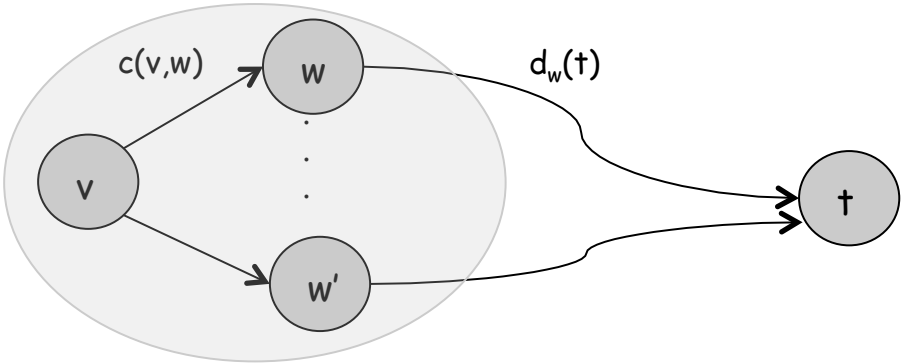
Node v's operation



- v's local view
  - v knows the costs c(v,w) to all his neighbors w
  - v exchanges updates with this neighbors about their $d_w(t)$ (and his $d_v(t)$)
- v locally computes:
  - $d_v(t) = \min_w \{c(v,w) + d_w(t)\}$
- Asynchronous and distributed operation

- v's forwarding table entry
  - $(t, d_v(t), w^*)$
  - $d_v(t)$: length of shortest path from v to t.
  - w^*: the next hop after v, on the shortest path to t

Single destination "t" can be implicit



Let $d_v(t)$:= cost of shortest path from v to t

Then $d_v(t) = \min_w \{c(v,w) + d_w(t)\}$

cost from neighbor w to destination t

cost from v to neighbor w

min taken over all neighbors w of v

Shortest path from v ~~to t~~

- Each node v maintains local information
- 1- d(v): length of shortest path from v to t.
- 2- w: the next hop after v, on the shortest path to t.

- Each node makes local decisions

$$d(v) = \min_{w:(v,w)\in E} \{ d(w) + c_{vw} \}$$

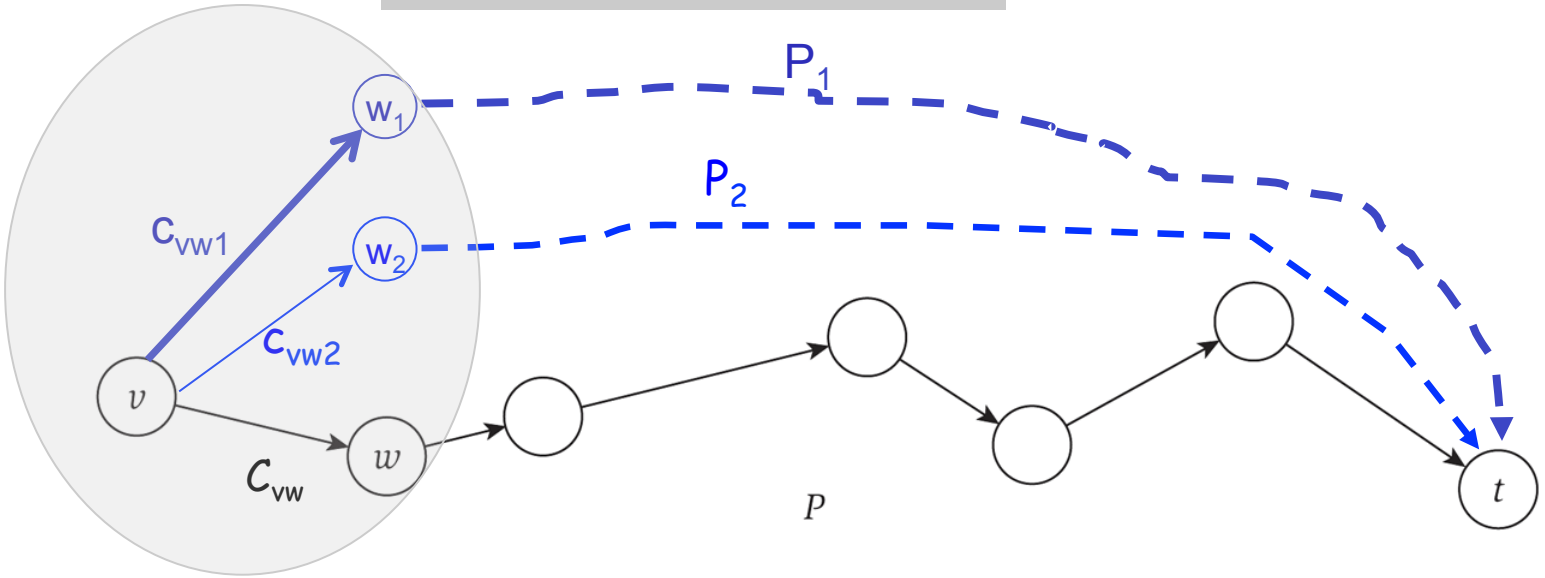
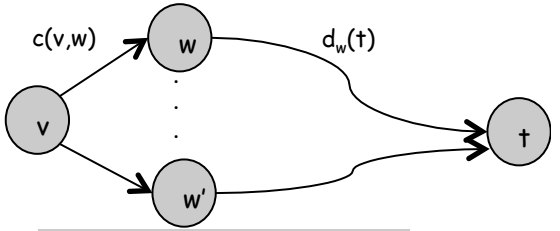


Figure 6.22 The minimum-cost path P from v to t using at most i edges.

Implementation 1 : Synchronous, Pull-based



$$d(v) = \min_{w:(v,w)\in E} \{ d(w) + c_{vw} \}$$

Def: d(i, v) = length of shortest v-t path P using at most i edges.

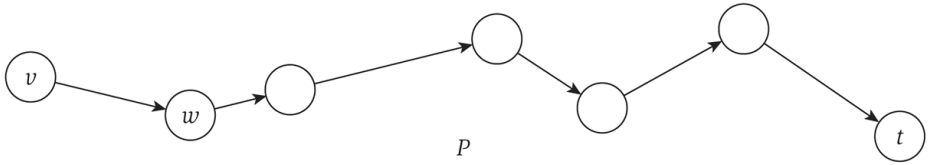


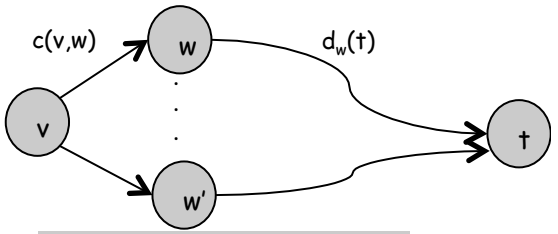
Figure 6.22 The minimum-cost path P from v to t using at most i edges.

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      d[0, v] ← ∞
   d[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V (in any order)
         d[i, v] ← d[i-1, v]
      foreach edge (v, w) ∈ E
         d[i, v] ← min {d[i, v], d[i-1, w] + c_vw }
}
```

Implementation 2: Synchronous, Push-based

Rule: time is slotted, nodes exchange d with neighbors




```
Push-Based-Shortest-Path(G, s, t) {
   foreach node v ∈ V {
      d[v] ← ∞
      successor[v] ← ϕ
   }

   d[t] = 0
   for i= 1 to n-1 {
      foreach node w ∈ V {
      if (d[w] has been updated in previous iteration) {
         foreach node v such that (v, w) ∈ E {
            if (d[v] > d[w] + c_vw) {
               d[v] ← d[w] + c_vw
               successor[v] ← w
            }
         }
      }
      If no d[w] value changed in iteration i, stop.
   }
}
```

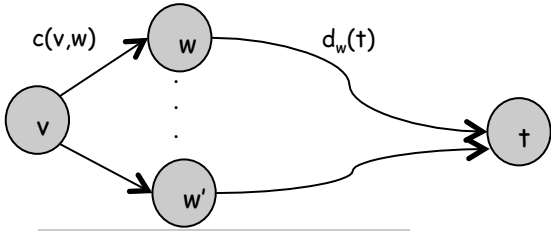
Rounds: synchronous implementation

w pushes updates (instead of v pulling)

Local +distributed update

can stop in <n-1 rounds

Implementation 3: Distributed, Asynchronous, Push-based



$$d(v) = \min_{w:(v,w)\in E} \{ d(w) + c_{vw} \}$$

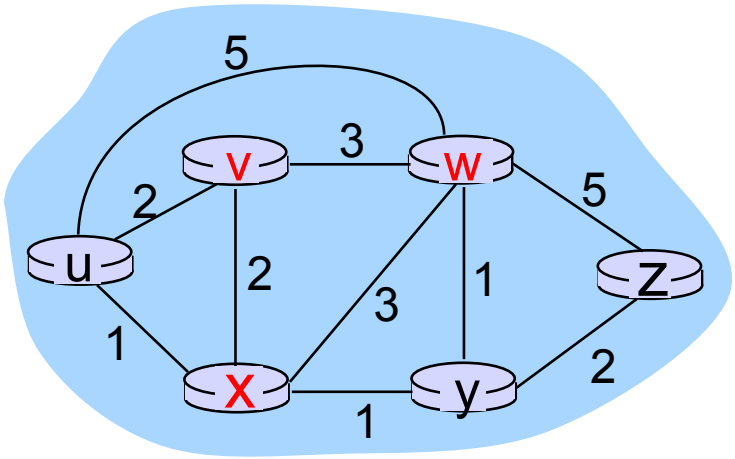
Rule: Every time a node w updates its d[w], it becomes "active" and notifies all its upstream neighbors:

```
Asynchronous-Shortest-Path(G, s, t) {
    foreach node v ∈ V {
        d[v] ← ∞; successor[v] ← ϕ; Active[v]=FALSE;
    }
    d[t]=0; successor[v]=t; Active[t]=TRUE;

    while for there exists an active node{
        choose active node w \* in any order *\
        foreach node v such that (v, w) ∈ E \*in any order*\{
              if (d[v] > d[w] + c_vw) {
                 d[v] ← d[w] + c_vw;
                 successor[v] ← w;
                   active[v]=TRUE;
              }
        }
        w becomes inactive
     }
}
```

Proof: One can prove [omitted] that the algorithm converges to the correct values of shortest paths, under mild assumptions.

Bellman-Ford example

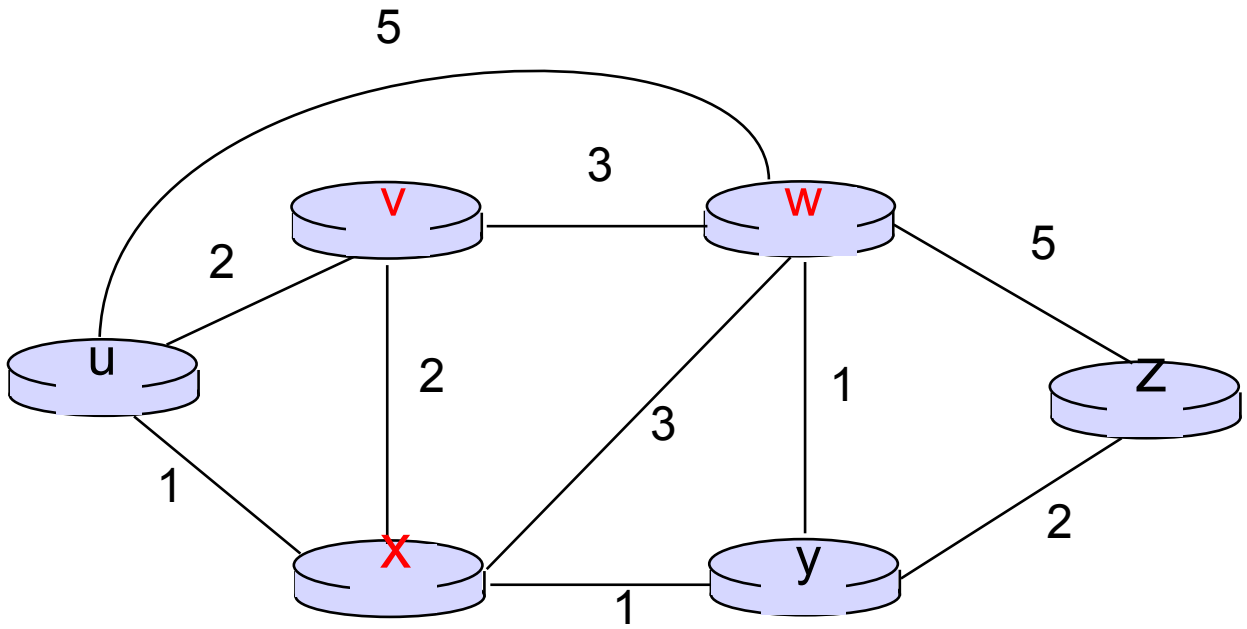


clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

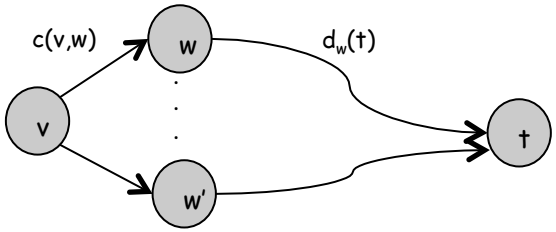
B-F equation says:

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

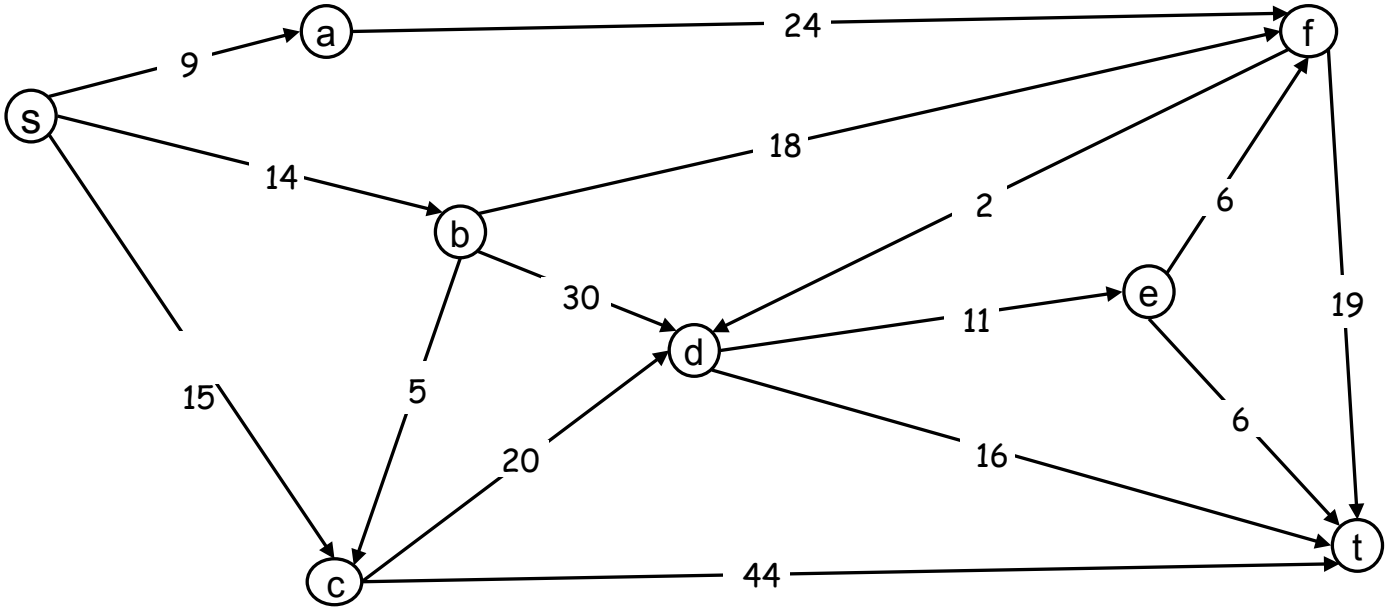
node achieving minimum is next hop in shortest path, used in forwarding table


5
3
v
w
2
5
u
2
1
z
3
1
2
x
y
1

Example
on the board



$$d(v) = \min_{w:(v,w)\in E} \{ d(w) + c_{vw} \}$$



Distance vector

- $D_v(t)$ = estimate of least cost from v to t
  - v maintains distance **vector** $\mathbf{d}_v = [d_v(t): t \in N]$ to all possible destinations $t \in N$.

- Node v:
  - knows cost to each neighbor w: c(v,w)
  - routing table maintains entries $(t, d_v(t), w^*)$, where:
    - t is any possible destination $t \in N$
    - the distance from v to t is $d_v(t)$
    - the next hop on the shortest path from v towards t is w^*
  - performs n separate computations, one for each potential destination

Distance vector algorithm

key idea:

- from time-to-time, each node v sends its own distance vector estimate to neighbors
- When v receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_v(t) \leftarrow min_w\{c(v,t) + d_w(t)\} \quad \text{for each node } t \in N$$

- under minor, natural conditions, the estimate $d_v(t)$ *converge to the actual least cost* $d_v(t)$

Distance vector algorithm

iterative, synchronous:

each node:

In each iteration

Exchange DV with all neighbors

Recompute estimates to all destinations

Until the estimate does not change

Example of Synchronous Execution: **distance to x**

node x table

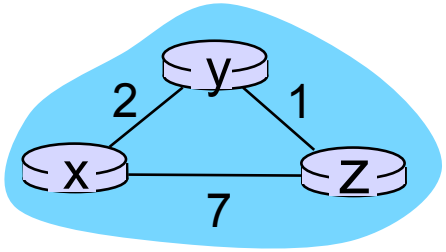
from \ cost to	x
x	0

node y table

from \ cost to	x
y	2

node z table

from \ cost to	x
z	7



time

Example of Synchronous Execution: **distance to x**

node x table

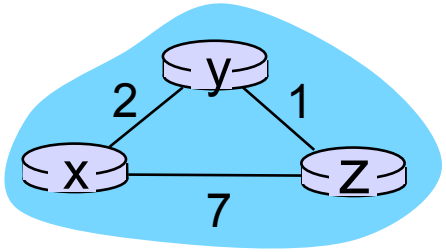
from \ cost to	x
x	0

node y table

from \ cost to	x
y	2

node z table

from \ cost to	x
z	7



time

Example of Synchronous Execution: **distance to x**

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

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

node x table

from	cost to x
x	0

from	cost to x
x	0

node y table

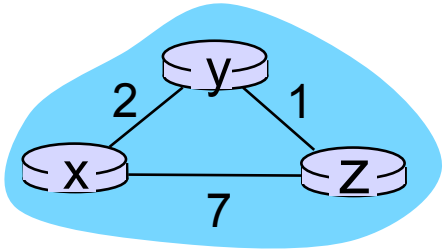
from	cost to x
y	2

from	cost to x
y	2

node z table

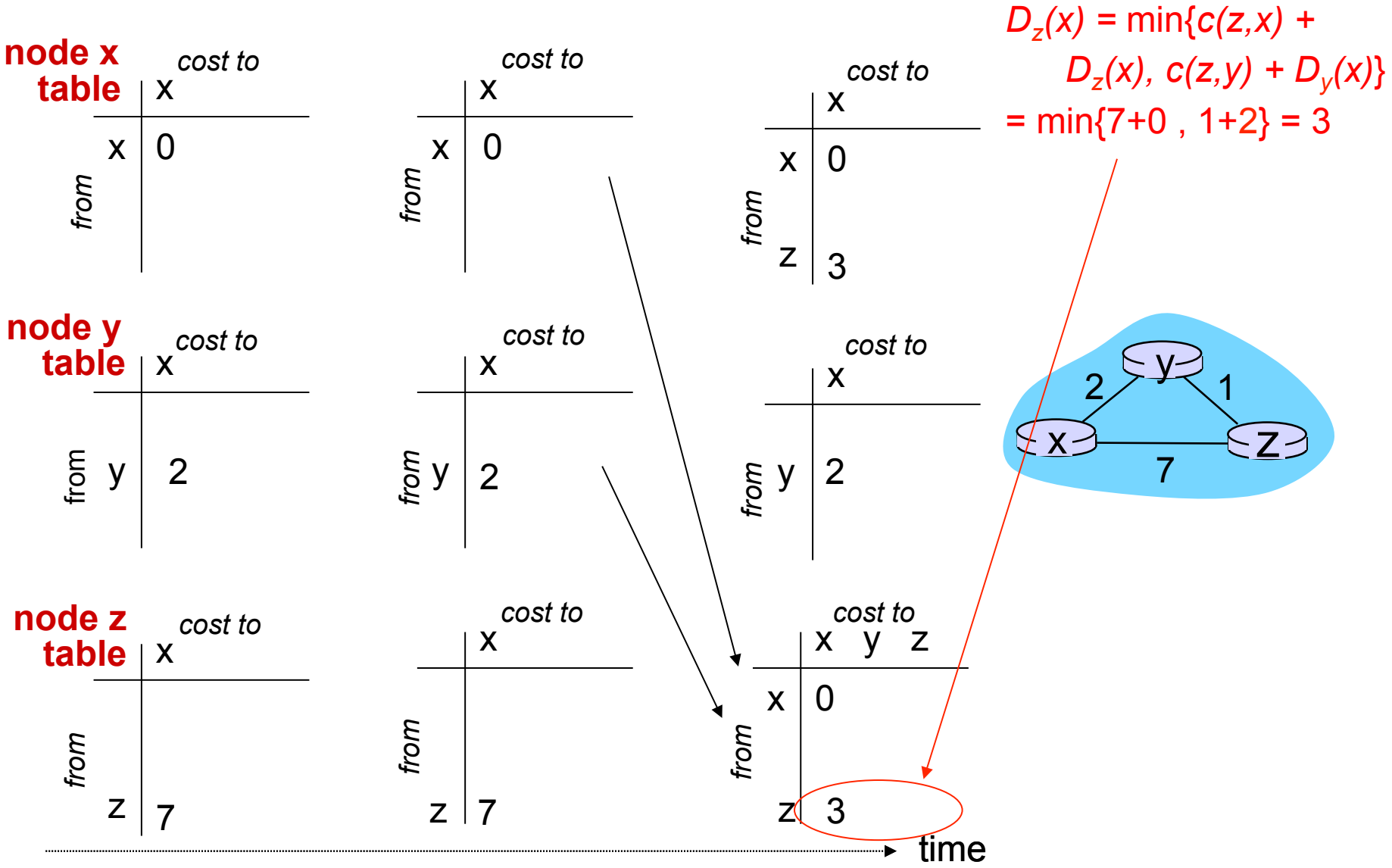
from	cost to x
z	7

from	cost to x
z	7



time

Example of Synchronous Execution: **distance to x**



Example of Synchronous Execution: **all dest**

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

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

node x table

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

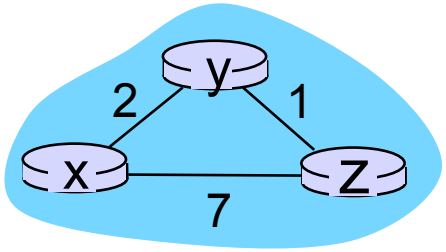
from	cost to x	cost to y	cost to z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

from	cost to x	cost to y	cost to z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

from	cost to x	cost to y	cost to z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

Example of Synchronous Execution: **all dest**

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

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

node x table

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

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

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

node y table

from	cost to x	cost to y	cost to z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

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

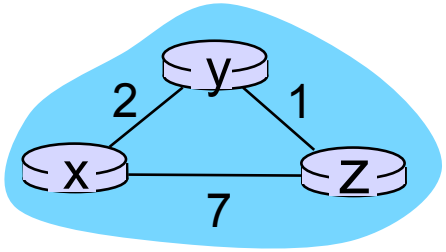
from	cost to x	cost to y	cost to z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

from	cost to x	cost to y	cost to z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

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

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



time

Distance vector algorithm

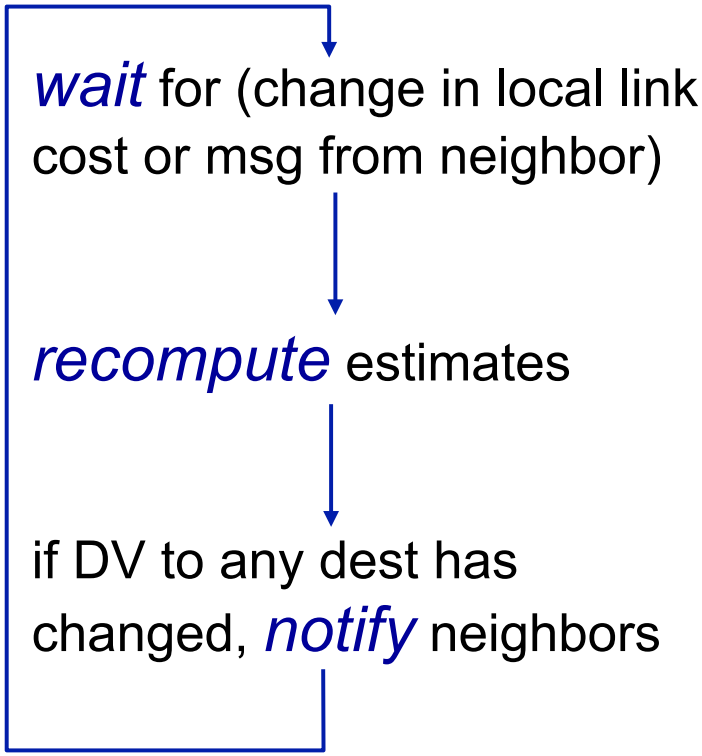
iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbor

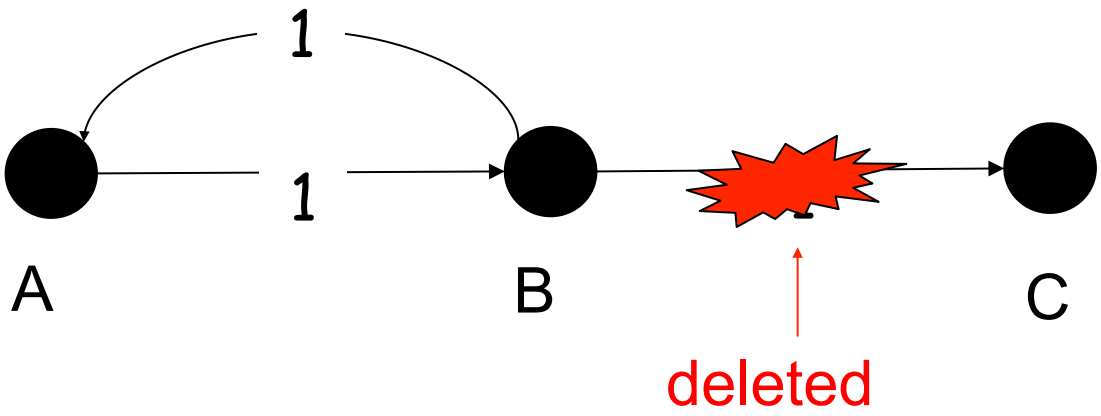
distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

each node:



"Counting to Infinity" Example



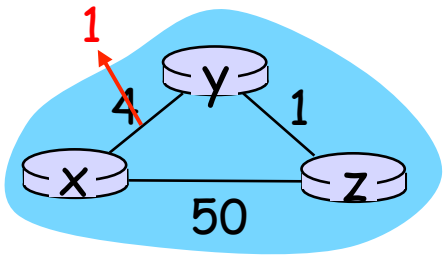
- Originally:
  - B to C: via C, cost 1,
  - A to C: via B, cost 2
- After link failure:
  - B to C: via A, cost 3
  - A to C: via B, cost 4
  - B to C: via A, cost 5
  - A to C: via B, cost 6,
  - ++

- Fixes: (1) max hop count (2) poisonous reverse (3) keep path state.

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.

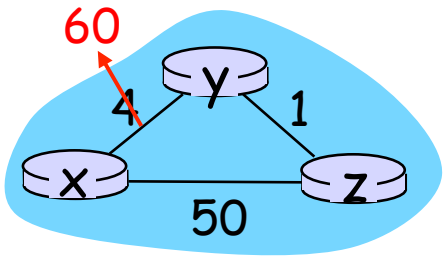
t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

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 convergence
  - y to x: 4 via x
  - z to x: 5 via y
  - change of c(x,y): 4→60
  - y to x: 6 via z
  - z to x: 7 via y
  - …
  - Counting to 50



Solutions:

- *Poisoned Reverse:* if Z routes through Y to get to X, then Z advertises to Y that 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?
  - No, if loops involve more than 2 nodes.
- Other fixes: (1) max hop count (2) keep path state.

Comparison of LS and DV algorithms

- ***In LS:*** each node broadcasts to all other nodes information about its neighborhood (neighbors+link costs)
- ***In DV:*** each node talks only to its neighbors and tells them its estimates of path costs to all other destinations

message complexity

- with n nodes, E links
- ***LS:*** O(nE) msgs sent
- ***DV:*** exchange between neighbors
  - convergence time varies

speed of convergence

- ***LS:***
  - Naïve: $O(n^2)$
  - Efficient implementation: O(nlogE)
  - may have oscillations
- ***DV:*** convergence time varies
  - Efficient implementation: O(mn) time, O(m+n) memory
  - may be routing loops
  - count-to-infinity problem

robustness: what happens if router malfunctions?

LS:

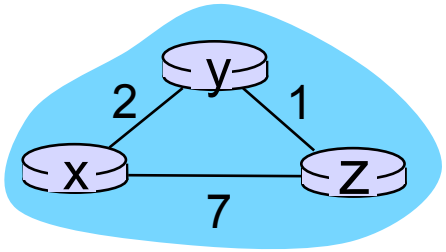
- node can advertise incorrect *link* cost
- each node computes only its *own* table

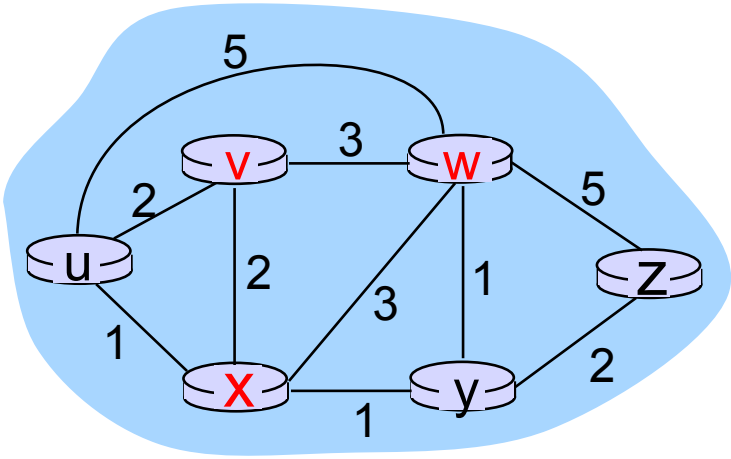
DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

Examples of asynchronous execution

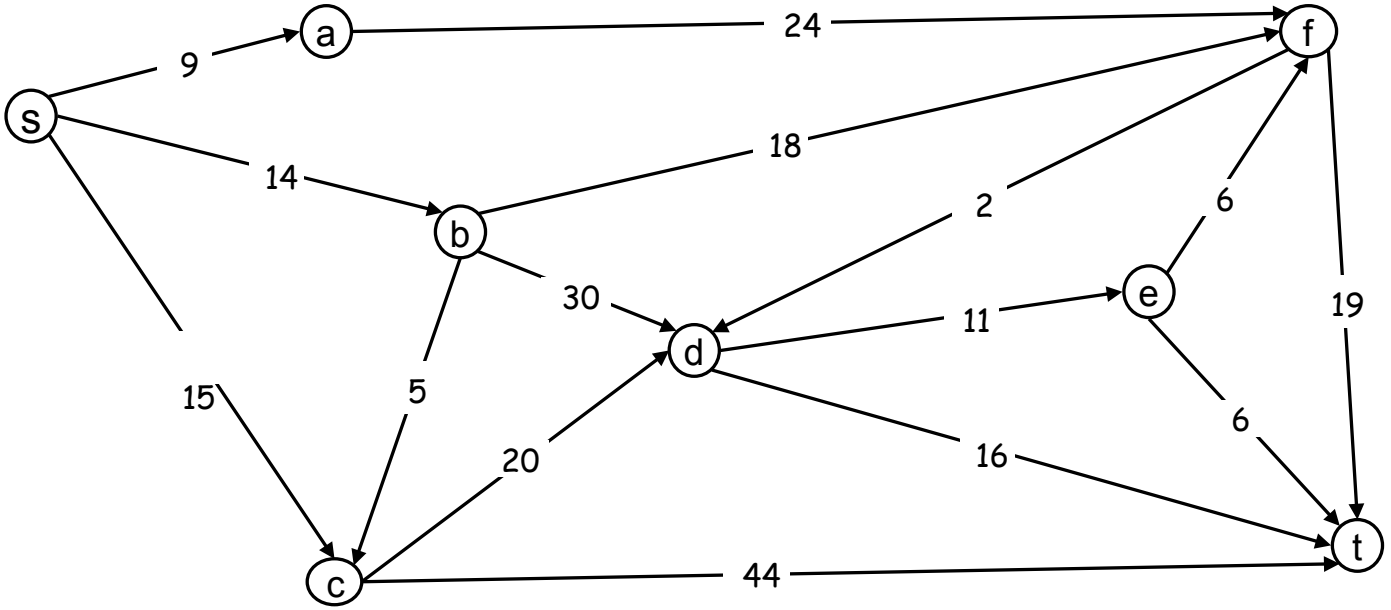
(on the board)





Example for your own practice/discussion

- Find shortest paths from s to all other nodes, using Dijkstra
- Find shortest paths from all nodes to T, using Bellman-Ford
  - asynchronous execution
  - consider that a link fails or changes cost. What happens?
- Also interactive exercises



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

Hierarchical routing

our routing study thus far - idealization

- all routers identical
- network "flat"

… *not* true in practice

scale: with 600 million destinations:

- 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

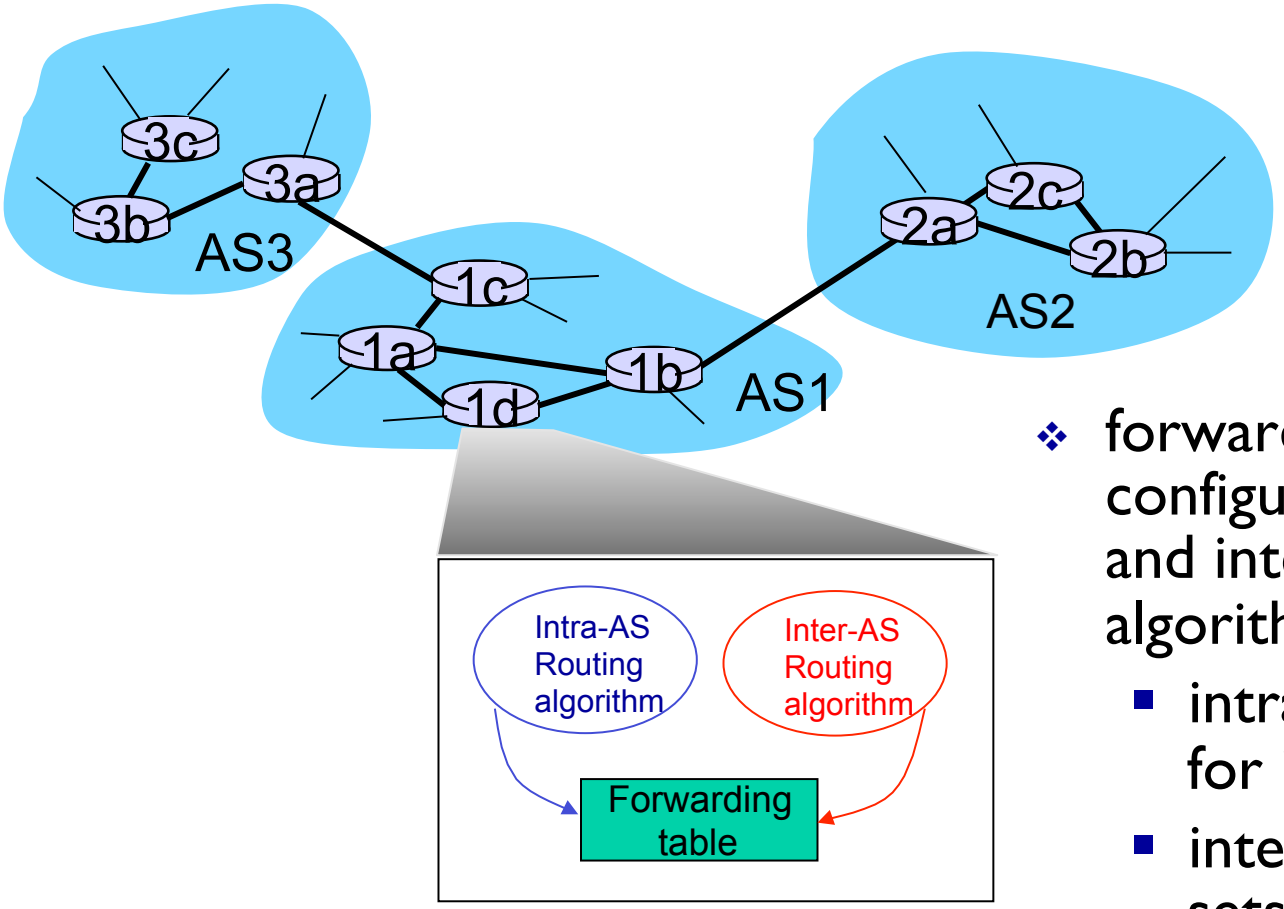
Hierarchical routing

- 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



- 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

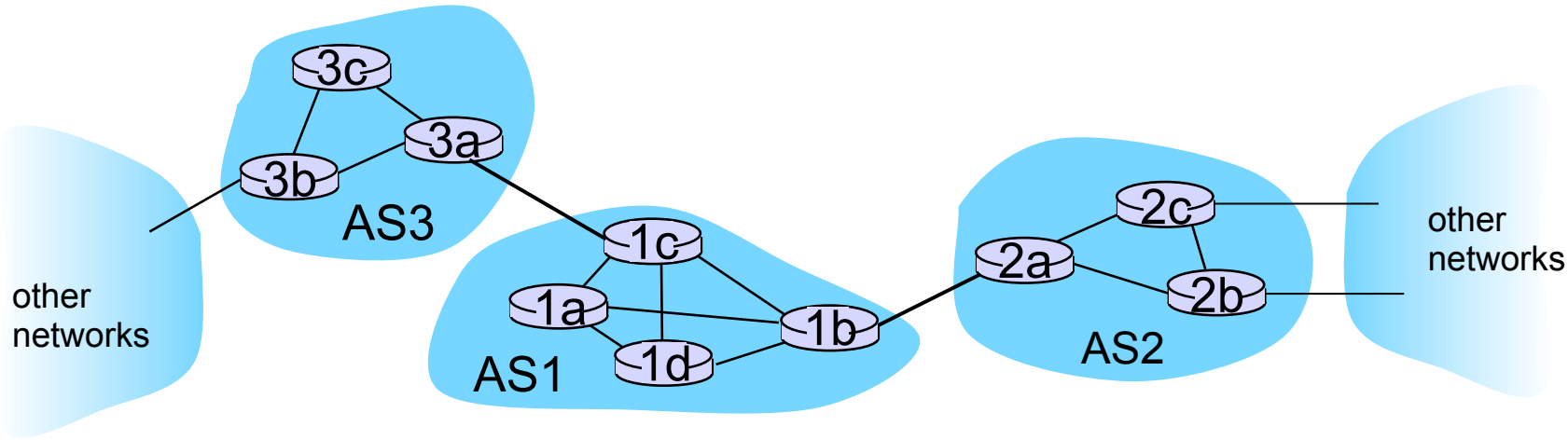
Inter-AS tasks

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

AS1 must:

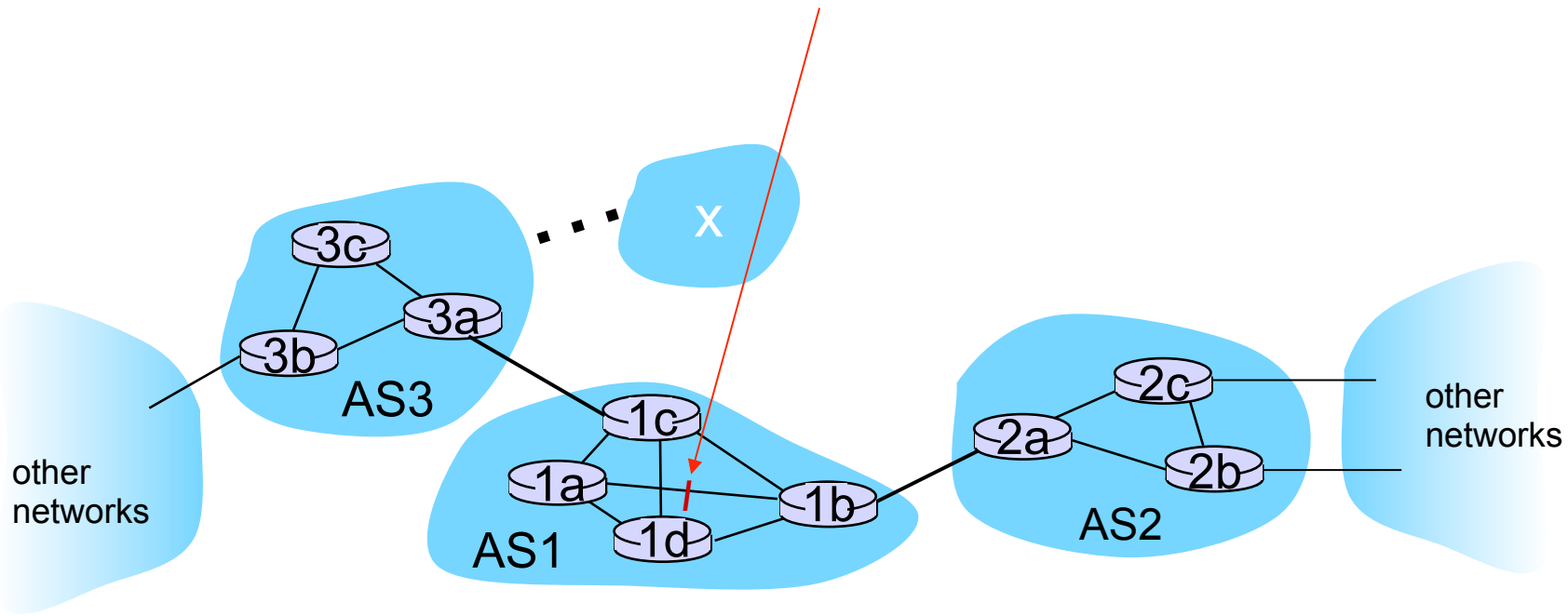
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



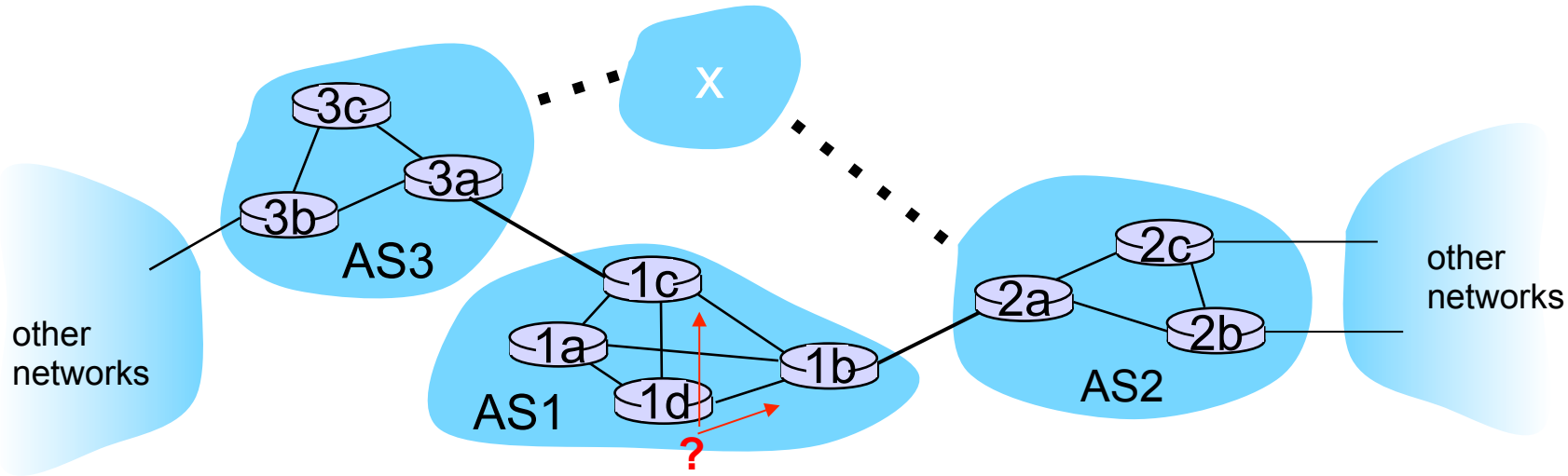
Example: setting forwarding table in router 1d

- 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!
- *hot potato routing: send* packet towards closest of two routers.



Summary: routing outside the AS