Network Layer
Part 7

Mark Allman
mallman@case.edu

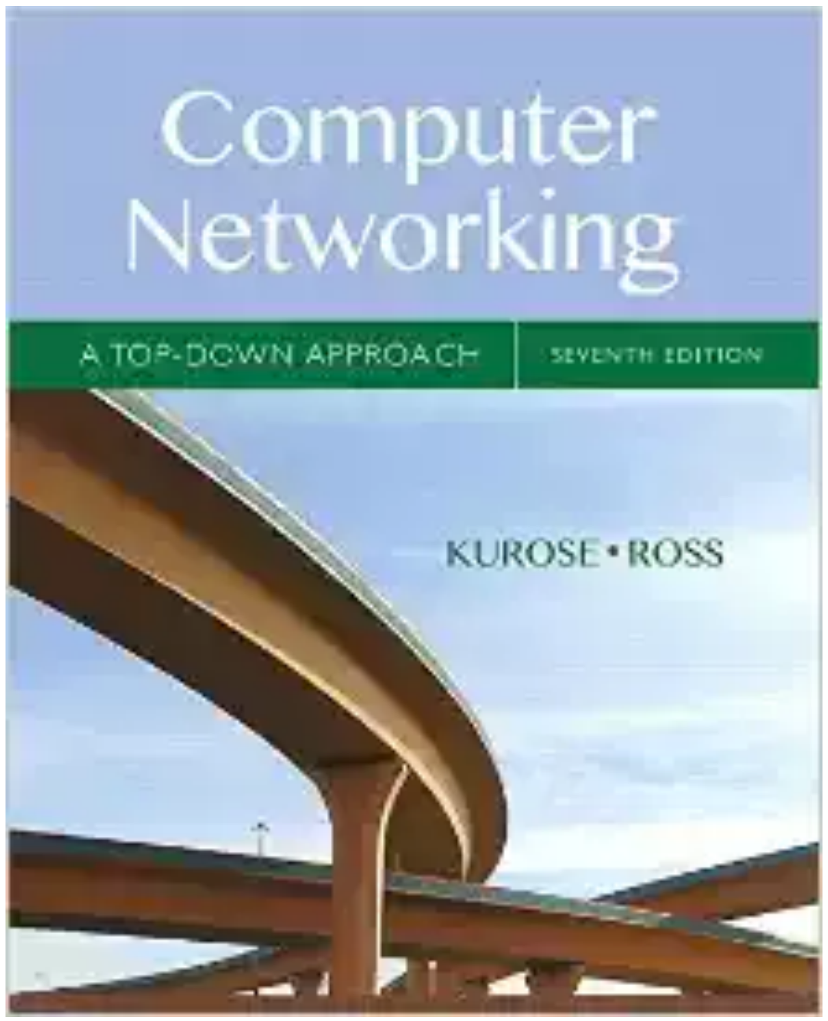
EECS 325/425
Fall 2018

*"I was working in the lab late one night
When my eyes beheld an eerie sight"*

These slides are more-or-less directly from the slide set developed by Jim Kurose and Keith Ross for their book "Computer Networking: A Top Down Approach, 5th edition".

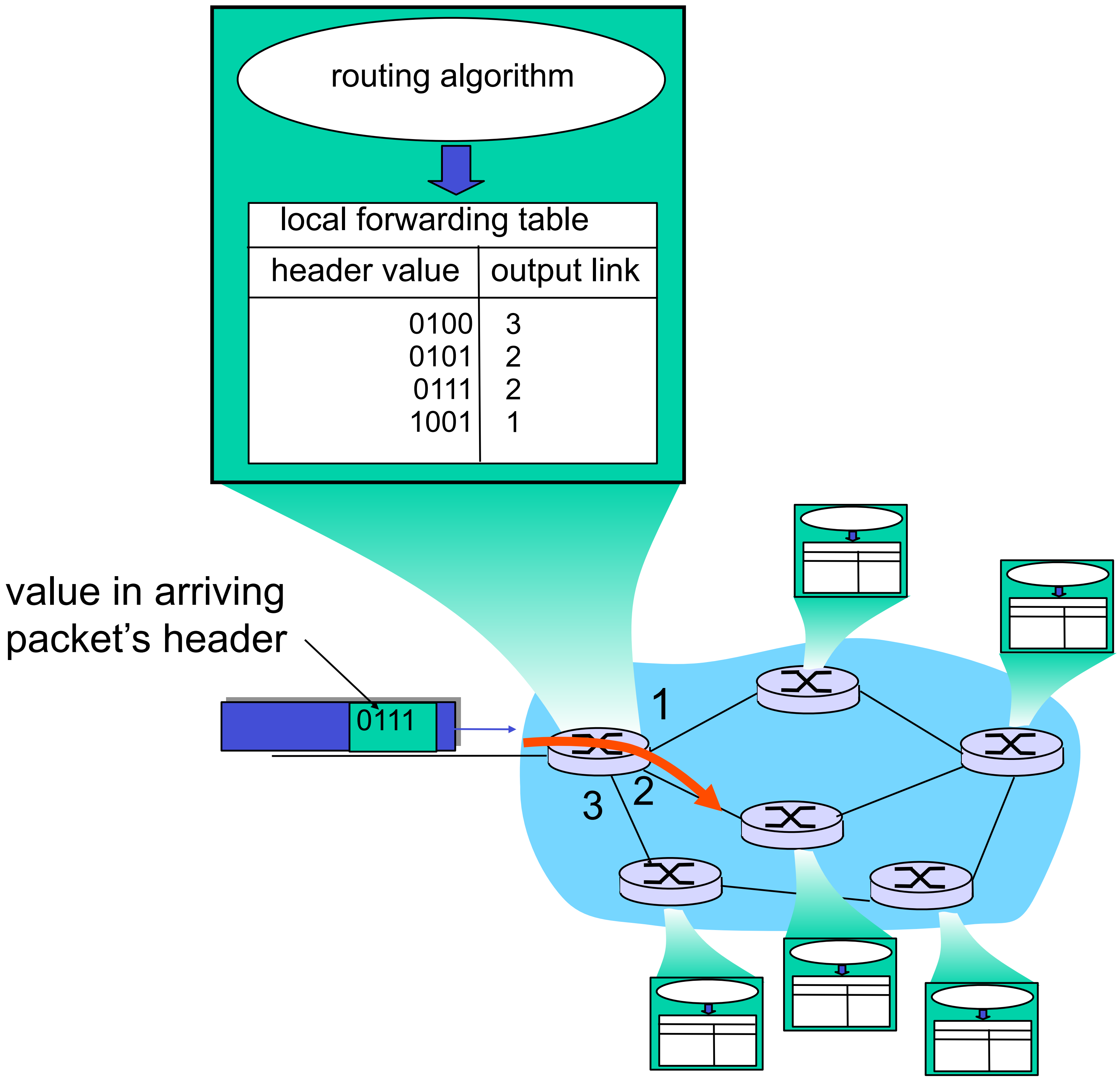
The slides have been lightly adapted for Mark Allman's EECS 325/425 Computer Networks class at Case Western Reserve University.

Reading Along ...

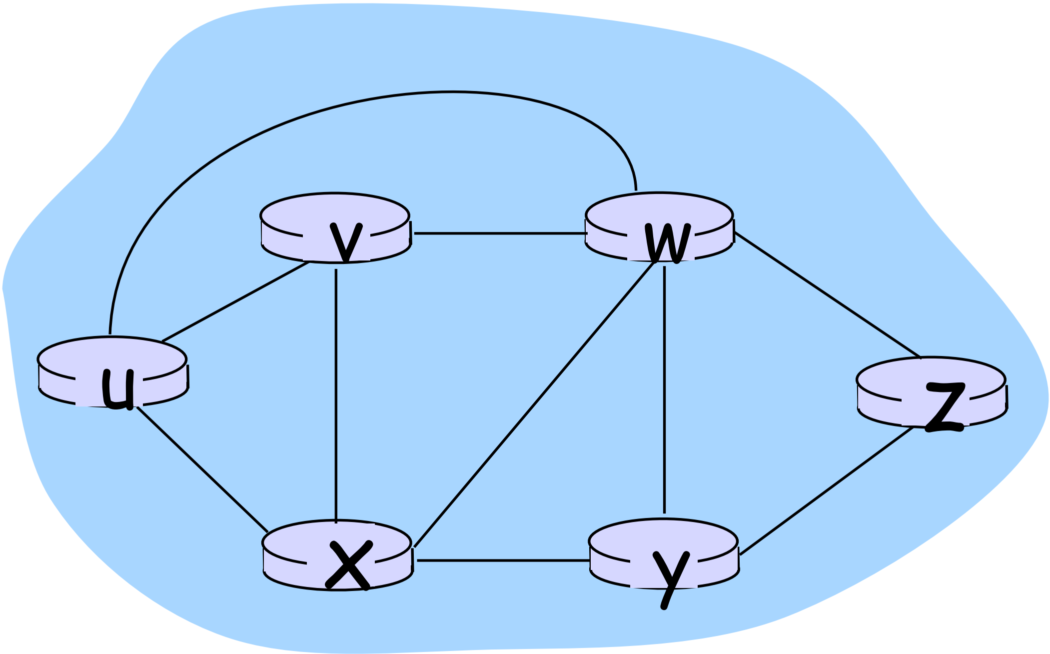


- Network layer is chapters 4 & 5
  - Routing Algorithms

Interplay between routing, forwarding



Graph abstraction



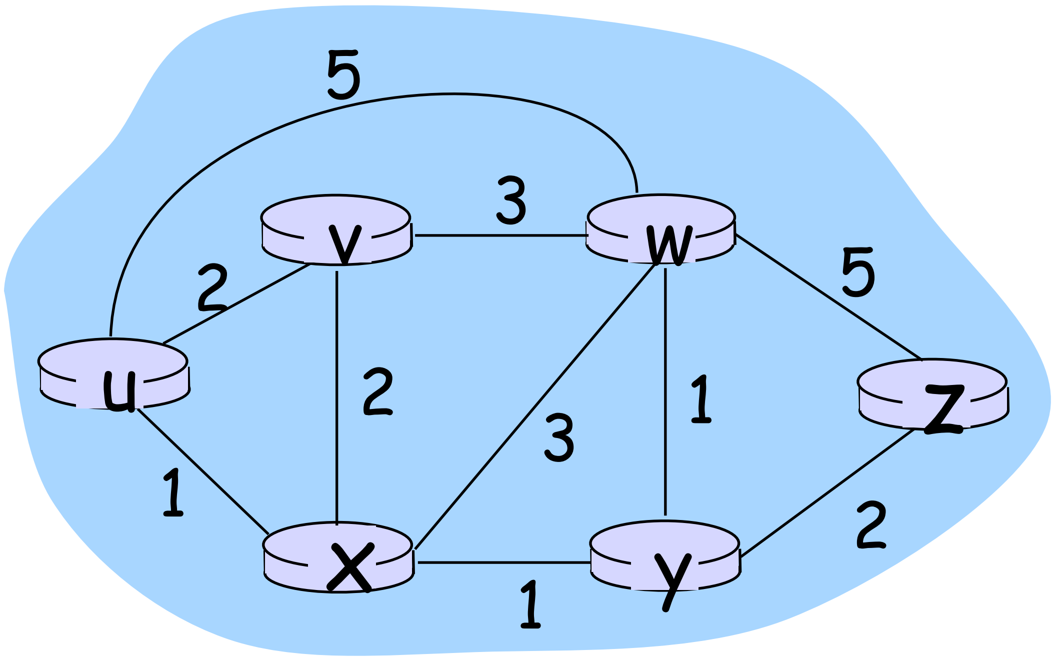
Graph: G = (N,E)

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

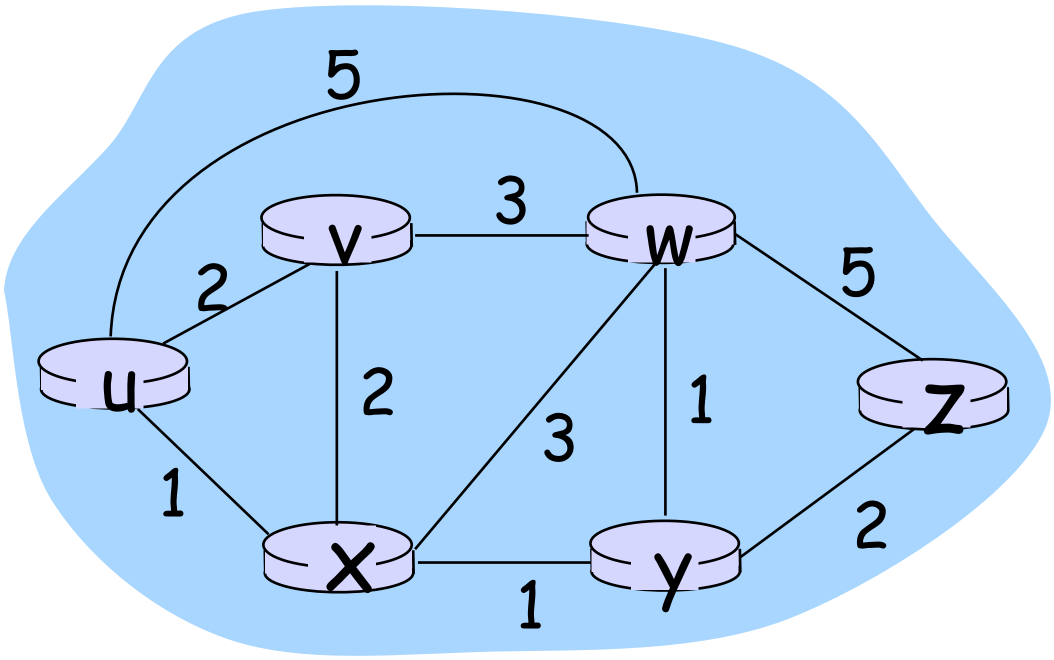
E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

Graph abstraction

Graph abstraction

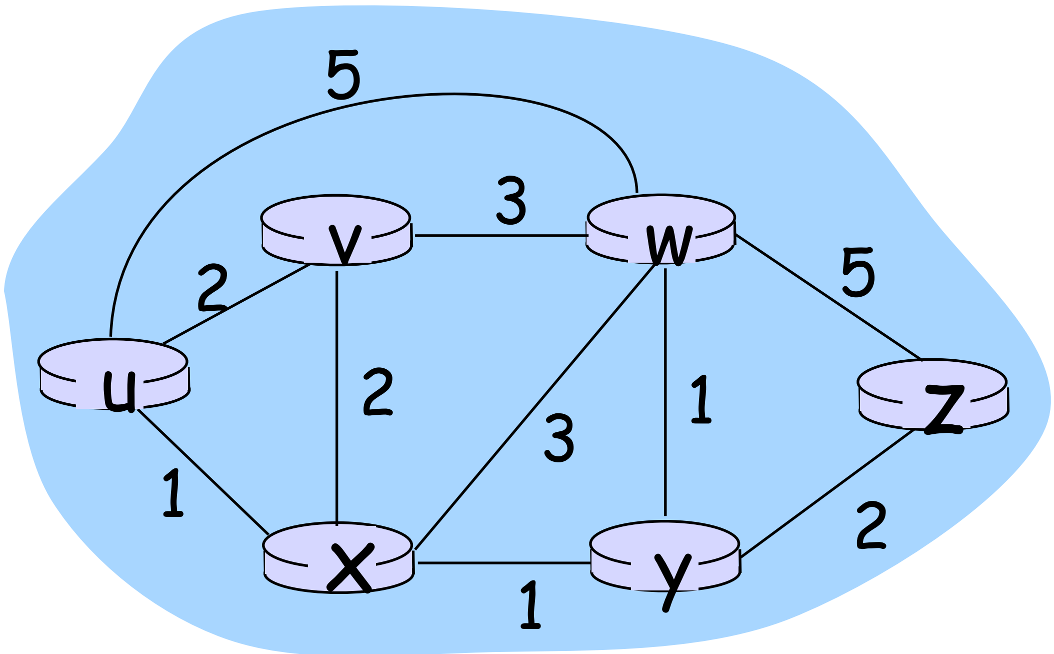


Graph abstraction



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

Graph abstraction

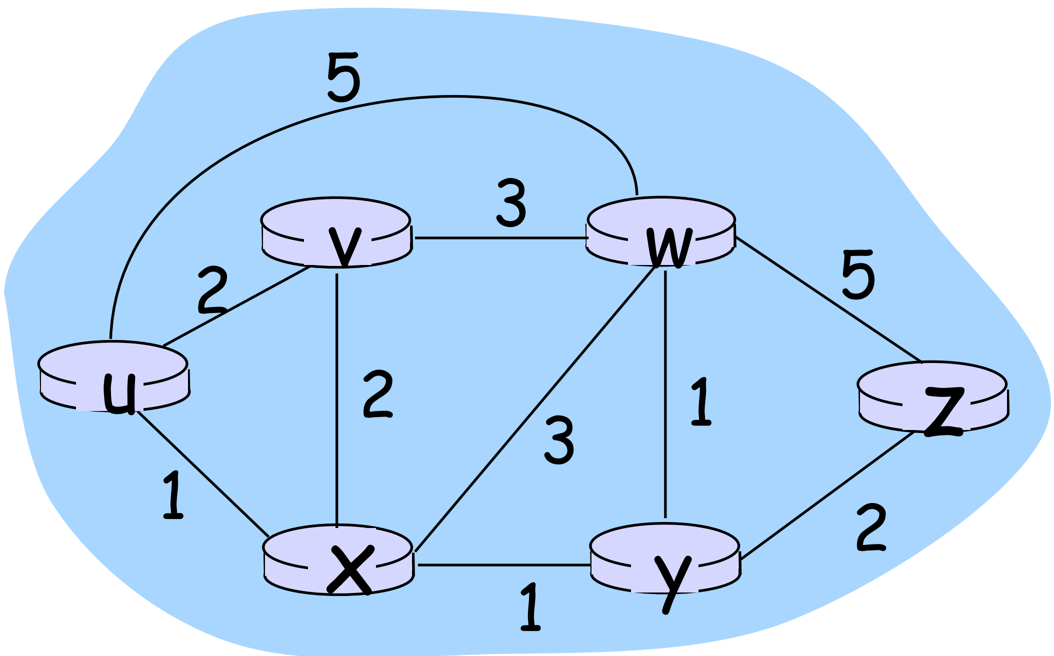


- c(x,x') = cost of link (x,x')

  - e.g., c(w,z) = 5

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

Graph abstraction



- $c(x,x')$ = cost of link (x,x')

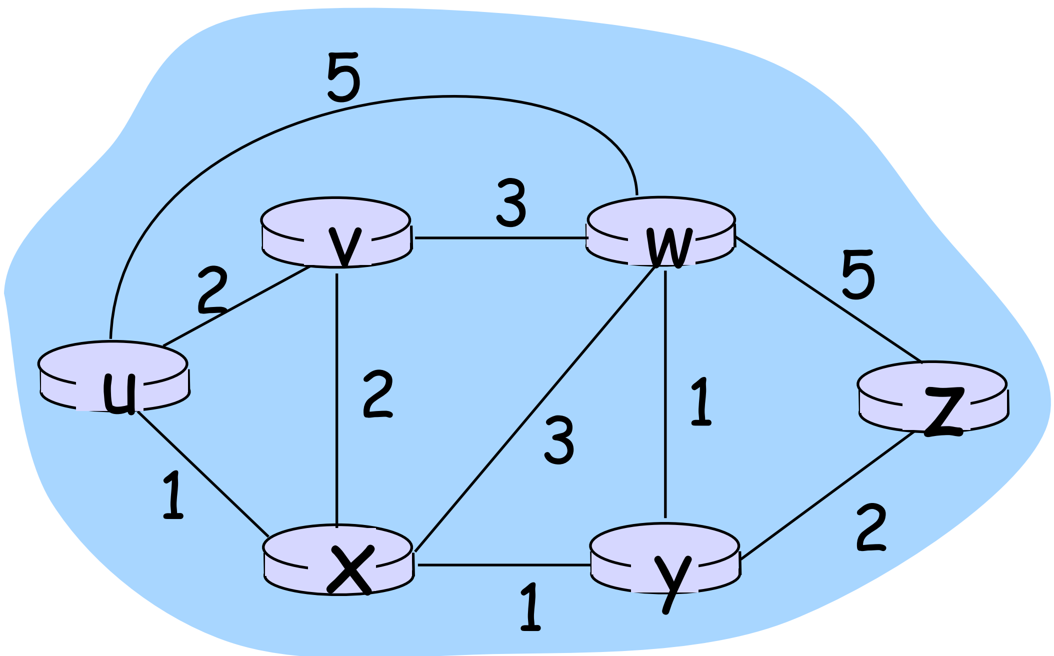
  - e.g., $c(w,z) = 5$

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

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

E.g., cost of $(u,w,z) = c(u,w) + c(w,z) = 5 + 5 = 10$

Graph abstraction



- $c(x,x')$ = cost of link (x,x')

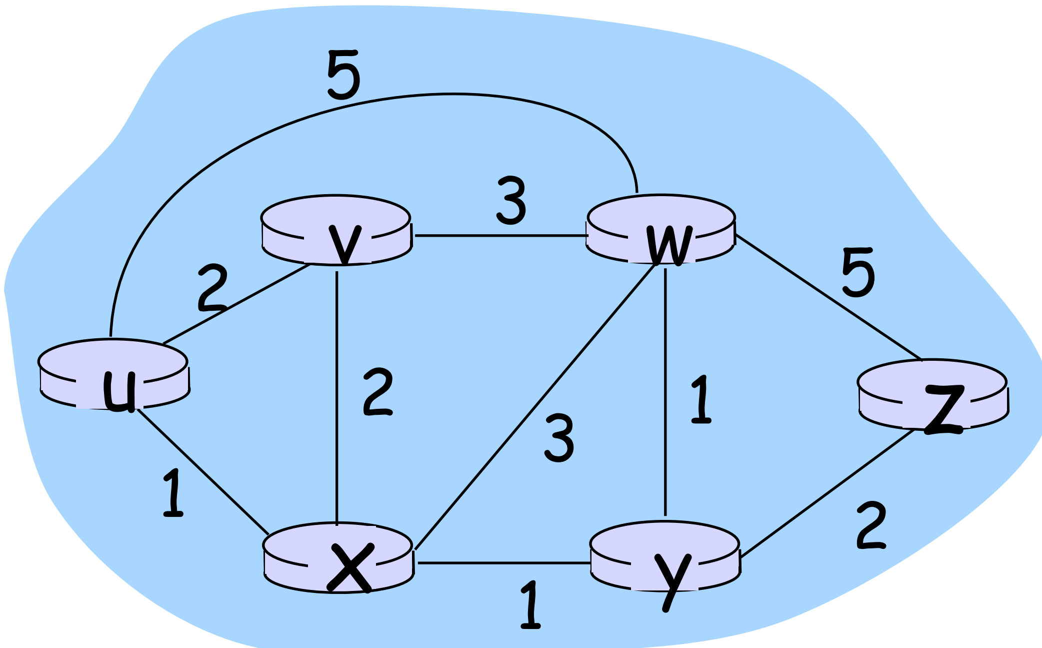
  - e.g., $c(w,z) = 5$

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

Cost of path $(x_1, x_2, x_3,\ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$
E.g., cost of $(u,w,z) = c(u,w) + c(w,z) = 5 + 5 = 10$

Question: What path to use between u and z ?

Graph abstraction



- $c(x,x')$ = 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

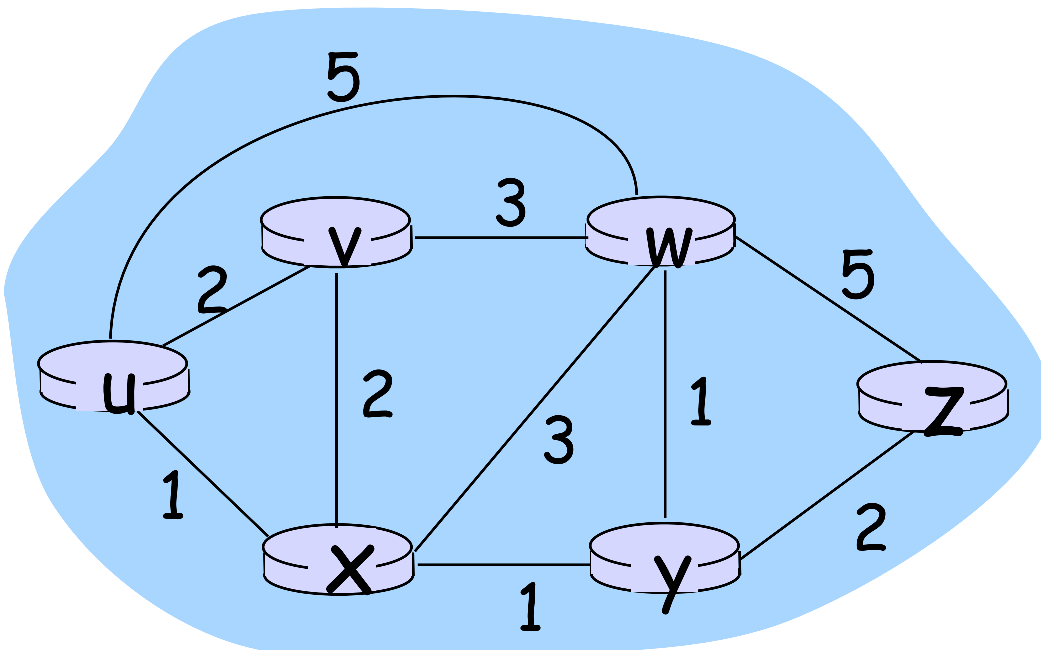
Cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

E.g., cost of $(u,w,z) = c(u,w) + c(w,z) = 5 + 5 = 10$

Question: What path to use between u and z ?

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

Graph abstraction



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

E.g., cost of $(u,w,z) = c(u,w) + c(w,z) = 5 + 5 = 10$

Question: What path to use between u and z ?

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

Routing algorithm: algorithm that finds least-cost path

Routing Algorithm classification

Routing Algorithm classification

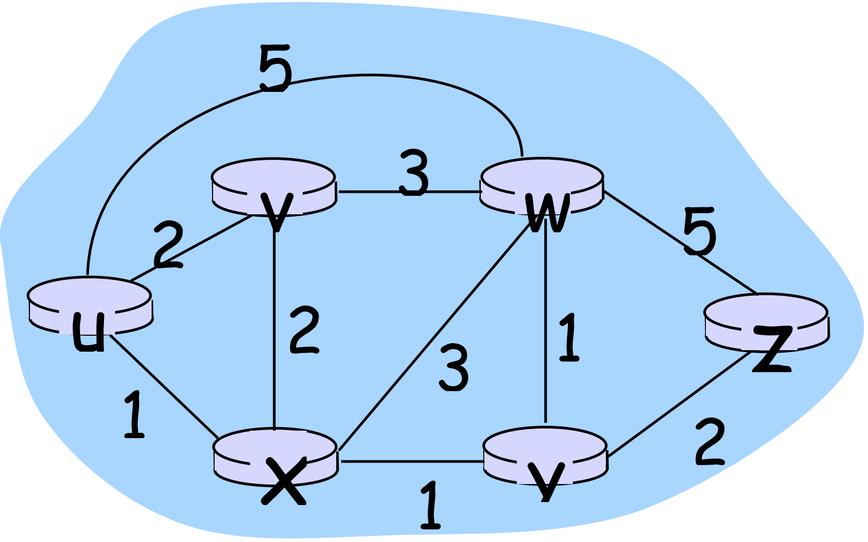
Global or decentralized information?

Routing Algorithm classification

Global or decentralized information?

Global:

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



Routing Algorithm classification

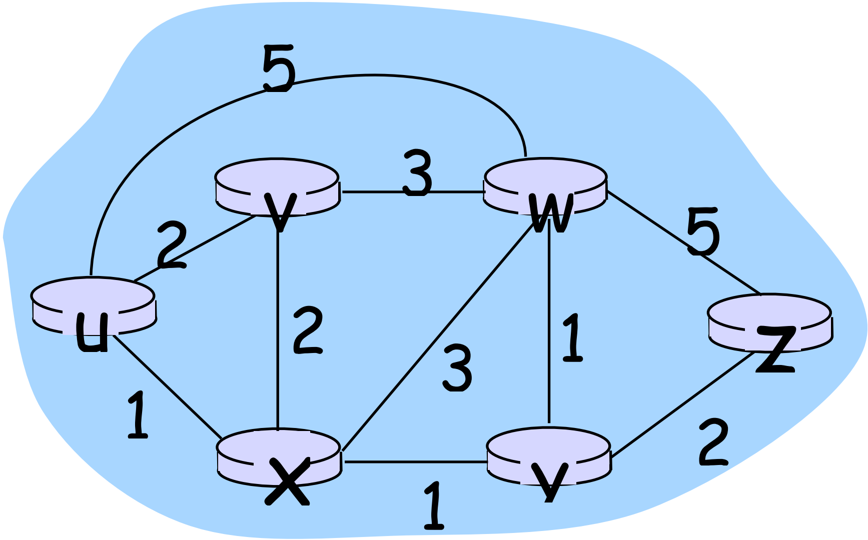
Global or decentralized information?

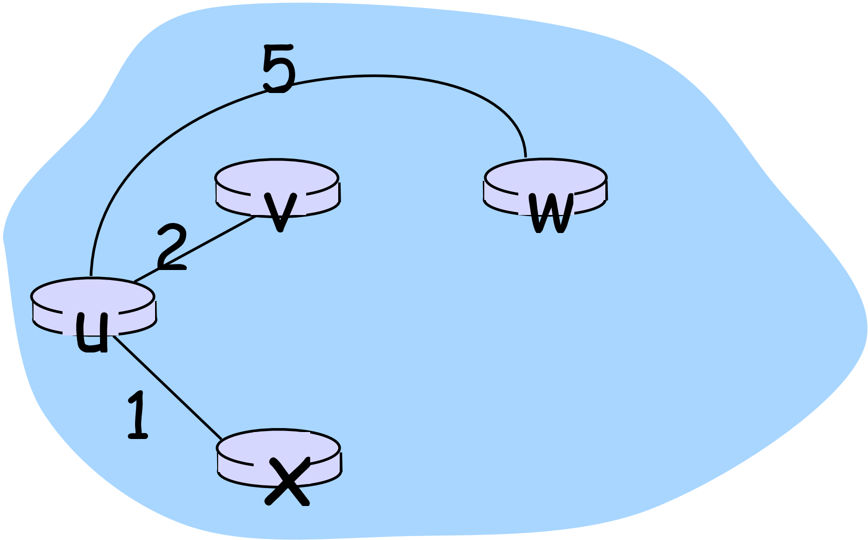
Global:

- 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





Routing Algorithm classification

Static or dynamic?

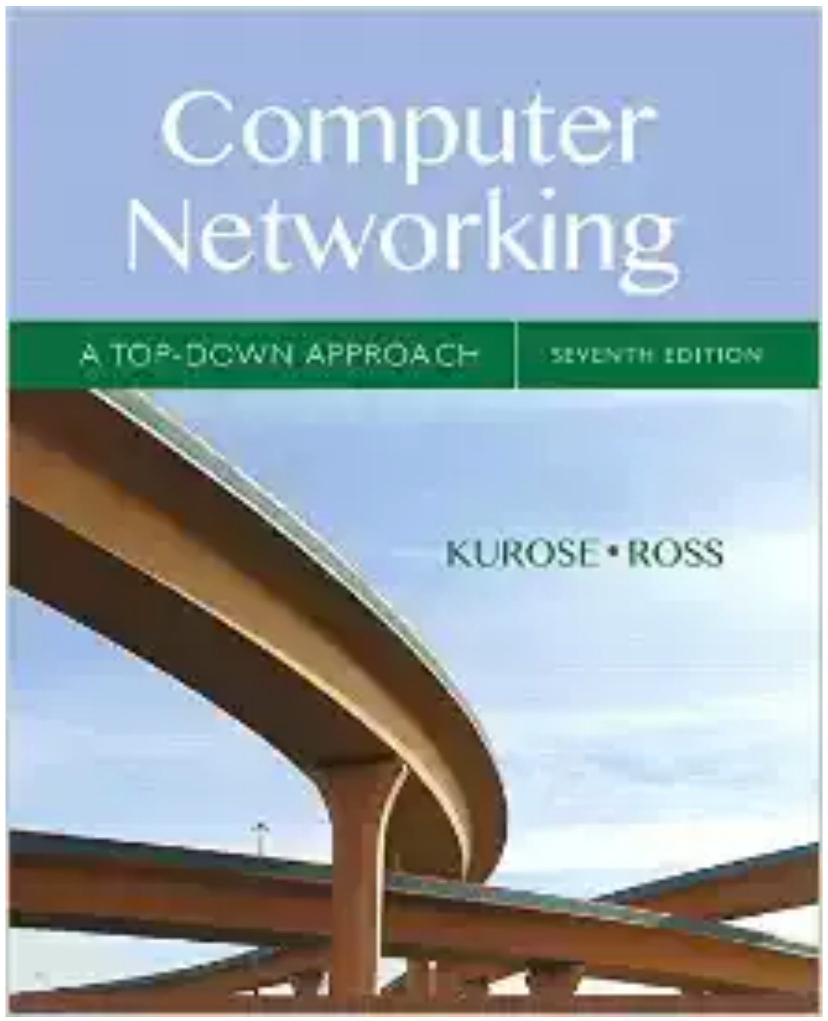
Static:

- routes change slowly over time

Dynamic:

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

Reading Along ...



- Network layer is chapters 4 & 5
  - Routing Algorithms Link state routing

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

Dijsktra's Algorithm

```
Initialization:
  N' = {a}
  for all nodes b
    if b adjacent to a
        then D(b) = c(a,b)
    else D(b) = ∞

 Loop
   find e not in N' such that D(e) is a minimum
    add e to N'
    update D(f) for all f adjacent to e and not in N' :
        D(f) = min( D(f), D(e) + c(e,b) )
    /* new cost to f is either old cost to f or known
     shortest path cost to e plus cost from e to f */
 until all nodes in N'
```

Dijkstra's algorithm, discussion

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(nlogn)

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

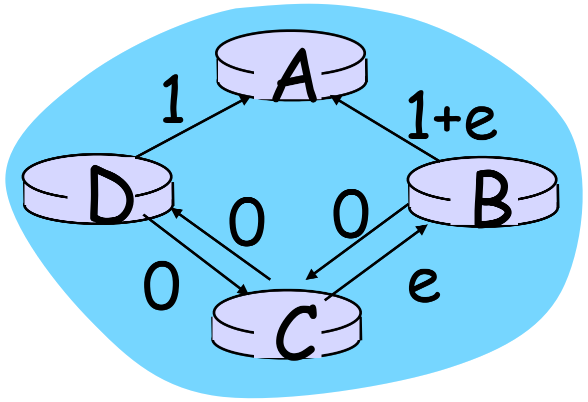
Dijkstra's algorithm, discussion

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(nlogn)

Oscillations possible:

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



initially

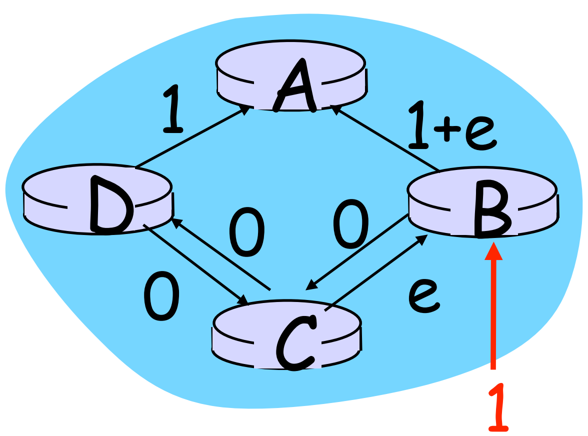
Dijkstra's algorithm, discussion

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(nlogn)

Oscillations possible:

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



initially

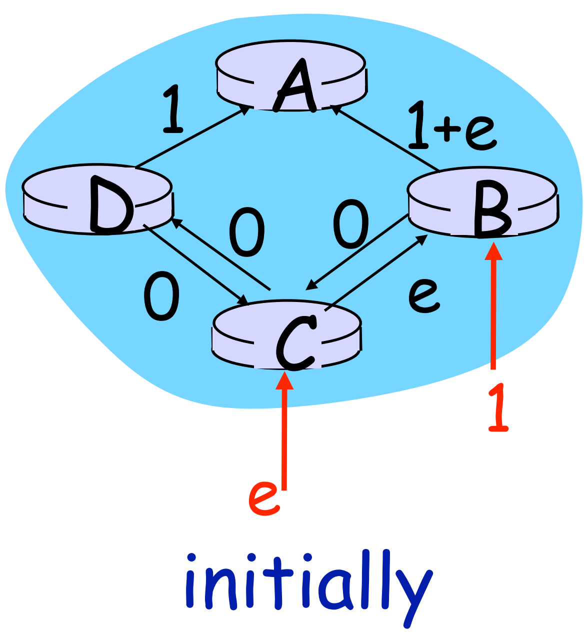
Dijkstra's algorithm, discussion

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



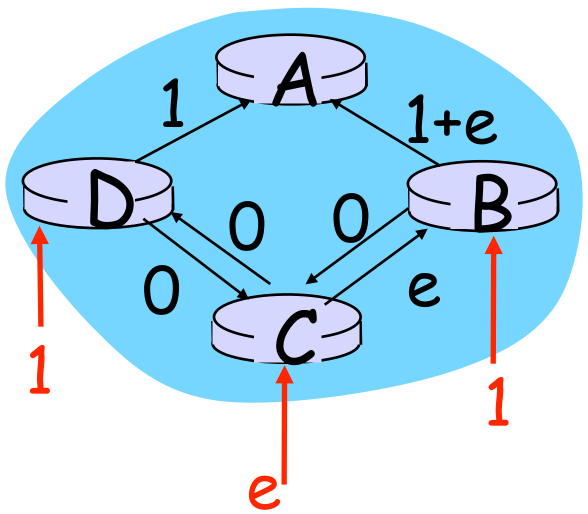
Dijkstra's algorithm, discussion

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

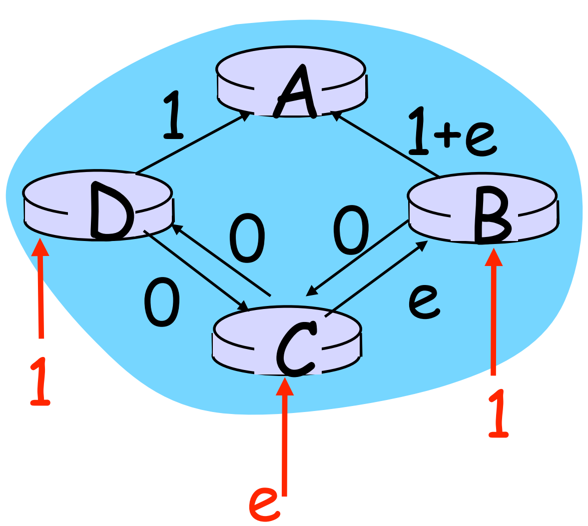
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

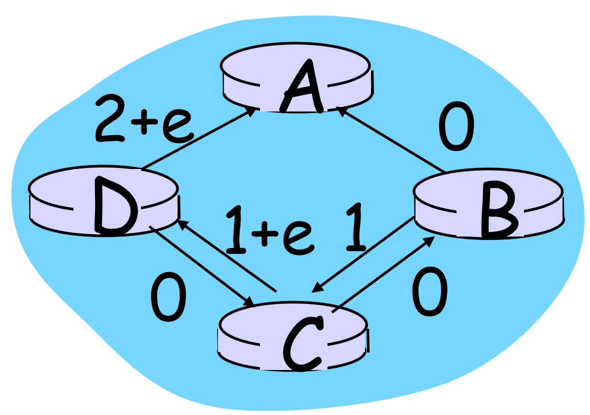
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

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



initially



... recompute routing

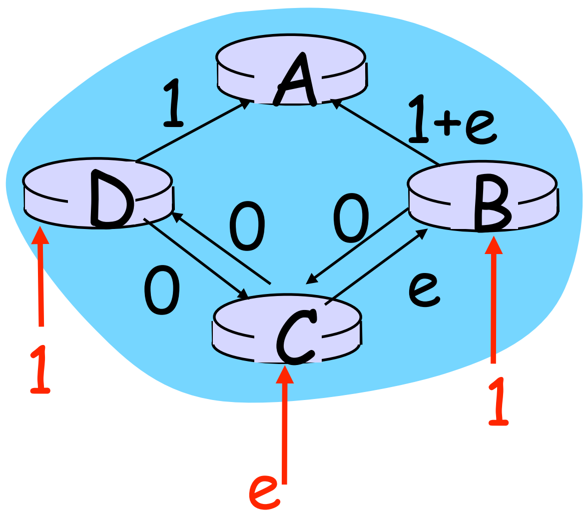
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

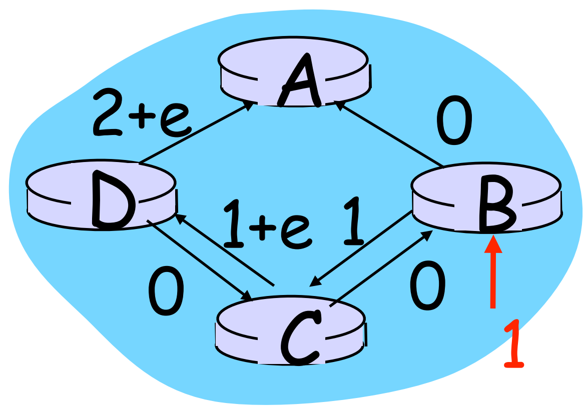
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

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



initially



… recompute routing

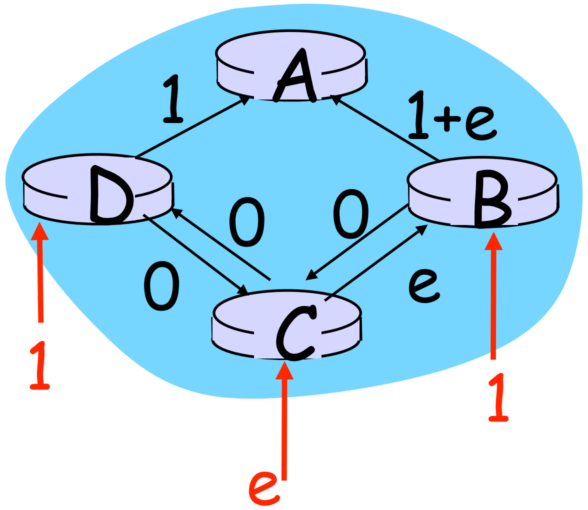
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

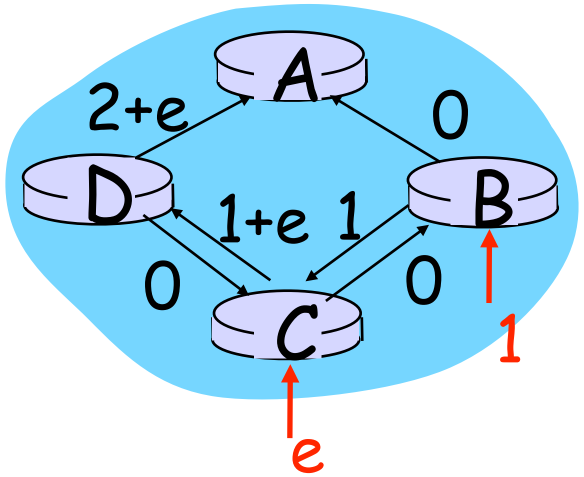
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

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



initially



… recompute routing

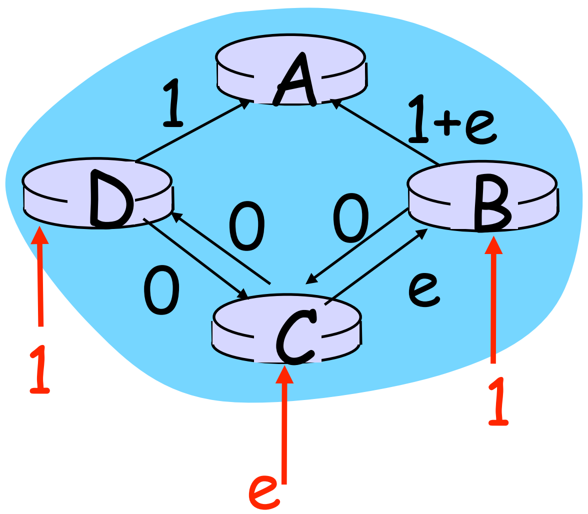
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

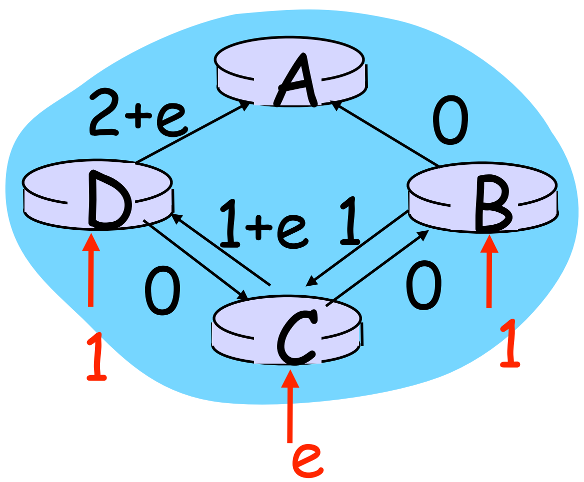
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

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



initially



… recompute routing

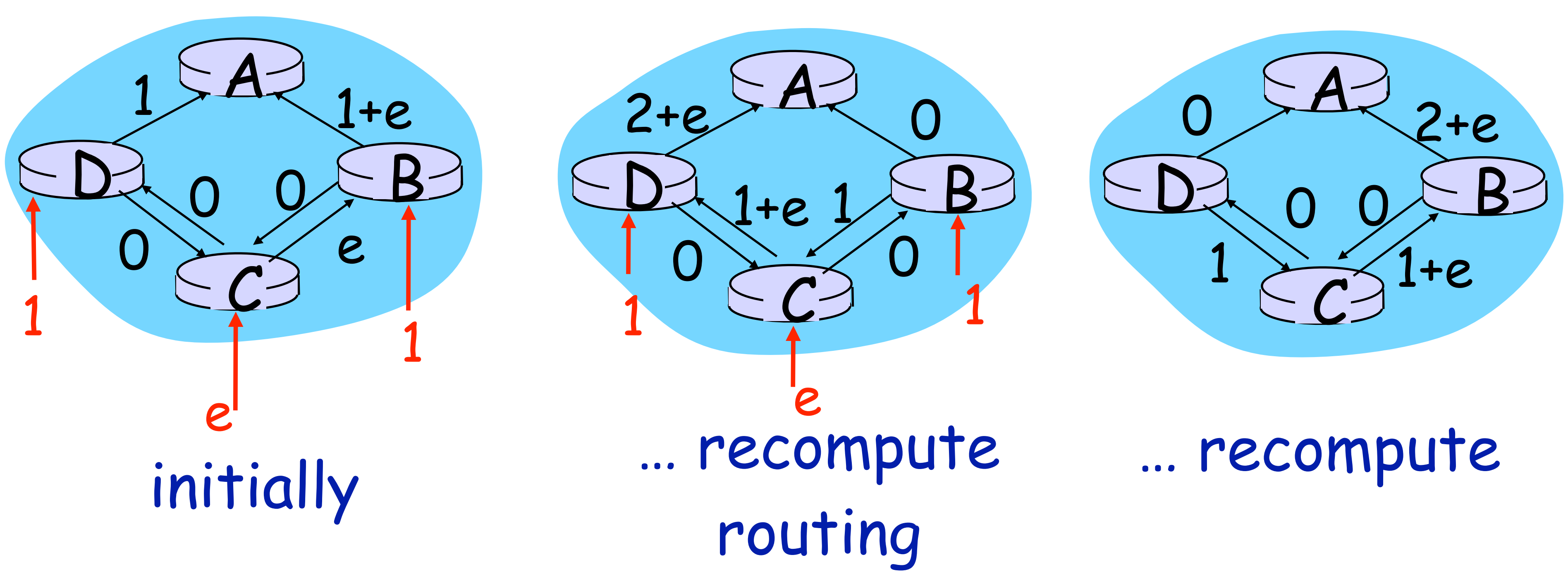
Dijkstra's algorithm, discussion

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(nlogn)

Oscillations possible:

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



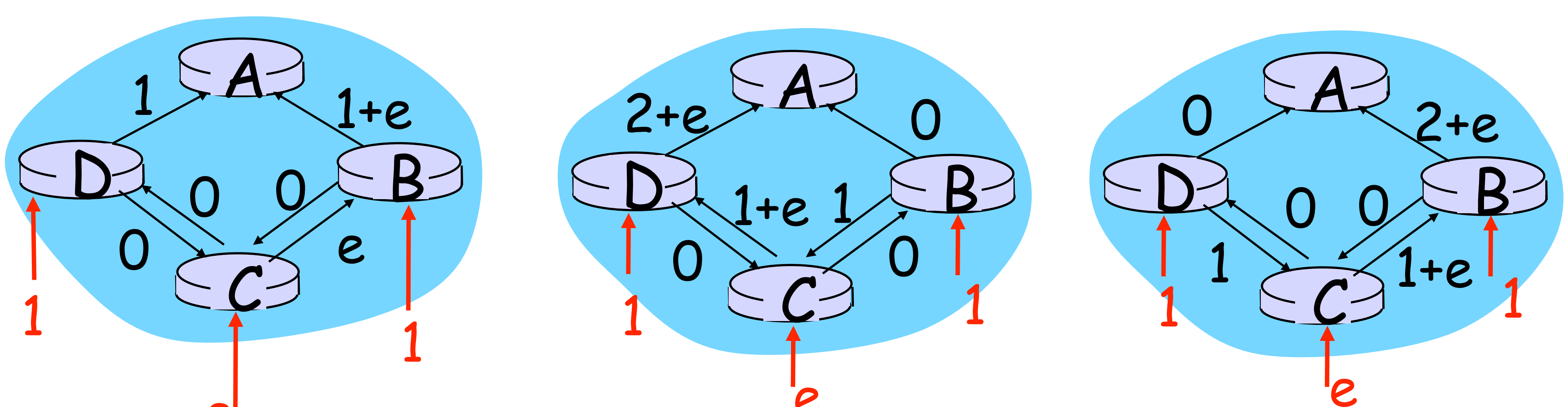
Dijkstra's algorithm, discussion

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(nlogn)

Oscillations possible:

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



initially ... recompute routing ... recompute

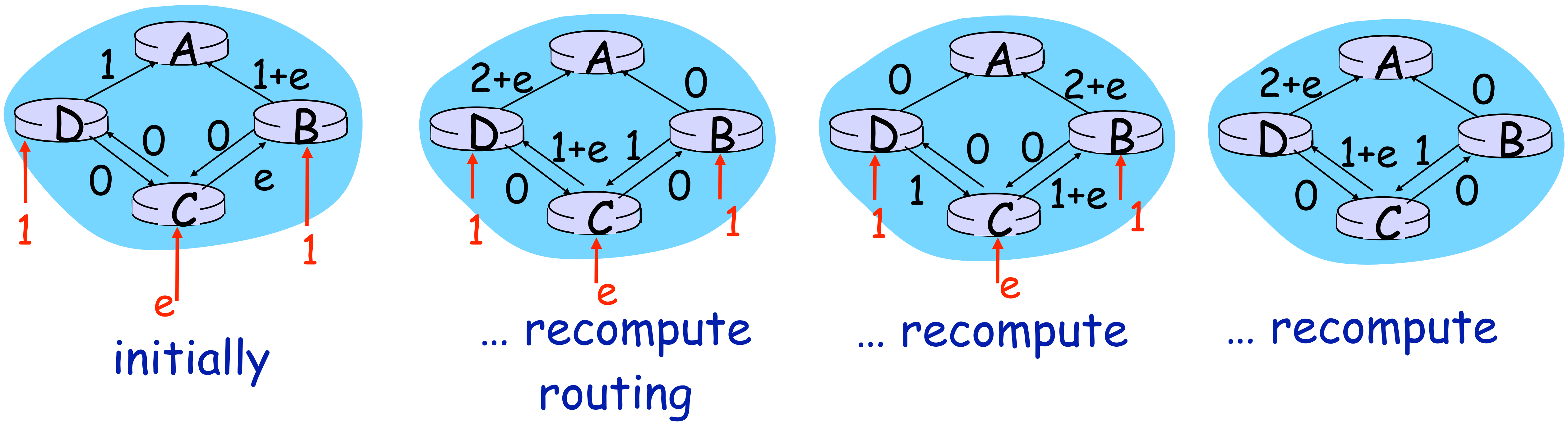
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

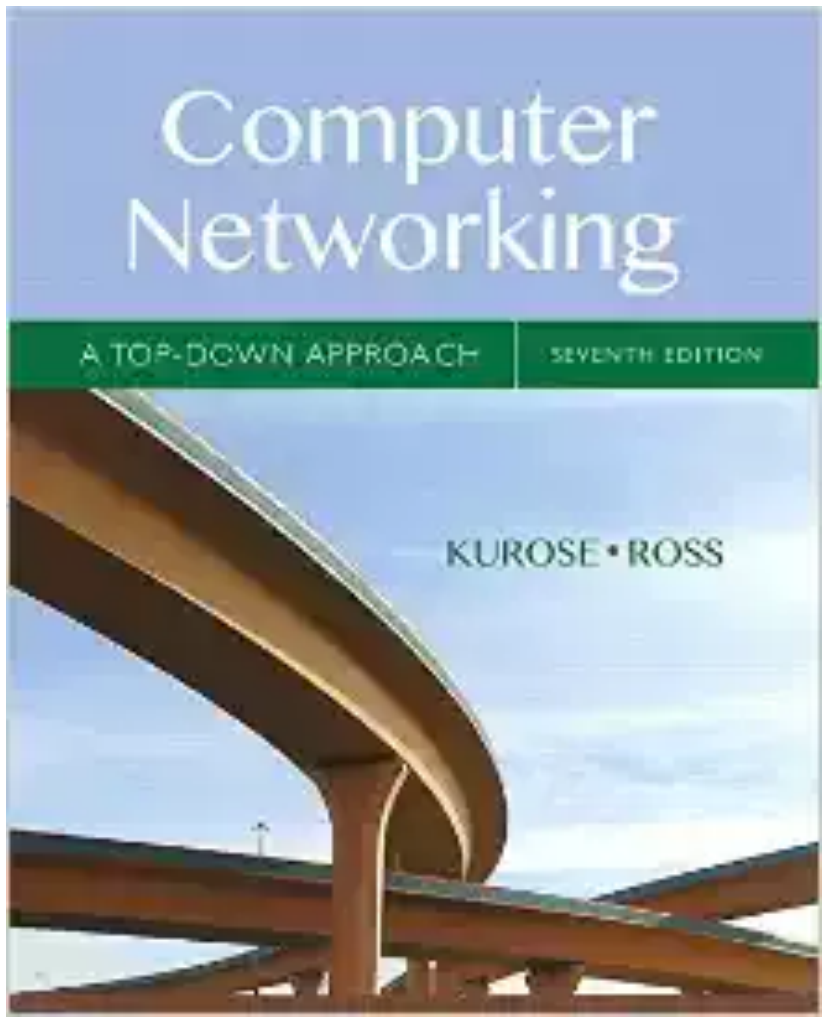
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

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



Reading Along ...



- Network layer is chapters 4 & 5
  - Routing Algorithms Distance vector routing

Distance Vector Algorithm

Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

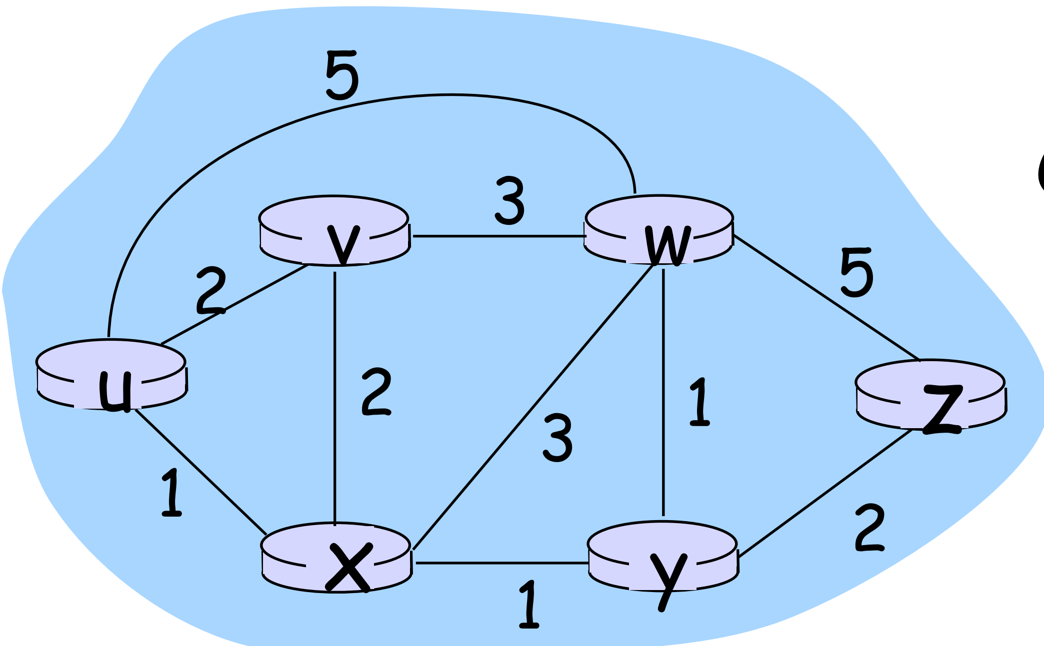
$d_x(y)$:= cost of least-cost path from x to y

Then

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

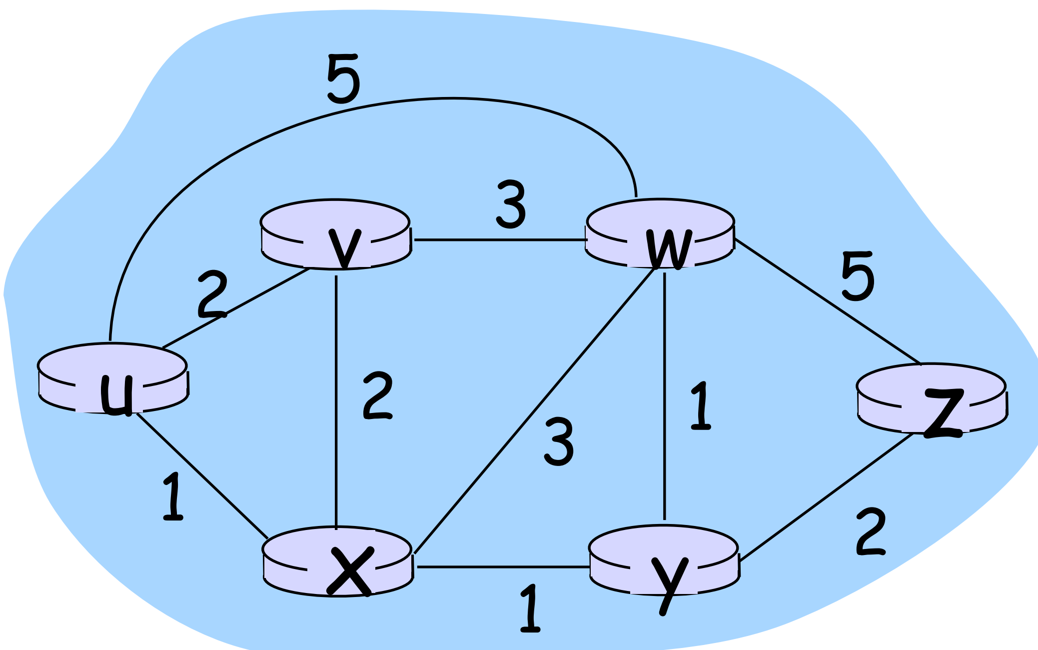
where min is taken over all neighbors v of x

Bellman-Ford example



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

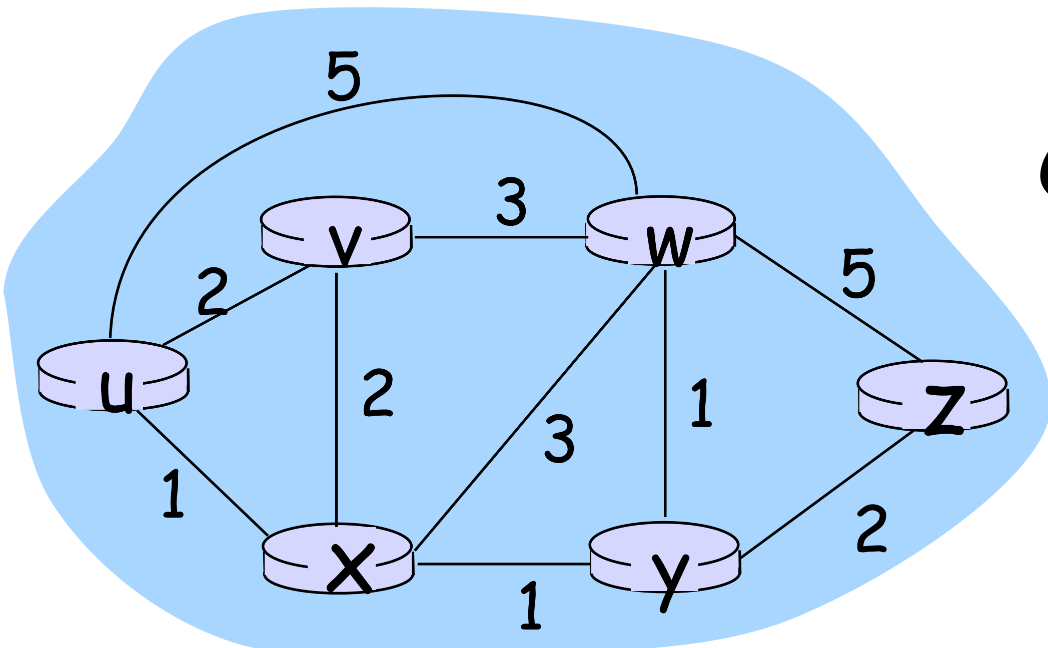
Bellman-Ford example



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

B-F equation says:

Bellman-Ford example

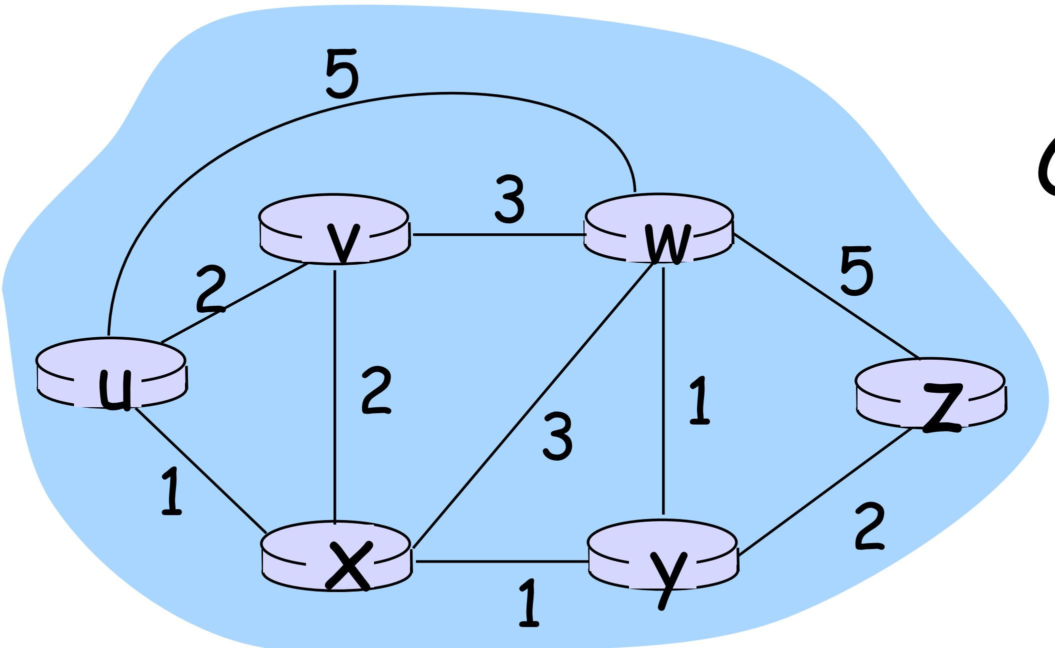


Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z),$$

Bellman-Ford example

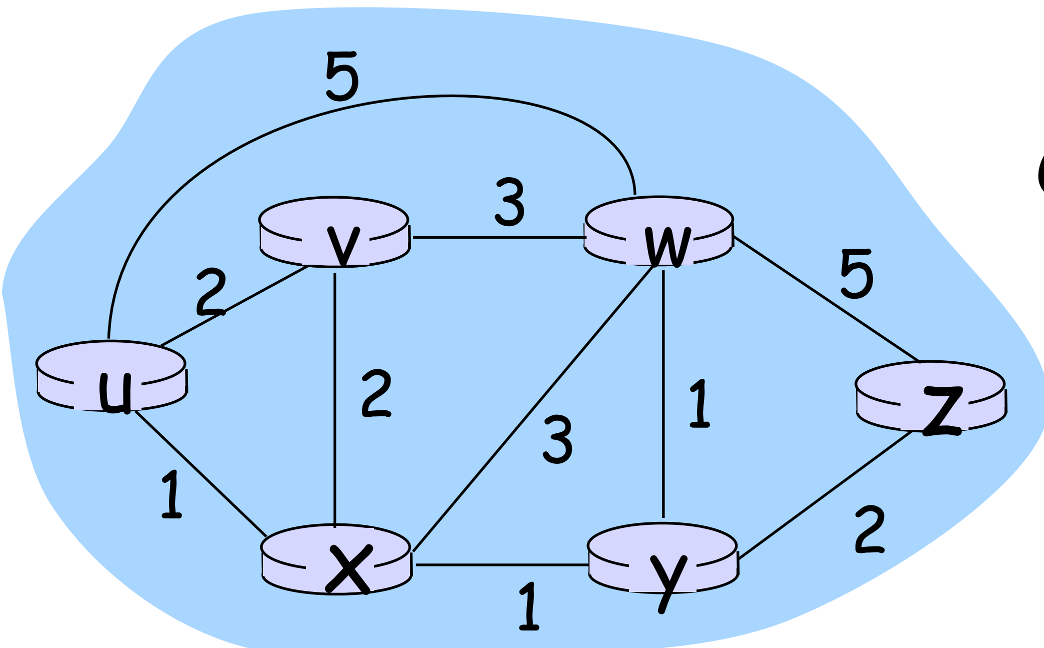


Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z),$$
$$c(u,x) + d_x(z),$$

Bellman-Ford example

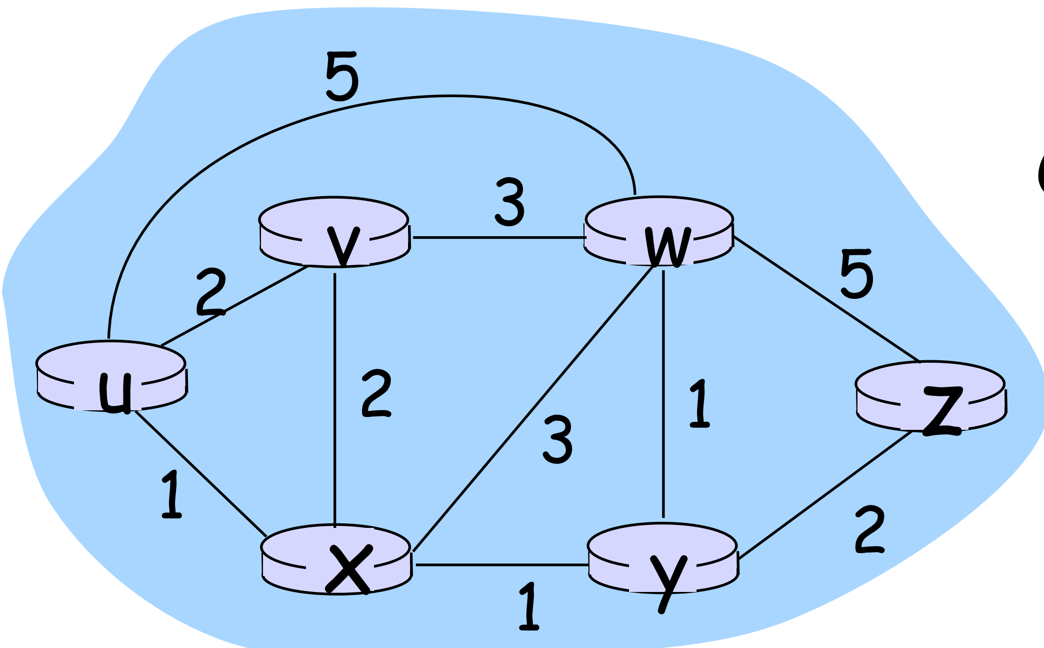


Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

Bellman-Ford example



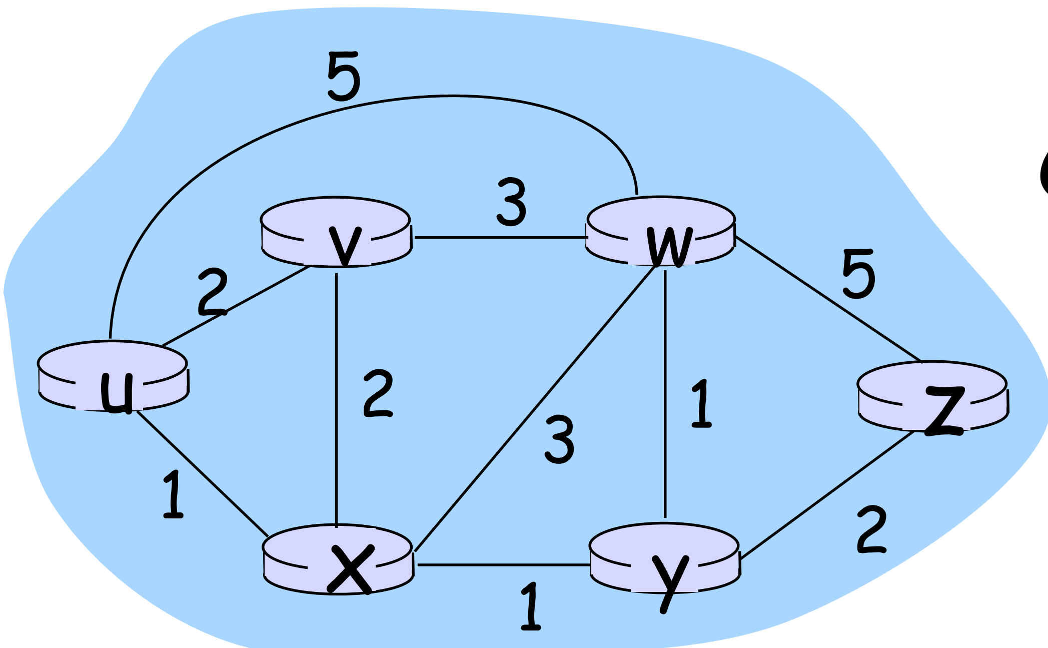
Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

$$= \min \{2 + 5,$$

Bellman-Ford example

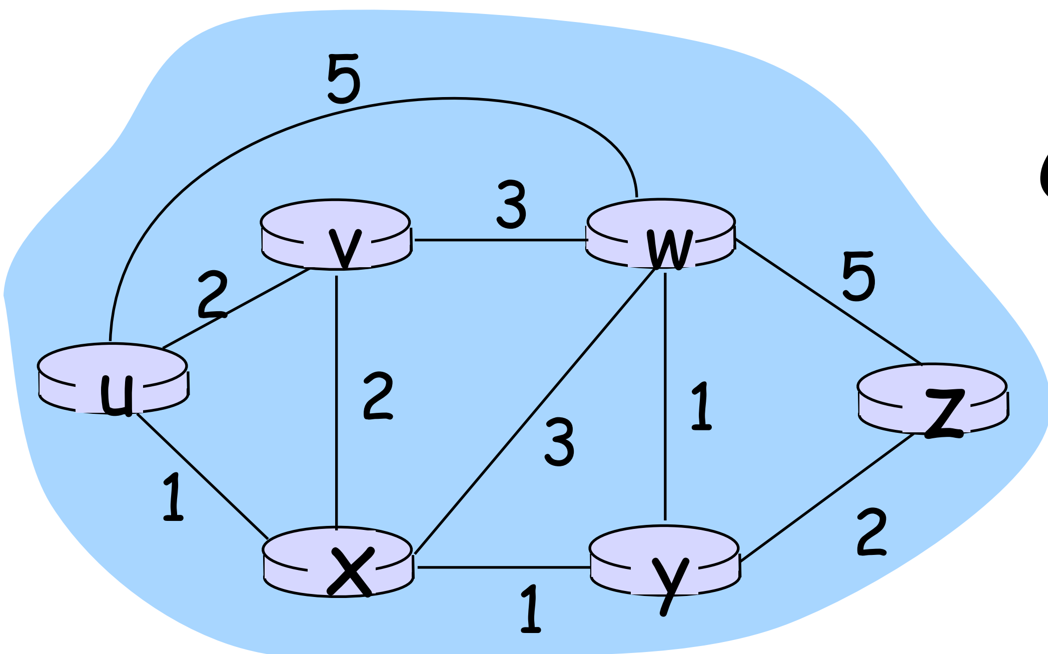


Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

Bellman-Ford example



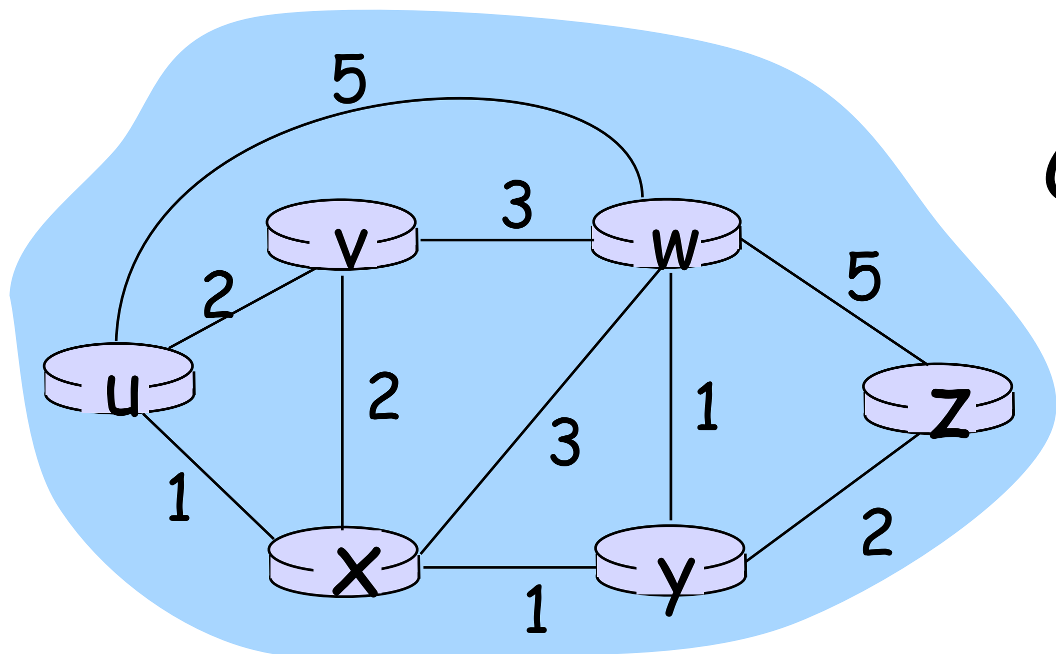
Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

Bellman-Ford example



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

B-F equation says:

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

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 $\mathbf{D}_x = [D_x(y): y \in N]$
- node x:
  - knows cost to each neighbor v: $c(x,v)$
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

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)

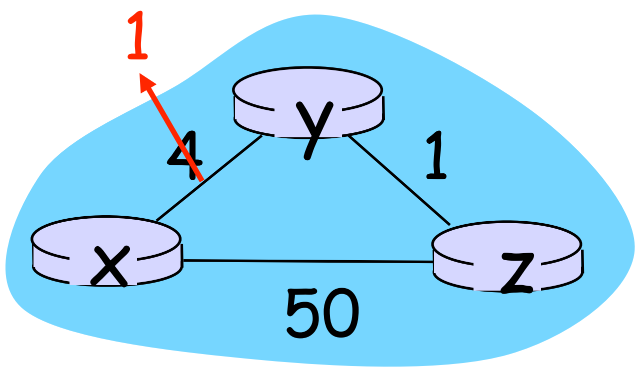
recompute estimates

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

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

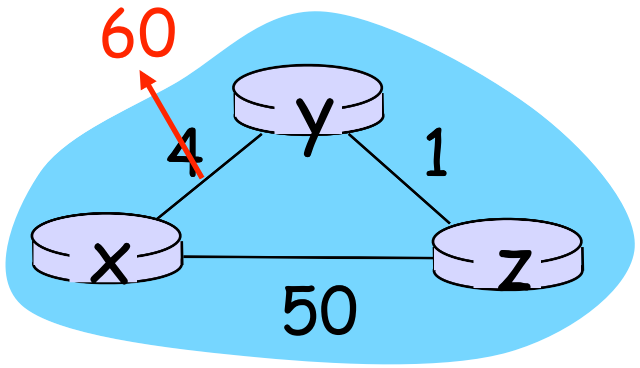


- good news travels fast

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



Comparison of LS and DV algorithms

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

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

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

Robustness: what happens if router malfunctions?

LS:

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

DV:

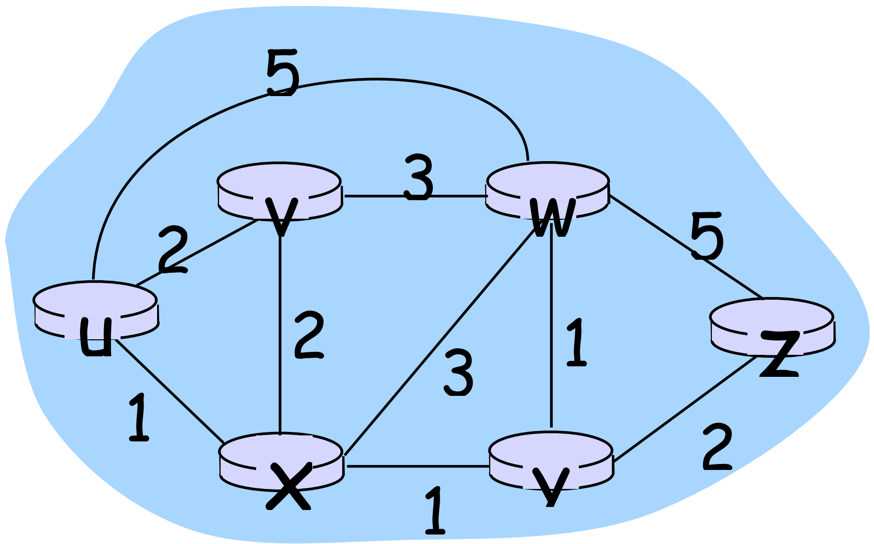
- DV node can advertise incorrect path cost
- each node's table used by others
  - error propagate thru network

Routing Algorithms

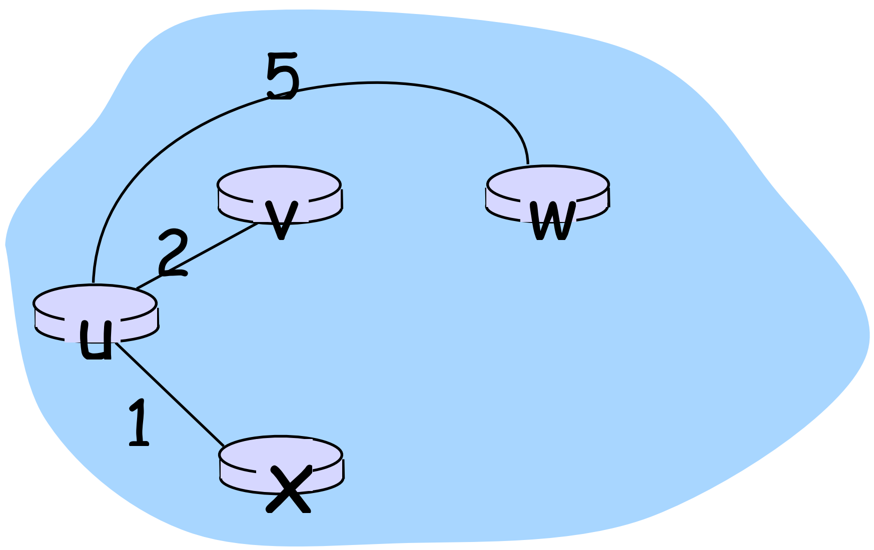
Link State

Distance Vector

Routing Algorithms

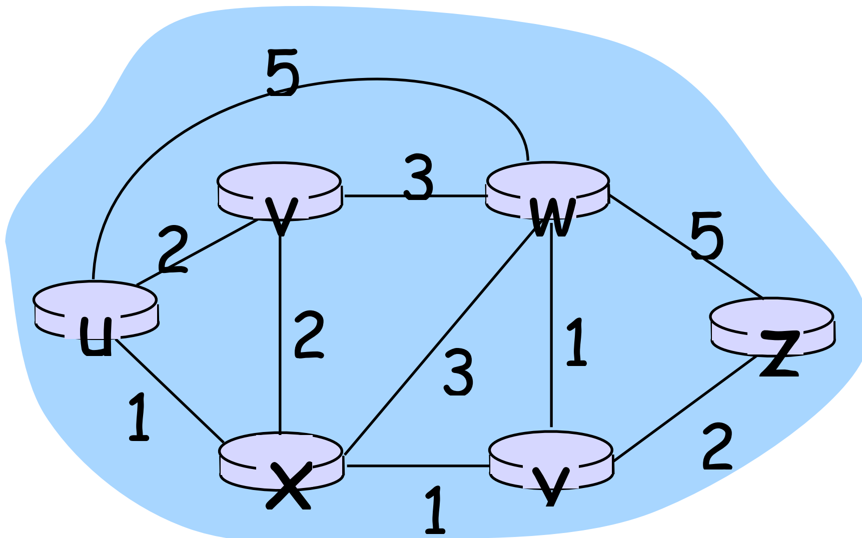


Link State

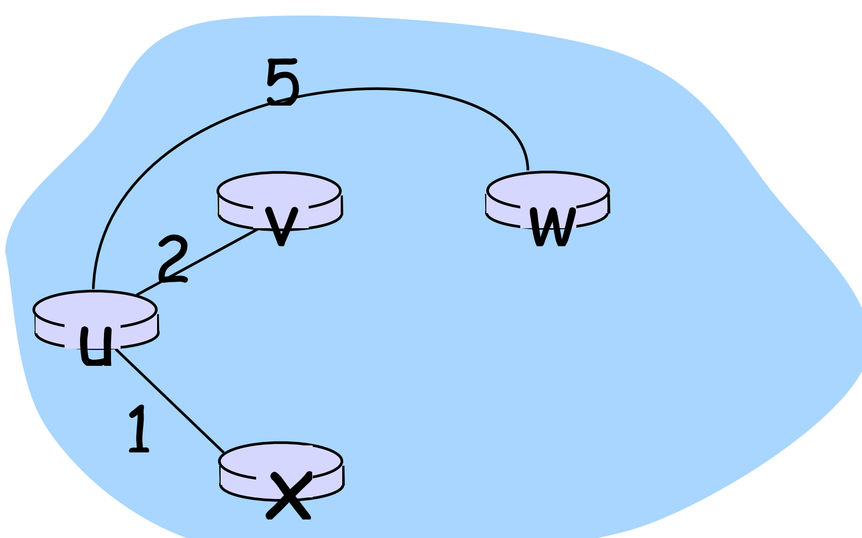


Distance Vector

Routing Algorithms



Link State

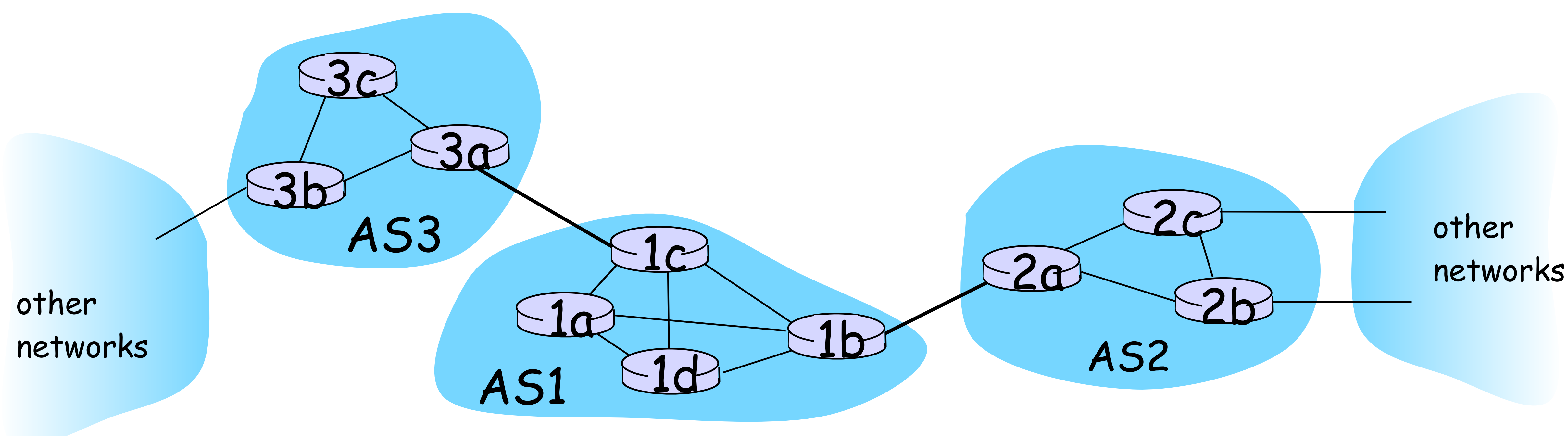


Distance Vector

- Both can work in small networks
- Neither great for dealing with big networks
  - millions of destinations

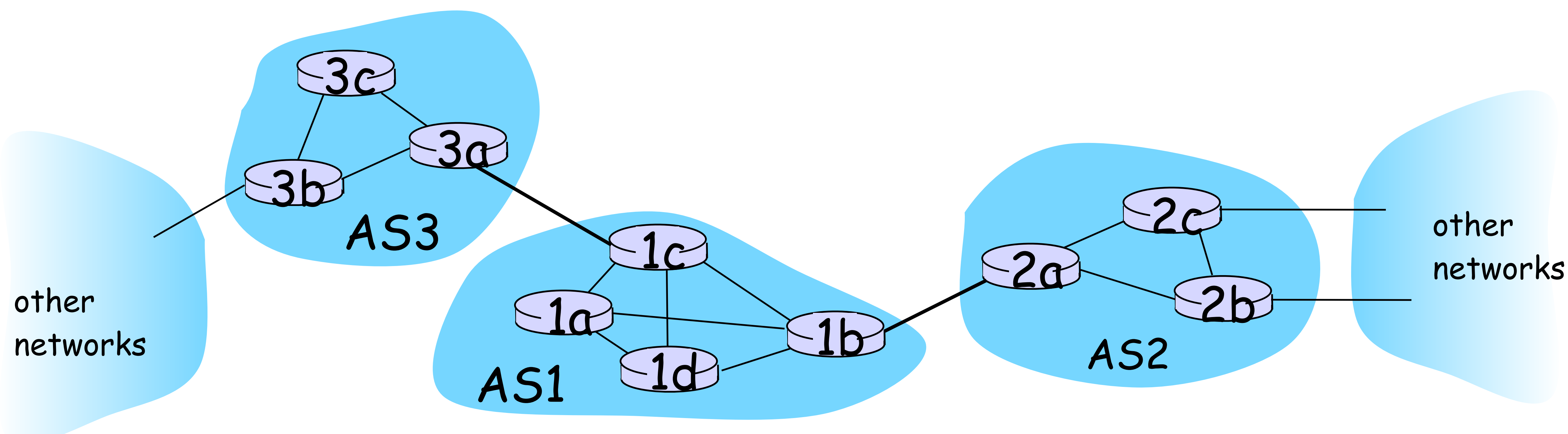
Routing Algorithms

- We must apply divide and conquer
- Aggregate routers within organizations together and deal with them as a single unit
  - An "autonomous system"

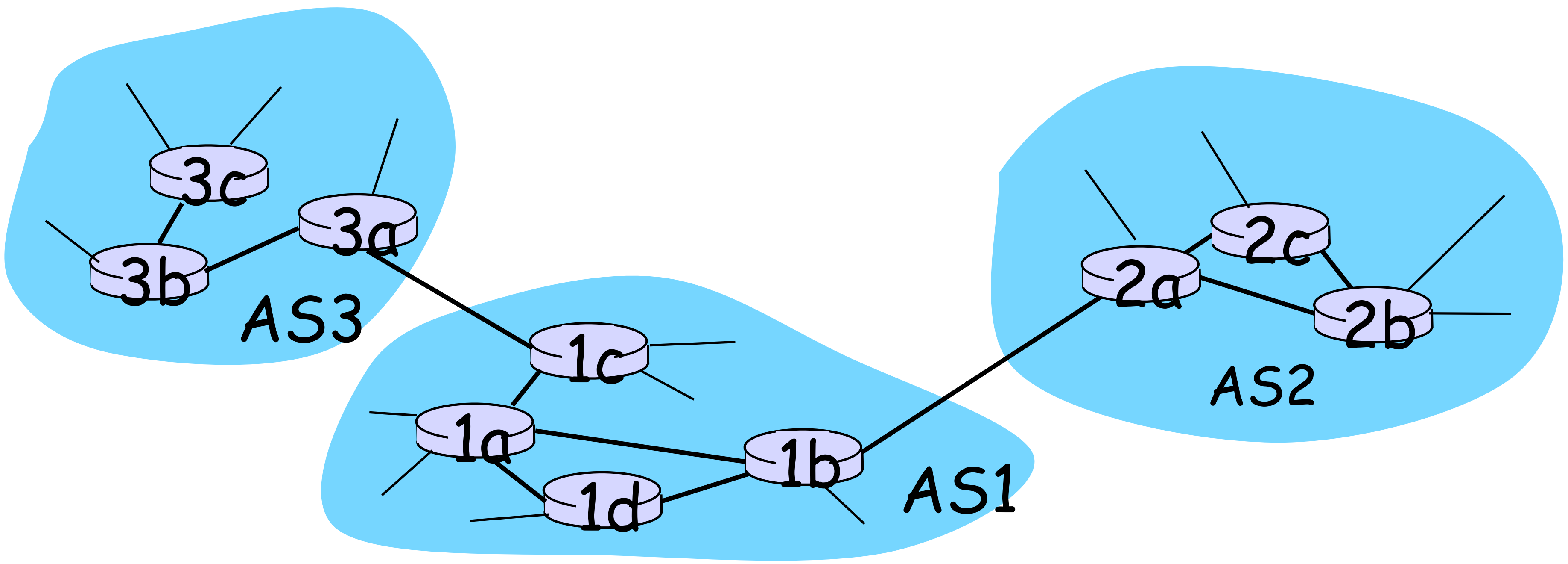


Routing Algorithms

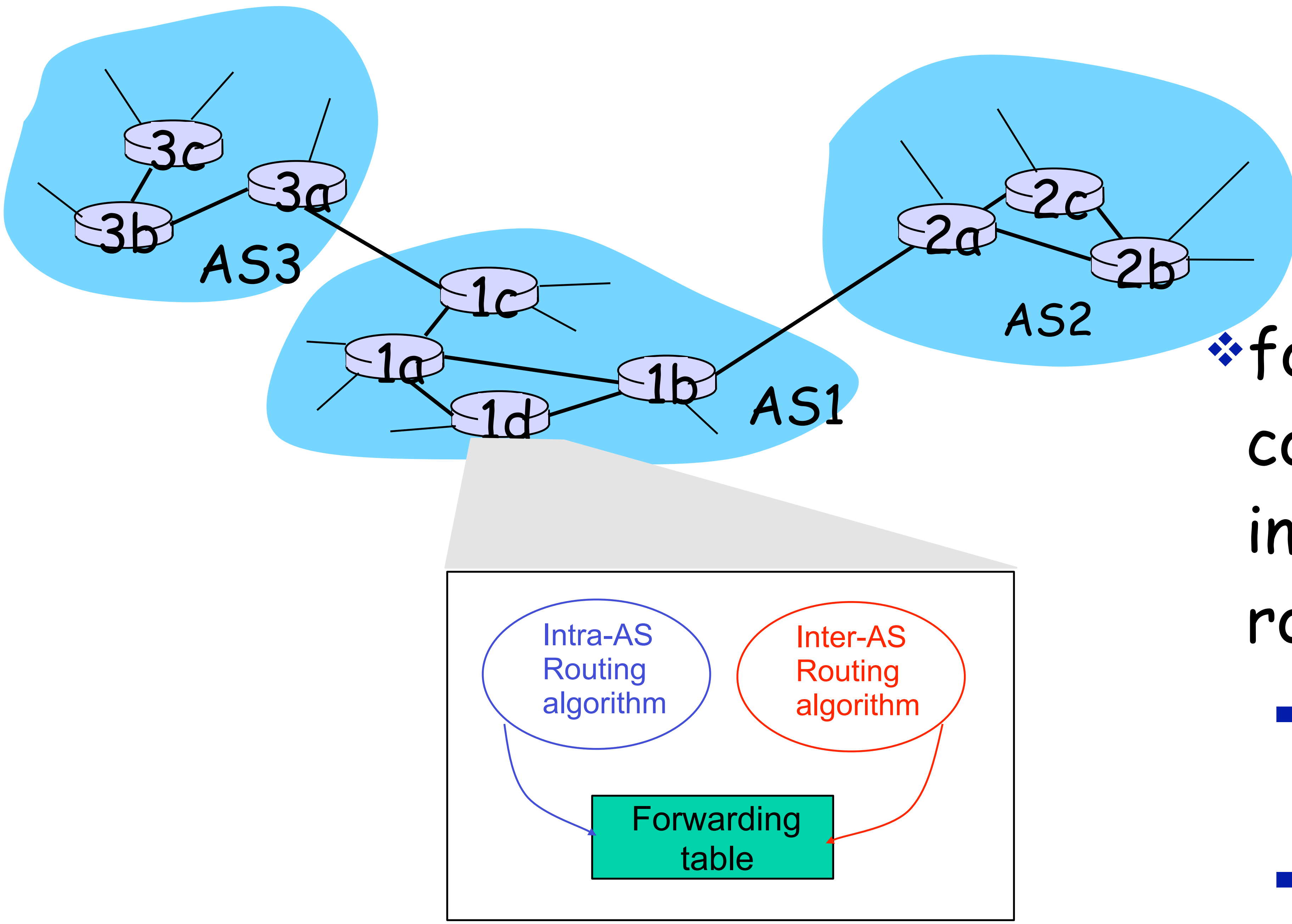
- Must have two routing algorithms:
  - Intra-AS or Interior routing algorithms to manage routing inside an AS
  - Inter-AS or Exterior routing algorithms to manage routing between ASes



Interconnected ASes



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