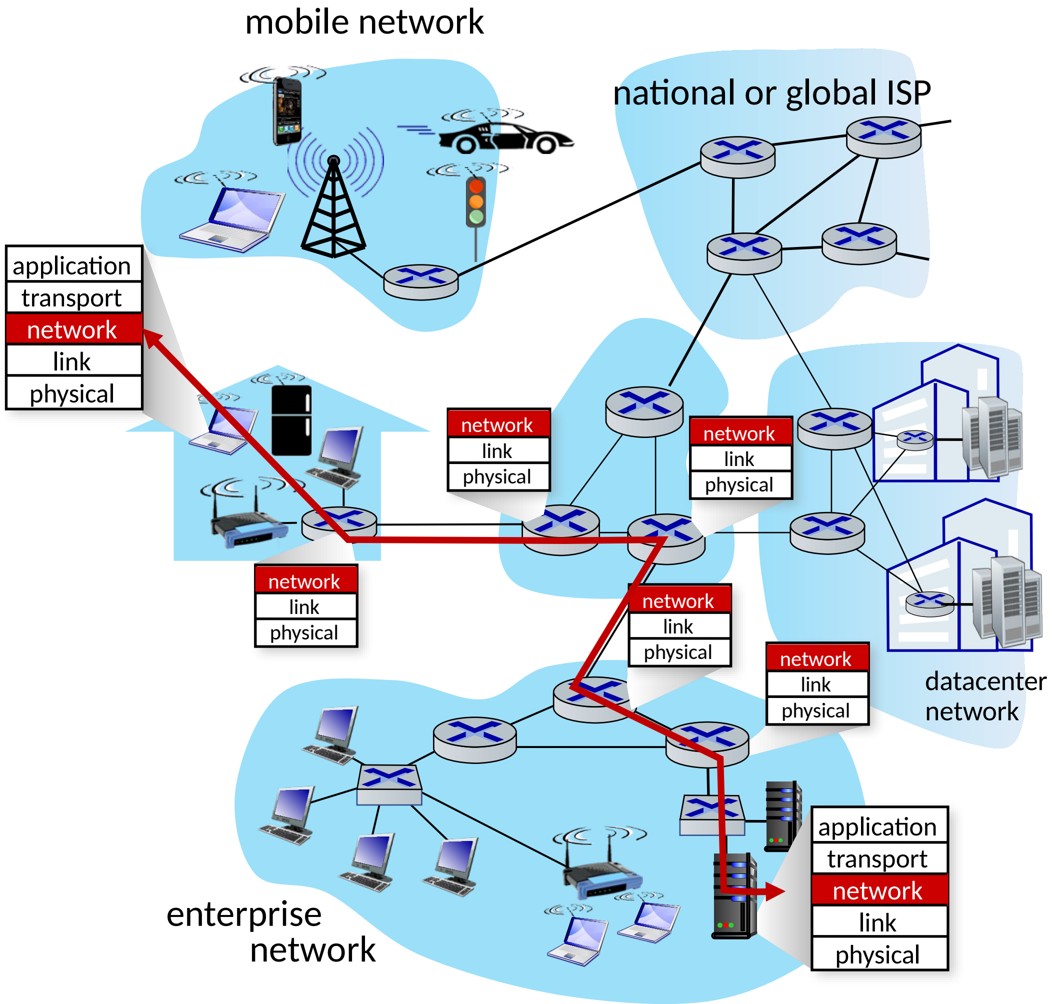
Network Layer: Routing Protocols

Computer Networking: A Top-Down Approach
9th edition
Jim Kurose, Keith Ross
Pearson, 2025

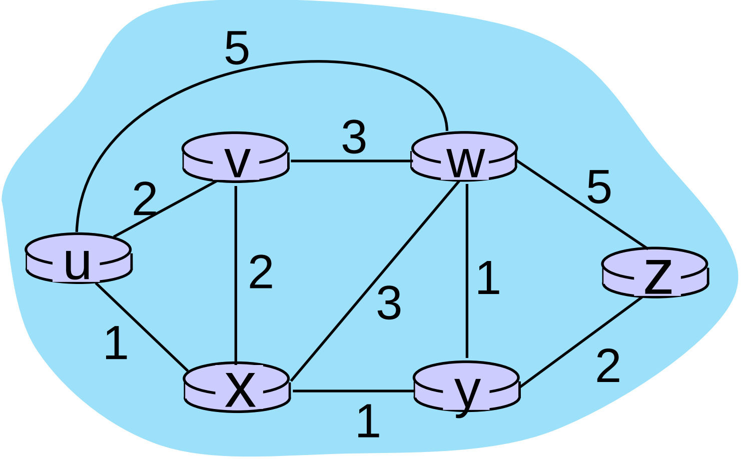
Routing protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets traverse from given initial source host to final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!



Graph abstraction: link costs



$c_{a,b}$: cost of *direct* link connecting *a* and *b*

e.g., $c_{w,z} = 5$, $c_{u,z} = \infty$

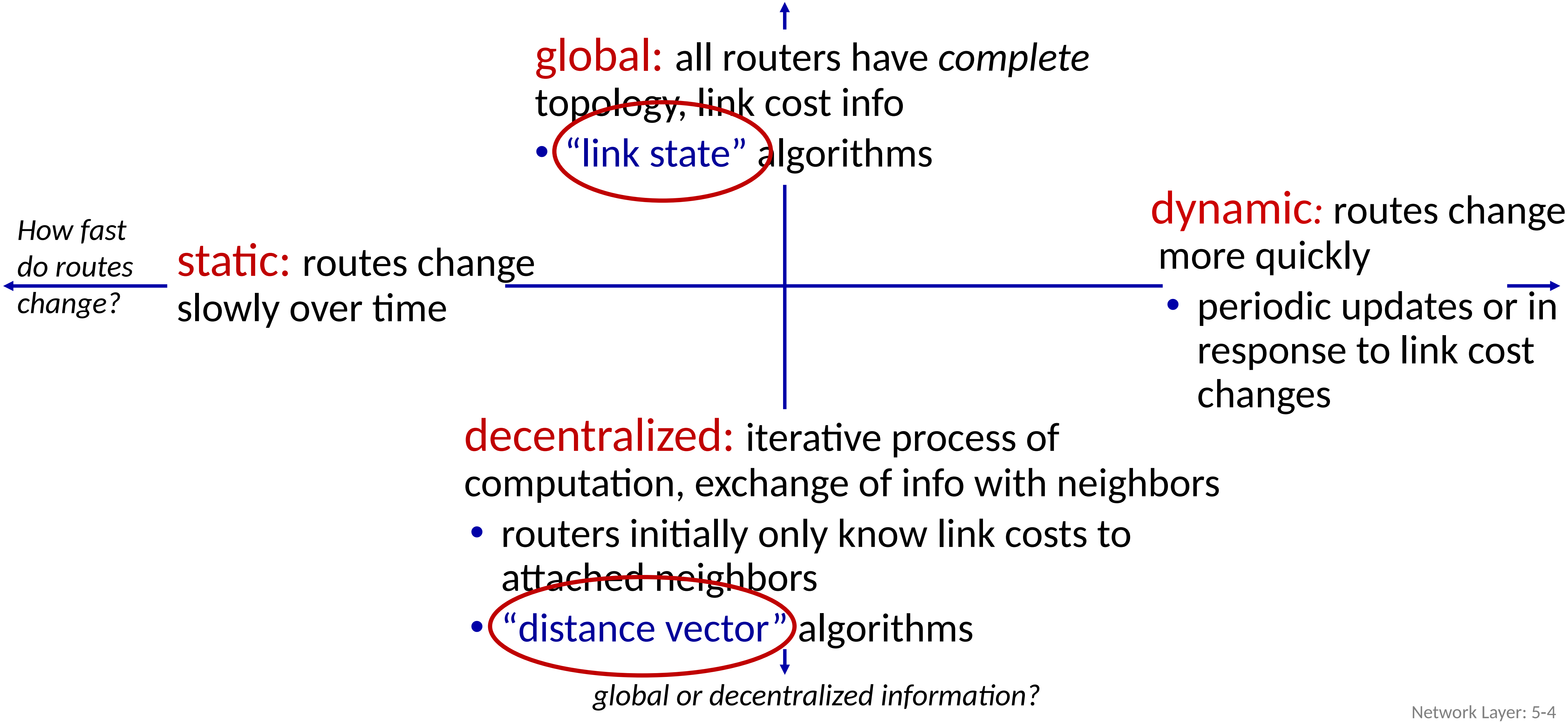
cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

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)* }

Routing algorithm classification



global: all routers have *complete* topology, link cost info

- "link state" algorithms

How fast do routes change?

static: routes change slowly over time

dynamic: routes change more quickly

- periodic updates or in response to link cost changes

decentralized: iterative process of computation, exchange of info with neighbors

- routers initially only know link costs to attached neighbors
- "distance vector" algorithms

global or decentralized information?

Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Dijkstra’s link-state routing algorithm

- centralized: network 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

- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v)*: *current* estimate of cost of least-cost-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

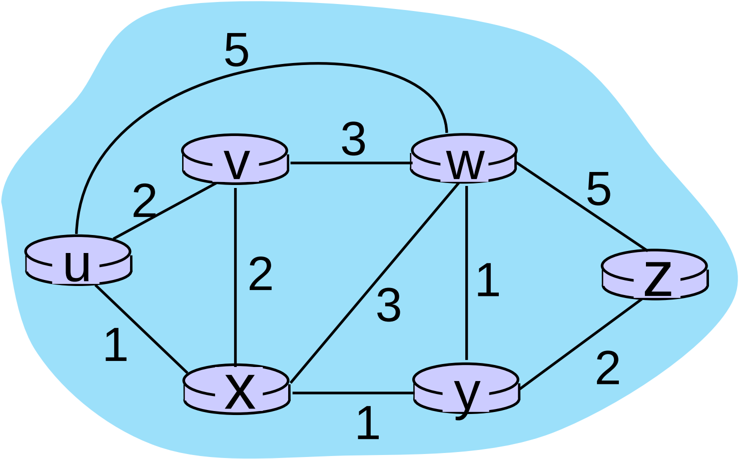
Dijkstra’s link-state routing algorithm

```
Initialization:
  $N' = \{u\}$                      /* compute least cost path from u to all other nodes */
  for all nodes $v$
    if $v$ adjacent to $u$          /* $u$ initially knows direct-path-cost only to direct neighbors */
      then $D(v) = c_{u,v}$         /* but may not be minimum cost! */
    else $D(v) = \infty$

Loop
  find $w$ not in $N'$ such that $D(w)$ is a minimum
  add $w$ to $N'$
  update $D(v)$ for all $v$ adjacent to $w$ and not in $N'$ :
    $D(v) = \min(D(v), D(w) + c_{w,v})$
  /* new least-path-cost to $v$ is either old least-cost-path to $v$ or known
  least-cost-path to $w$ plus direct-cost from $w$ to $v$ */
until all nodes in $N'$
```

Dijkstra’s algorithm: an example

		v	w	x	y	z
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						
2						
3						
4						
5						

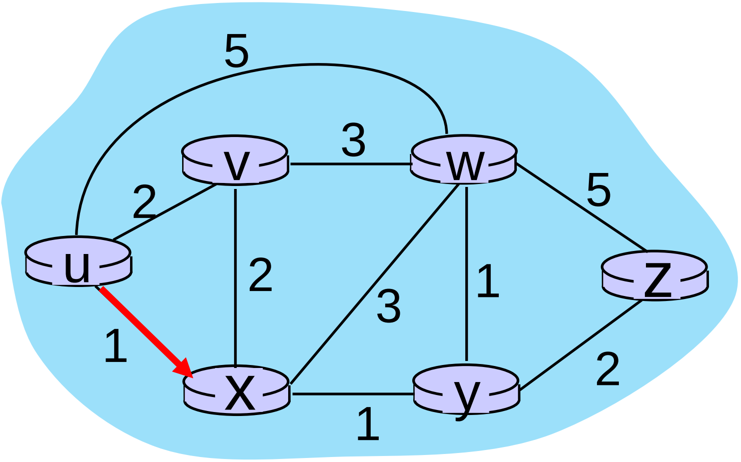


Initialization (step 0):

For all *a*: if *a* adjacent to *u* then *D(a)* = $c_{u,a}$

Dijkstra’s algorithm: an example

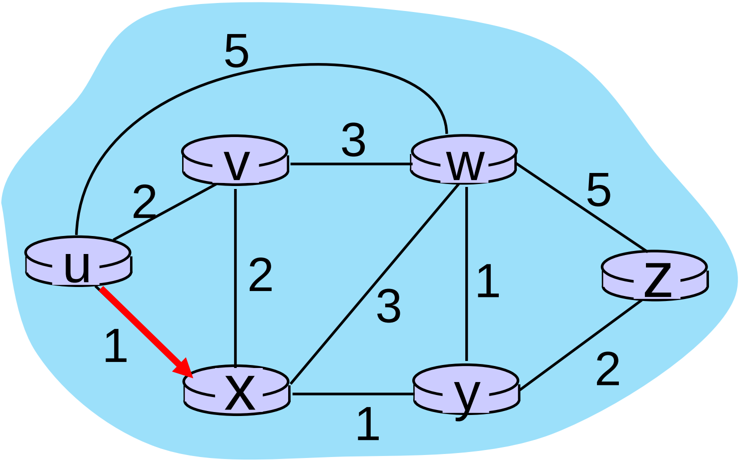
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux					
2						
3						
4						
5						



```
*Loop*
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
```

Dijkstra's algorithm: an example

		v	w	x	y	z
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						
3						
4						
5						



```
Loop
    find a not in N' such that D(a) is a minimum
    add a to N'
  update D(b) for all b adjacent to a and not in N' :
      D(b) = min ( D(b), D(a) + c_{a,b} )
```

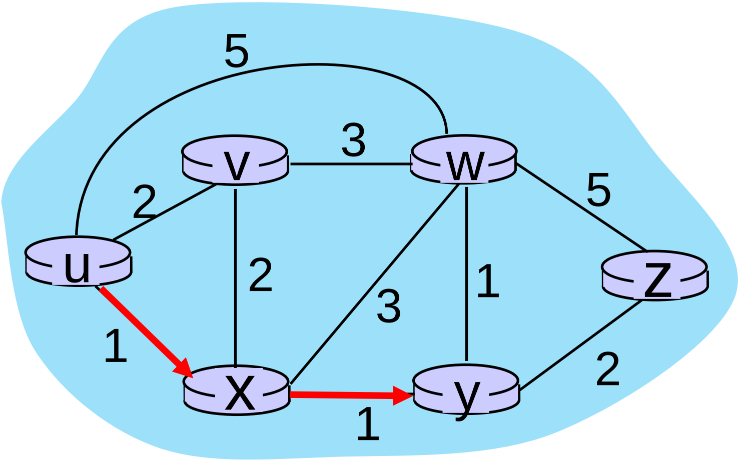
$D(v) = \min(D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$

$D(w) = \min(D(w), D(x) + c_{x,w}) = \min(5, 1+3) = 4$

$D(y) = \min(D(y), D(x) + c_{x,y}) = \min(inf, 1+1) = 2$

Dijkstra’s algorithm: an example

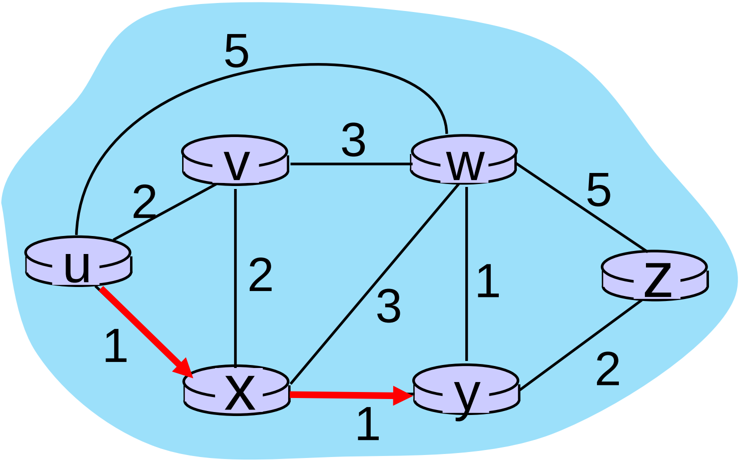
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy					
3						
4						
5						



```
Loop
    find a not in N' such that D(a) is a minimum
    add a to N'
```

Dijkstra's algorithm: an example

		v	w	x	y	z
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						
4						
5						



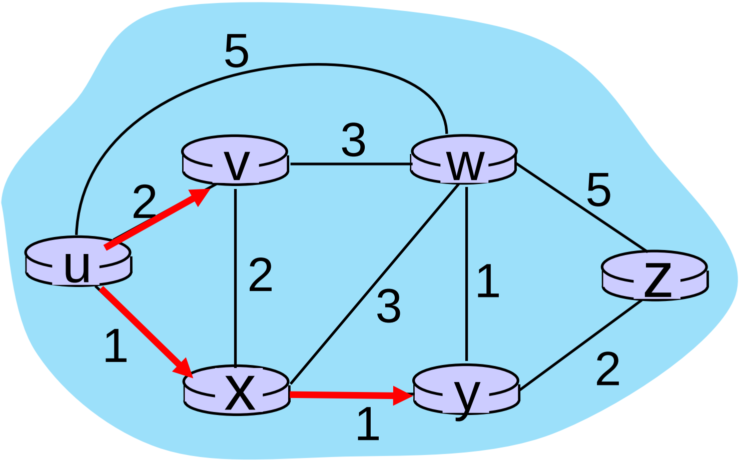
```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
    update $D(b)$ for all $b$ adjacent to $a$ and not in $N'$ :
        $D(b) = \min ( D(b), D(a) + c_{a,b} )$
```

$D(w) = \min (D(w), D(y) + c_{y,w}) = \min (4, 2+1) = 3$

$D(z) = \min (D(z), D(y) + c_{y,z}) = \min(\text{inf},2+2) = 4$

Dijkstra’s algorithm: an example

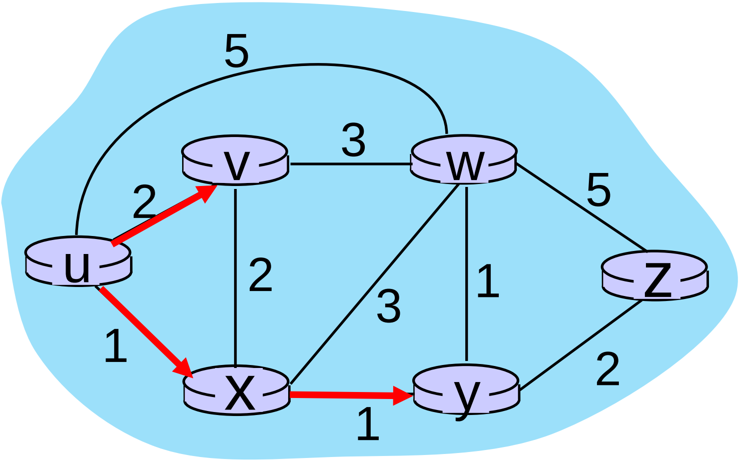
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv					
4						
5						



```
Loop
    find a not in N' such that D(a) is a minimum
    add a to N'
```

Dijkstra’s algorithm: an example

		v	w	x	y	z
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						
5						

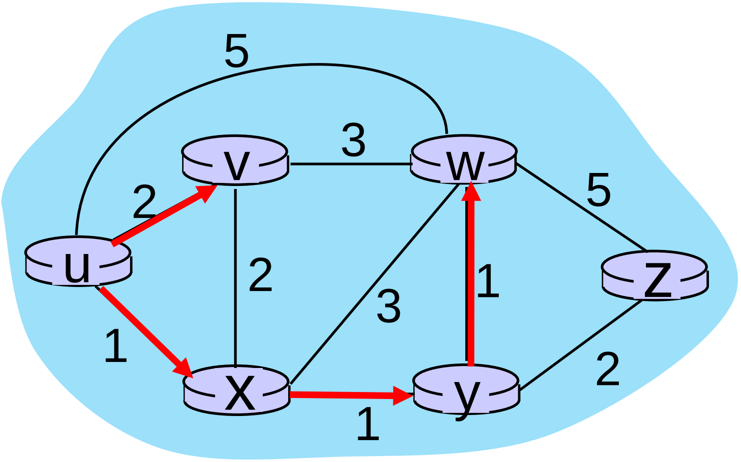


```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
    update $D(b)$ for all $b$ adjacent to $a$ and not in $N'$ :
        $D(b) = \min ( D(b), D(a) + c_{a,b} )$
```

$D(w) = min (D(w), D(v) + c_{v,w}) = min (3, 2+3) = 3$

Dijkstra’s algorithm: an example

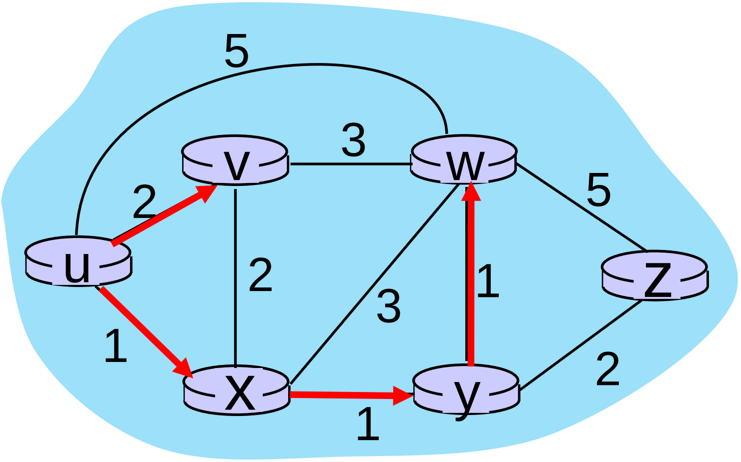
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					
5						



```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
```

Dijkstra's algorithm: an example

		v	w	x	y	z
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						

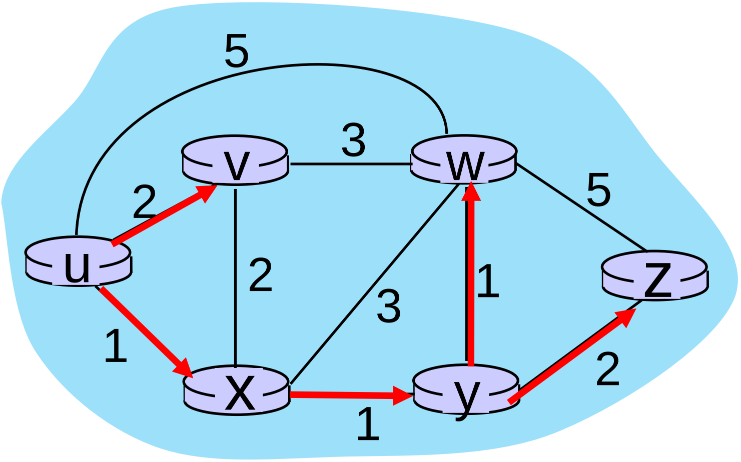


```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
    update $D(b)$ for all $b$ adjacent to $a$ and not in $N'$ :
        $D(b) = \min ( D(b), D(a) + c_{a,b} )$
```

$D(z) = \min (D(z), D(w) + c_{w,z}) = \min (4, 3+5) = 4$

Dijkstra’s algorithm: an example

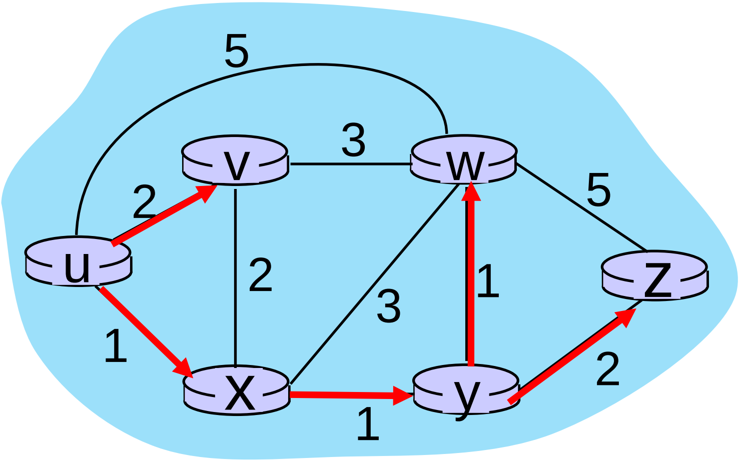
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
```

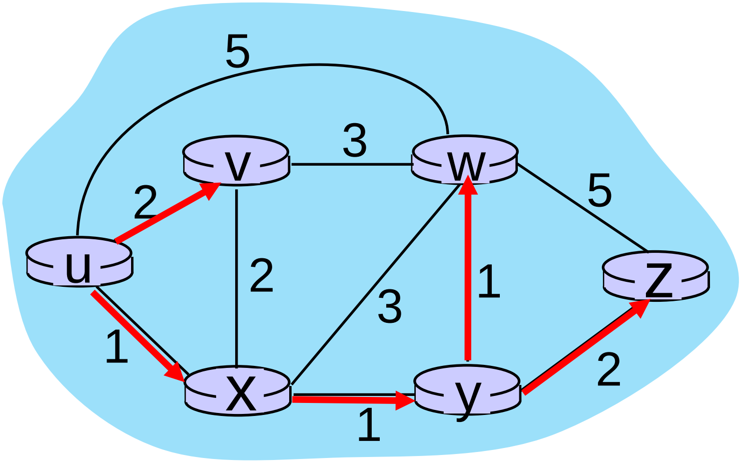
Dijkstra’s algorithm: an example

		v	w	x	y	z
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					

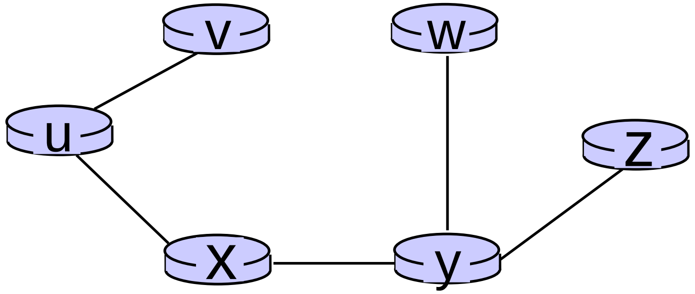


```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
    update $D(b)$ for all $b$ adjacent to $a$ and not in $N'$ :
        $D(b) = \min ( D(b), D(a) + c_{a,b} )$
```

Dijkstra’s algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

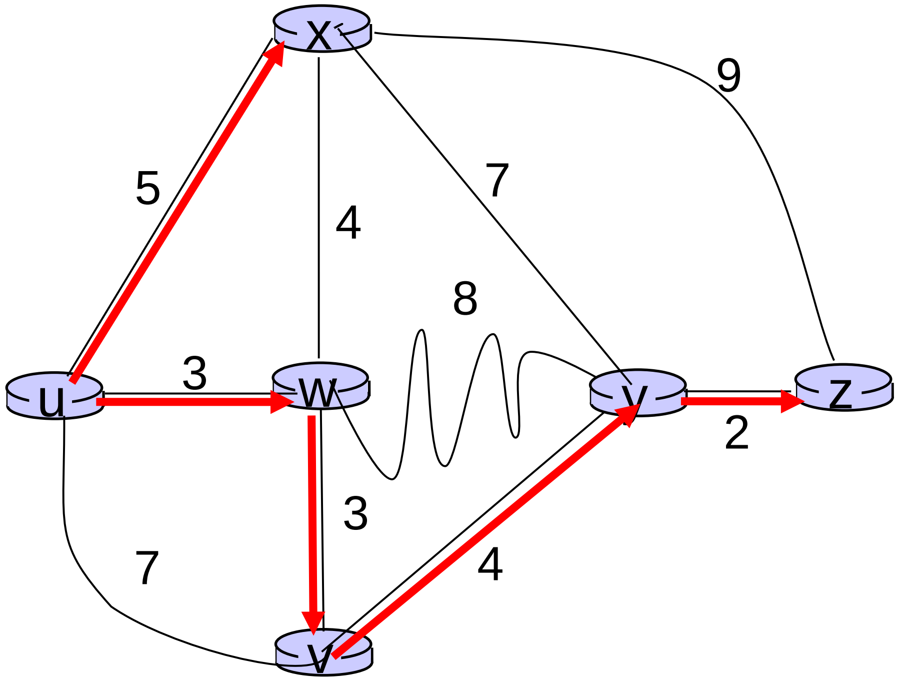
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Dijkstra’s algorithm: another example

		v	w	x	y	z
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 least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

Dijkstra's algorithm: discussion

algorithm complexity: *n* nodes

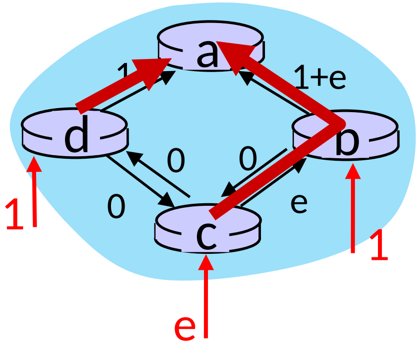
- each of *n* iteration: need to check all nodes, *w*, not in *N*
- *n*(*n*+1)/2 comparisons: O(n^2) complexity
- more efficient implementations possible: O(*n*log*n*)

message complexity:

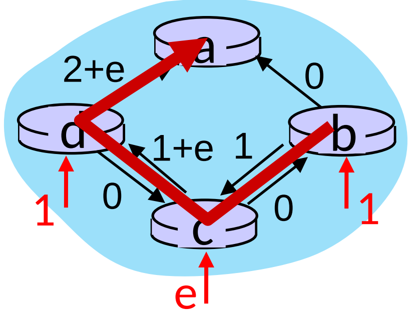
- each router must *broadcast* its link state information to other *n* routers
- efficient (and interesting!) broadcast algorithms: O(*n*) link crossings to disseminate a broadcast message from one source
- each router's message crosses O(*n*) links: overall message complexity: O(n^2)

Dijkstra’s algorithm: oscillations possible

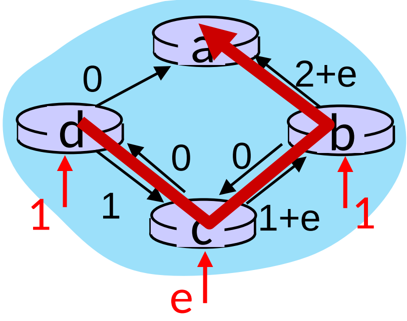
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



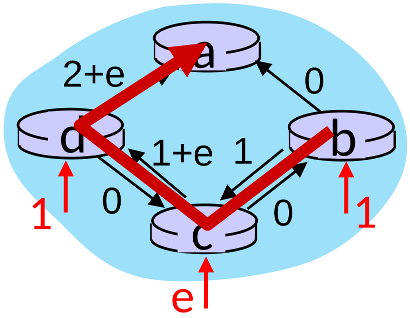
initially



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

Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

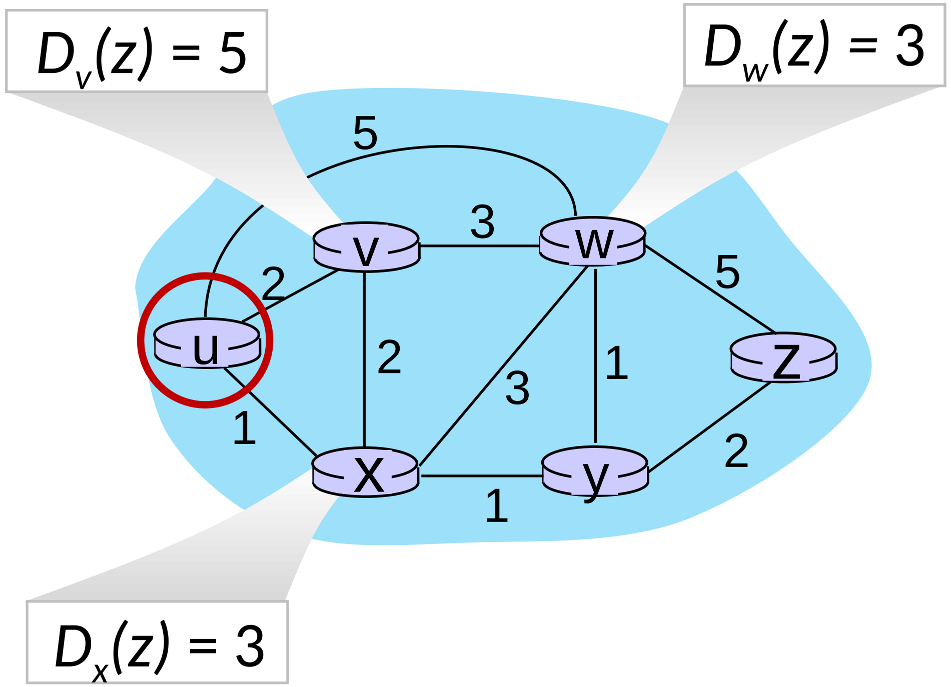
v’s estimated least-cost-path cost to y

min taken over all neighbors v of x

direct cost of link from x to v

Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x*,*v*,*w*, know that for destination *z*:



Bellman-Ford equation says:

$$\boxed{D_u(z)} = \min \{ c_{u,v} + D_v(z), \boxed{c_{u,x} + D_x(z)}, c_{u,w} + D_w(z) \}$$

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

*node achieving minimum (x) is **next hop** on estimated least-cost path to destination (z)*

Distance vector algorithm

key idea:

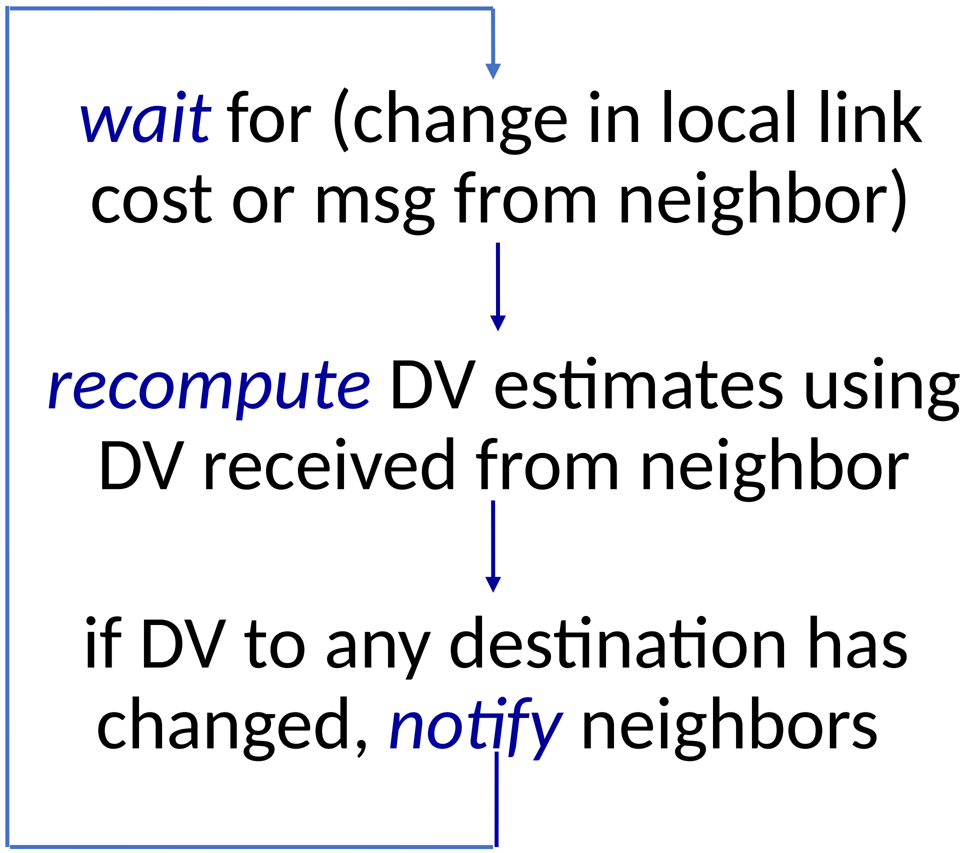
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

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

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $\mathrm{d}_x(\mathrm{y})$

Distance vector algorithm:

each node:



wait for (change in local link cost or msg from neighbor)

recompute DV estimates using DV received from neighbor

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

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

Distance vector: example

DV in a:

$D_a(a)=0$

$D_a(b) = 8$

$D_a(c) = \infty$

$D_a(d) = 1$

$D_a(e) = \infty$

$D_a(f) = \infty$

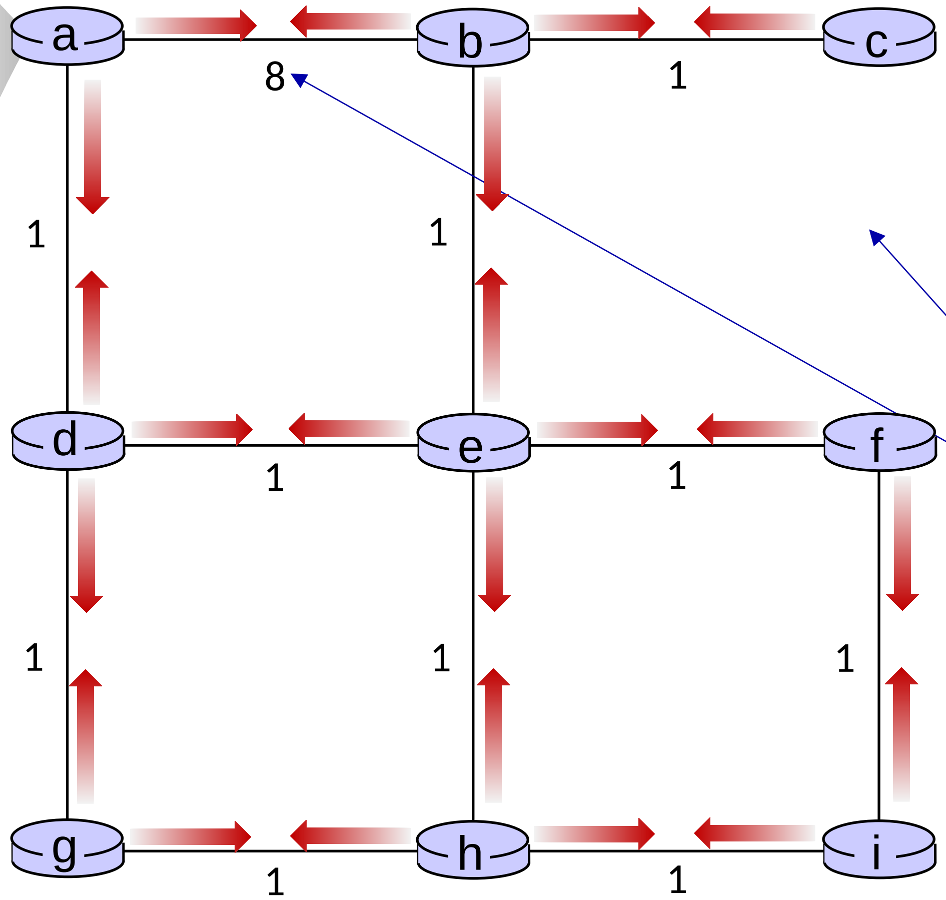
$D_a(g) = \infty$

$D_a(h) = \infty$

$D_a(i) = \infty$

t=0

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors



A few asymmetries:

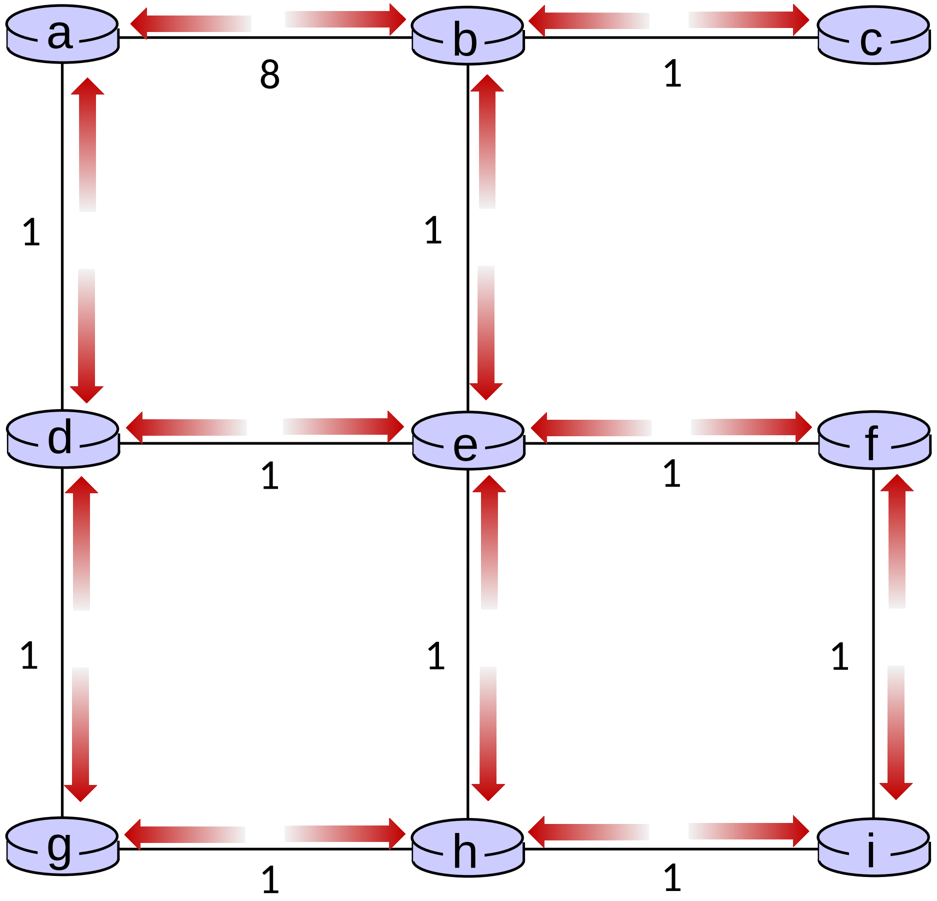
- missing link
- larger cost

Distance vector example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

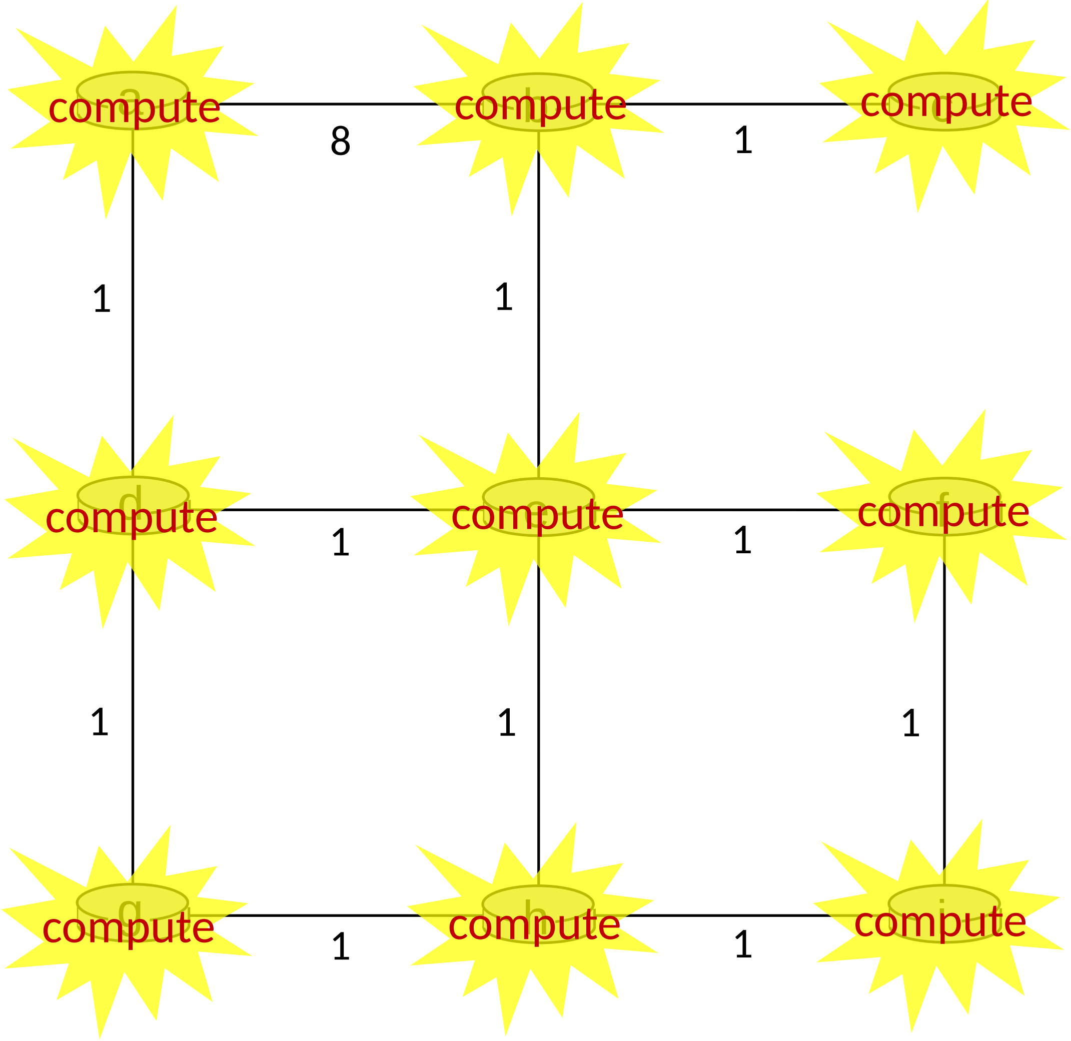


Distance vector example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

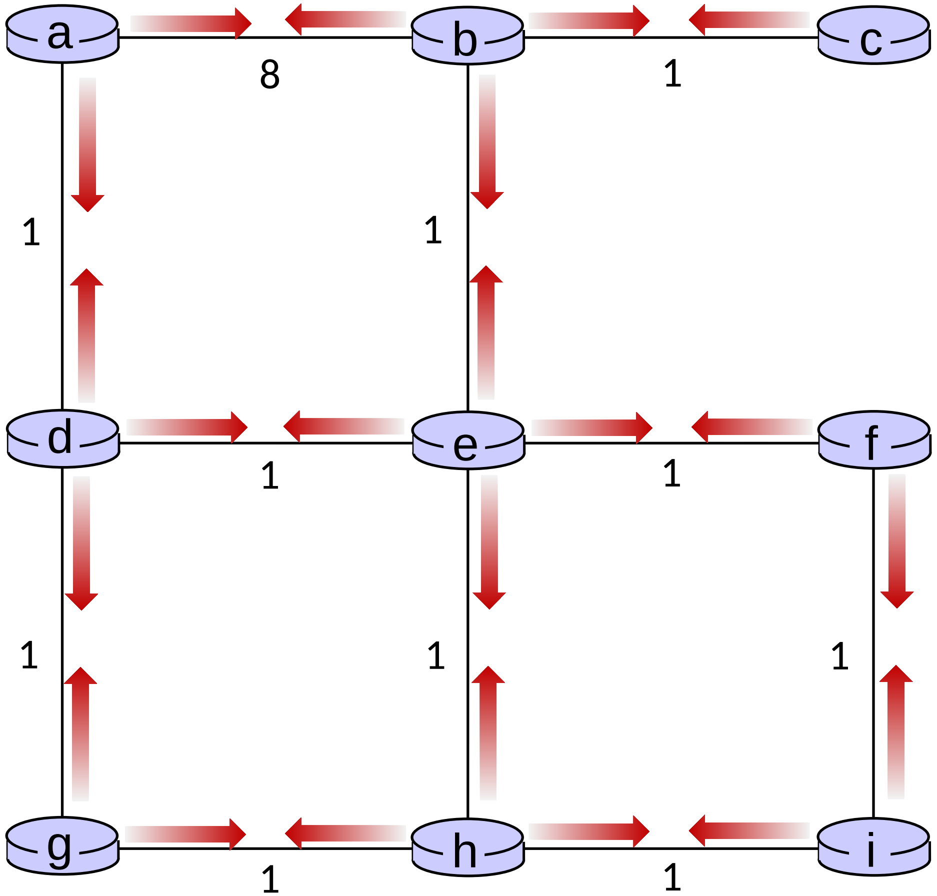


Distance vector example: iteration

t=1

All nodes:

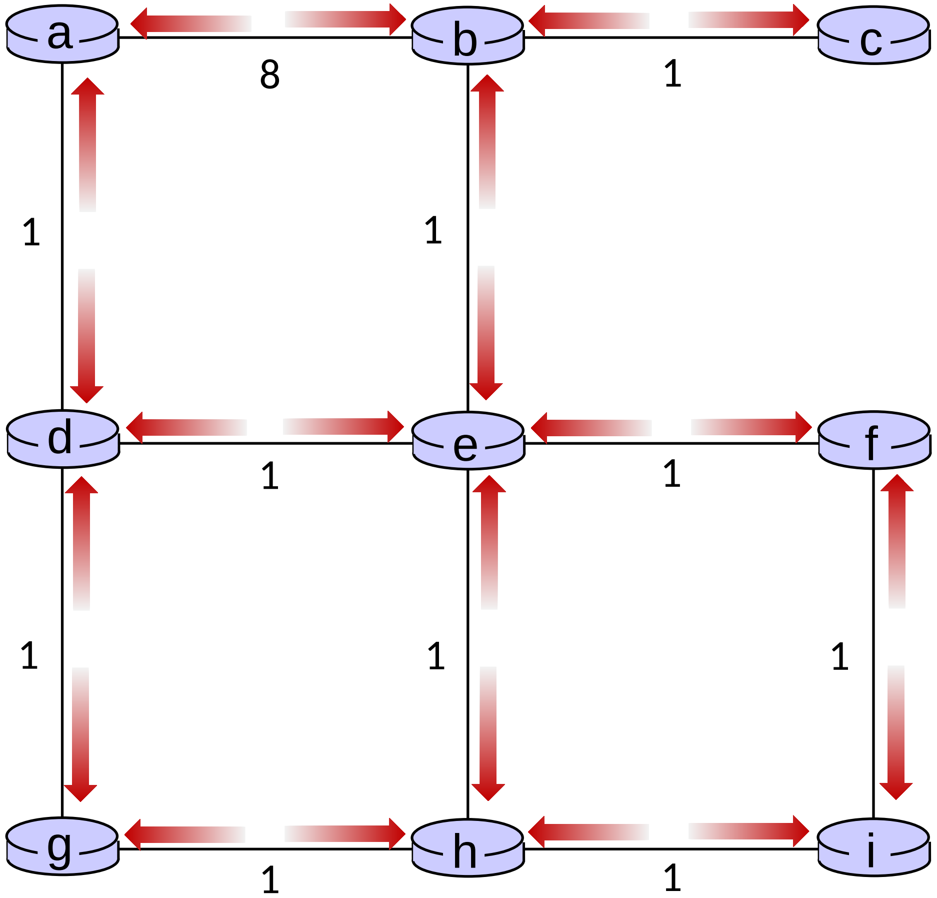
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



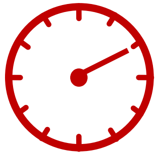
Distance vector example: iteration

t=2

All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



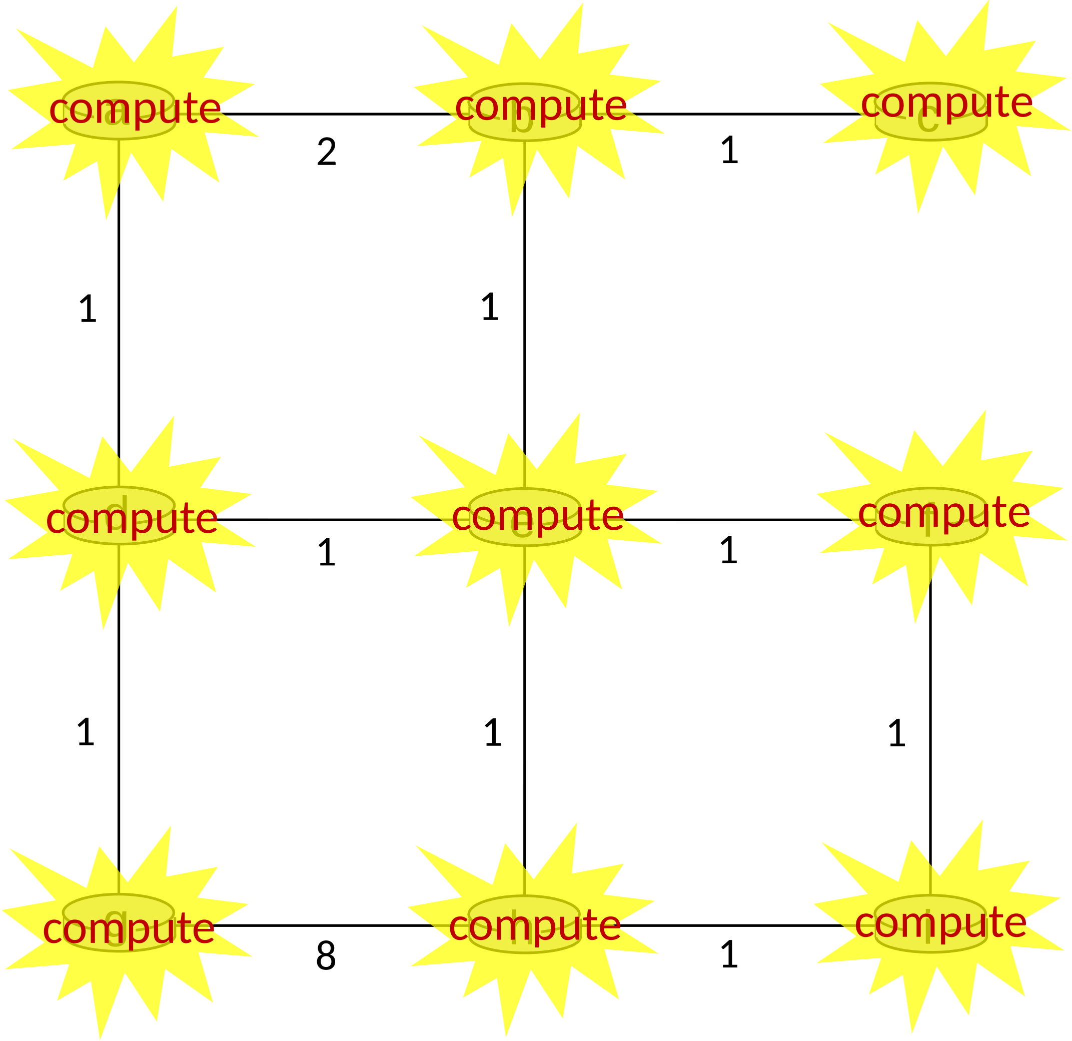
Distance vector example: iteration



t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

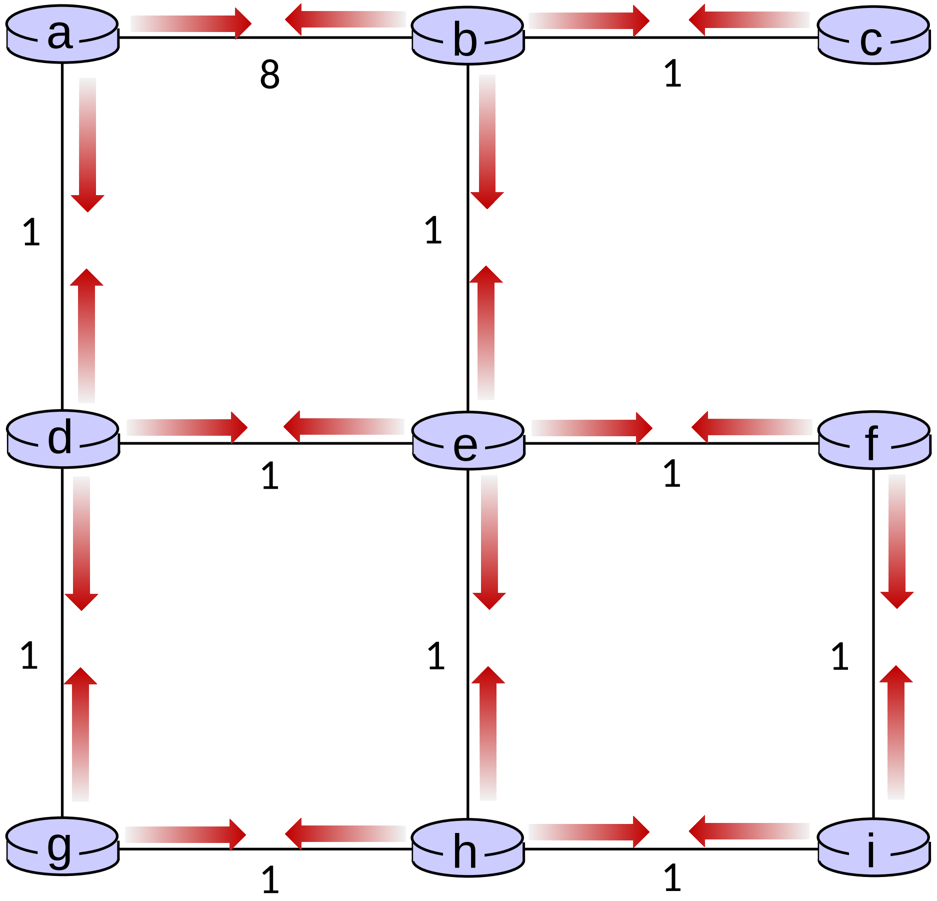


Distance vector example: iteration

t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

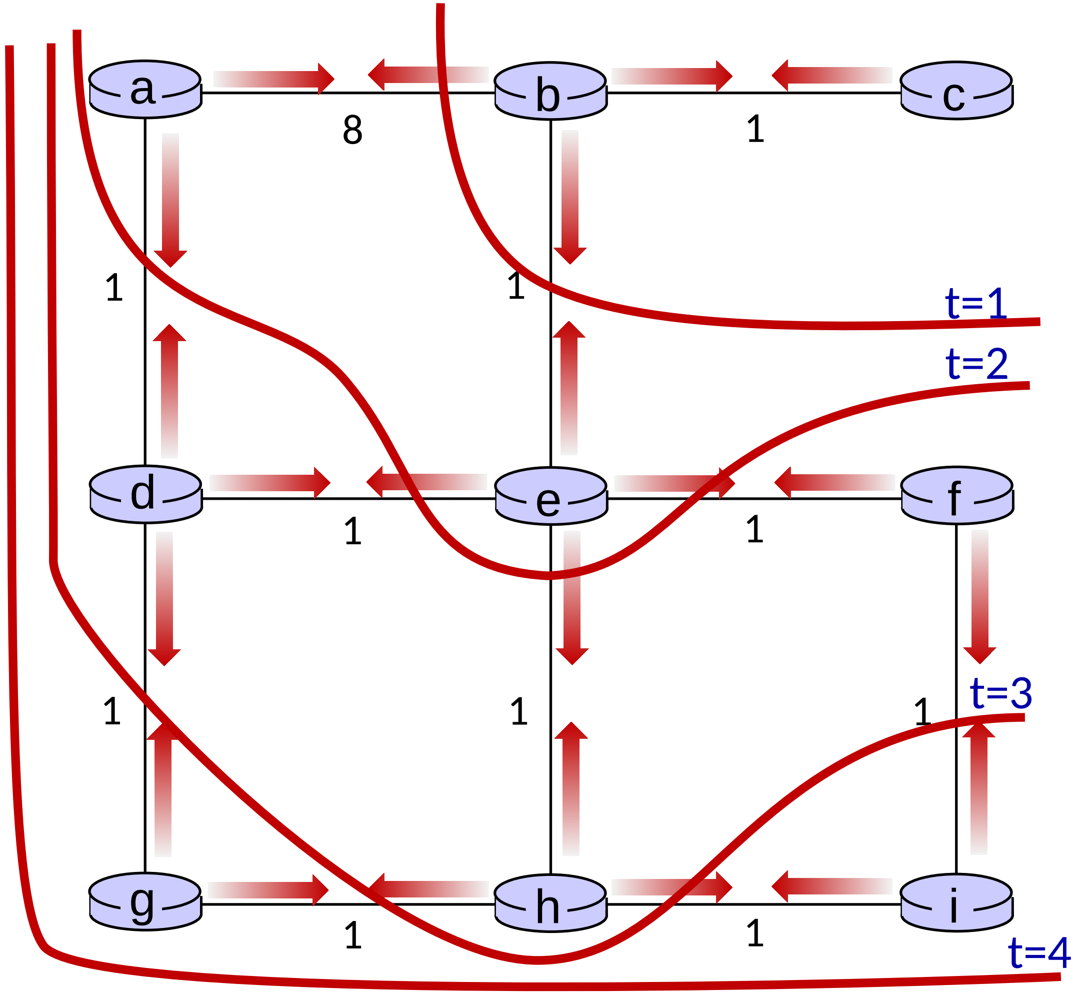
t=0 c's state at t=0 is at c only

t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b

t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well

t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at d, f, h

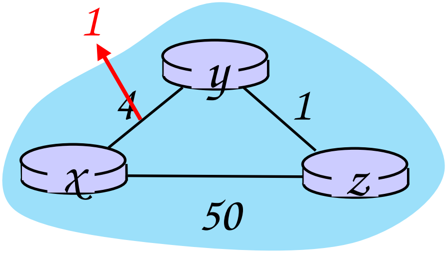
t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at g, i



Distance vector: link cost changes

link cost changes:

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



“good news travels fast”

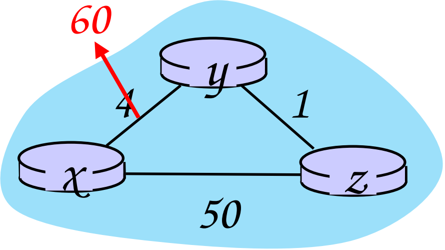
t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

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

t_2 : *y* receives *z*’s update, updates its DV. *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:
  - *y* sees direct link to *x* has new cost 60, but z has said it has a path at cost of 5. So *y* computes "my new cost to x will be 6, via z); notifies *z* of new cost of 6 to *x*.
  - *z* learns that path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y), notifies *y* of new cost of 7 to *x*.
  - *y* learns that path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y), notifies *z* of new cost of 8 to *x*.
  - *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y), notifies *y* of new cost of 9 to *x*.

    …
- see text for solutions. *Distributed algorithms are tricky!*

Comparison of LS and DV algorithms

message complexity

LS: *n* routers, O(n^2) messages sent

DV: exchange between neighbors; convergence time varies

speed of convergence

LS: O(n^2) algorithm, O(n^2) messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

LS:

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

DV:

- DV router can advertise incorrect *path* cost ("I have a *really* low-cost path to everywhere"): *black-holing*
- each router's DV is used by others: error propagate thru network

Making routing scalable

our routing study thus far - idealized

- all routers identical
- network "flat"

… not true in practice

scale: billions of destinations:

- can't store all destinations in routing tables!
- routing table exchange would swamp links!

administrative autonomy:

- Internet: a network of networks
- each network admin may want to control routing in its own network

Internet approach to scalable routing

aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

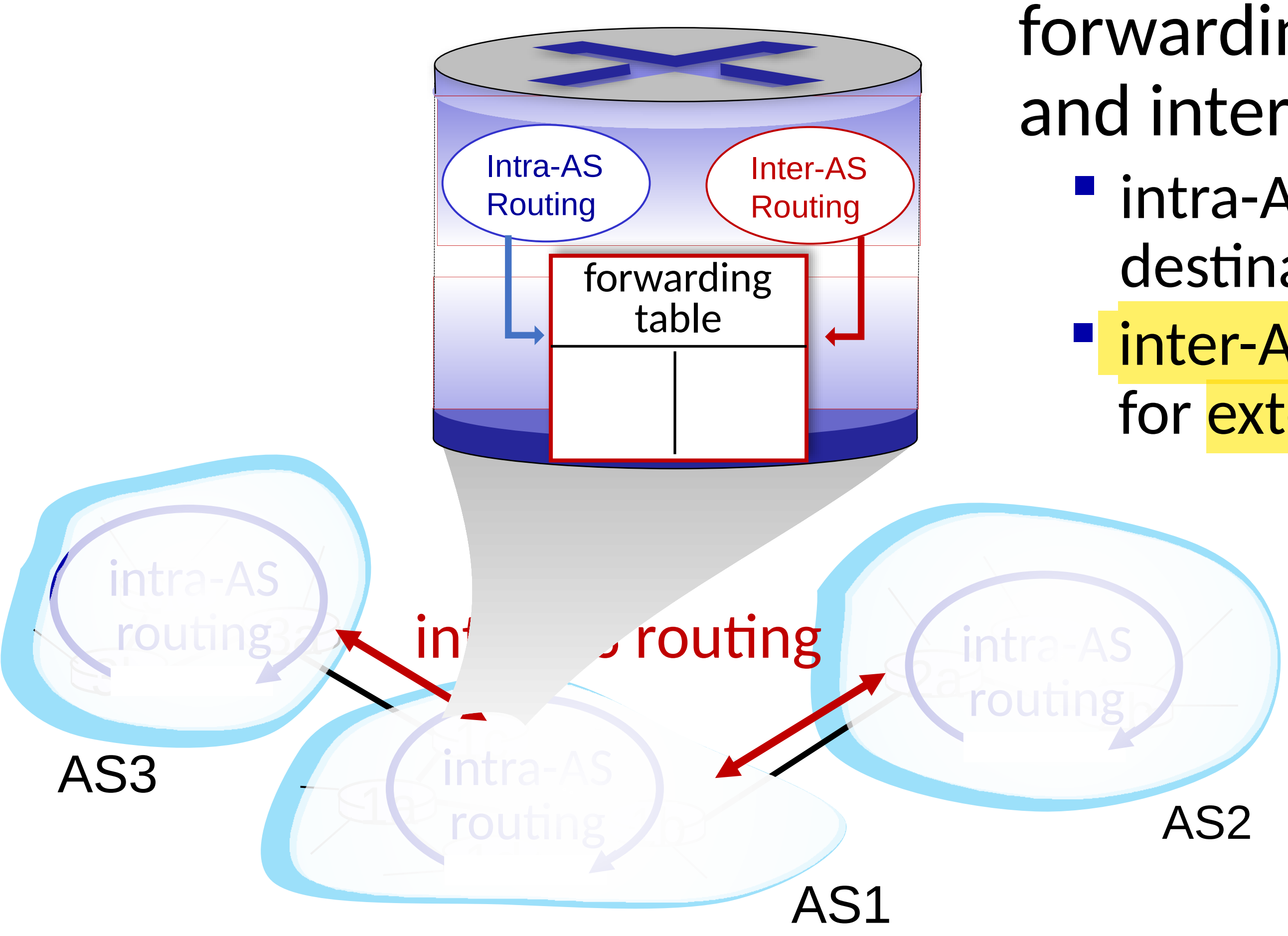
intra-AS (aka "intra-domain"): routing among routers *within same AS ("network")*

- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocols
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

inter-AS (aka "inter-domain"): routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

Interconnected ASes



forwarding table configured by intra- and inter-AS routing algorithms

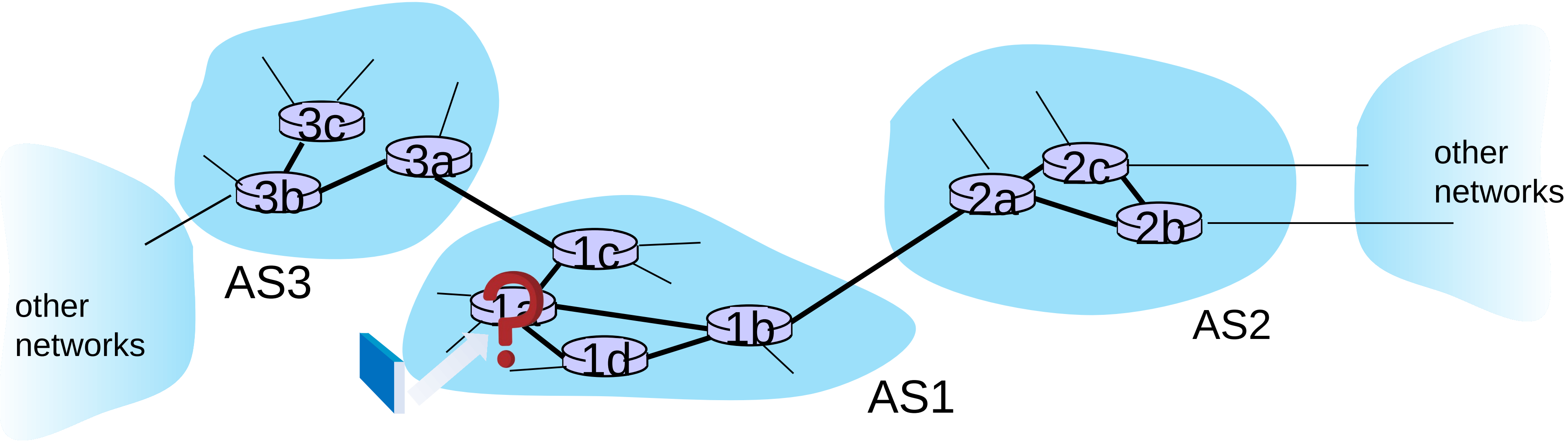
- intra-AS routing determine entries for destinations within AS
- inter-AS & intra-AS determine entries for external destinations

Inter-AS routing: a role in intradomain forwarding

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

AS1 inter-domain routing must:

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



Intra-AS routing: routing within an AS

most common intra-AS routing protocols:

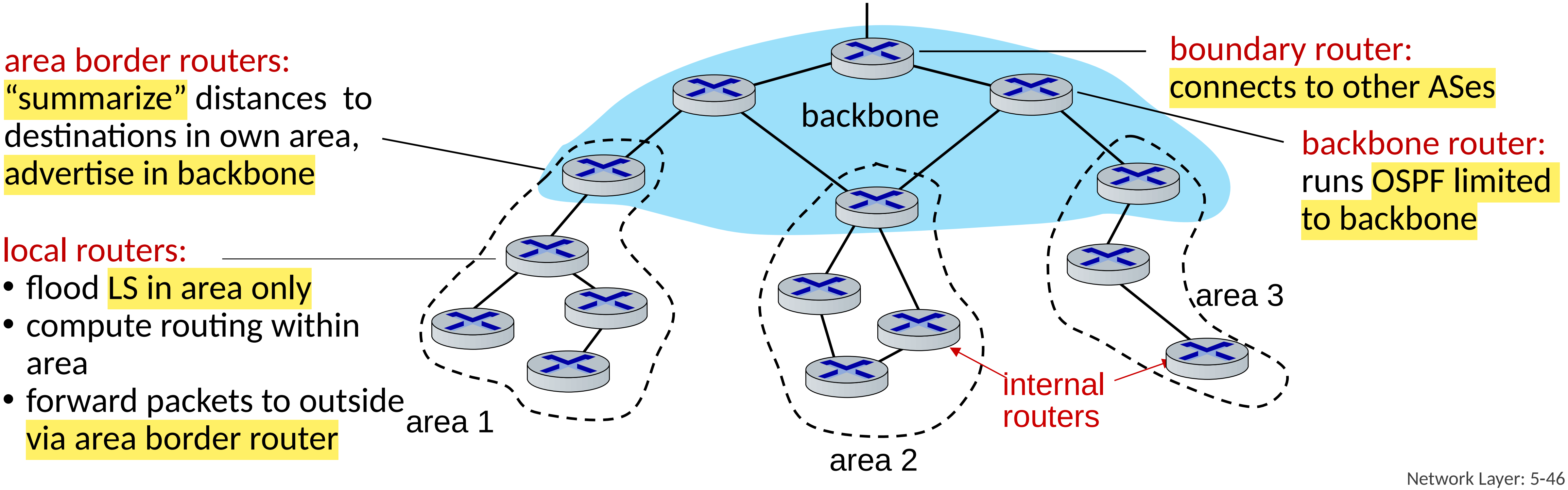
- RIP: Routing Information Protocol [RFC 1723]
  - classic DV: DVs exchanged every 30 secs
  - no longer widely used
- EIGRP: Enhanced Interior Gateway Routing Protocol
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- OSPF: Open Shortest Path First [RFC 2328]
  - link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

OSPF (Open Shortest Path First) routing

- "open": publicly available
- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra's algorithm to compute forwarding table
- *security:* all OSPF messages authenticated (to prevent malicious intrusion)

Hierarchical OSPF

- two-level hierarchy: local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations



area border routers:
"summarize" distances to destinations in own area, advertise in backbone

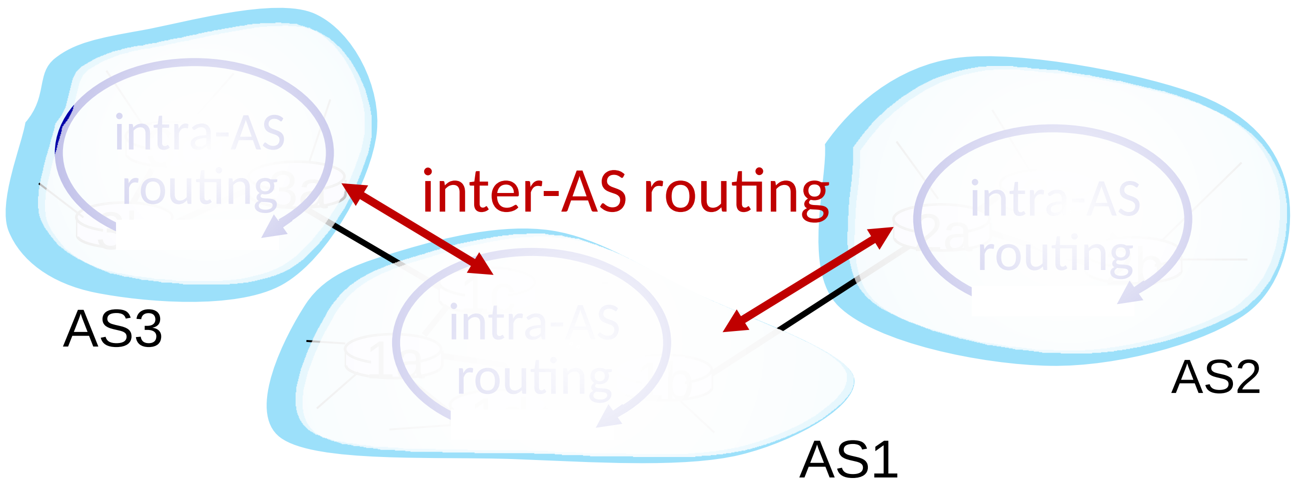
local routers:
- flood LS in area only
- compute routing within area
- forward packets to outside via area border router

boundary router:
connects to other ASes

backbone router:
runs OSPF limited to backbone

internal routers

Interconnected ASes



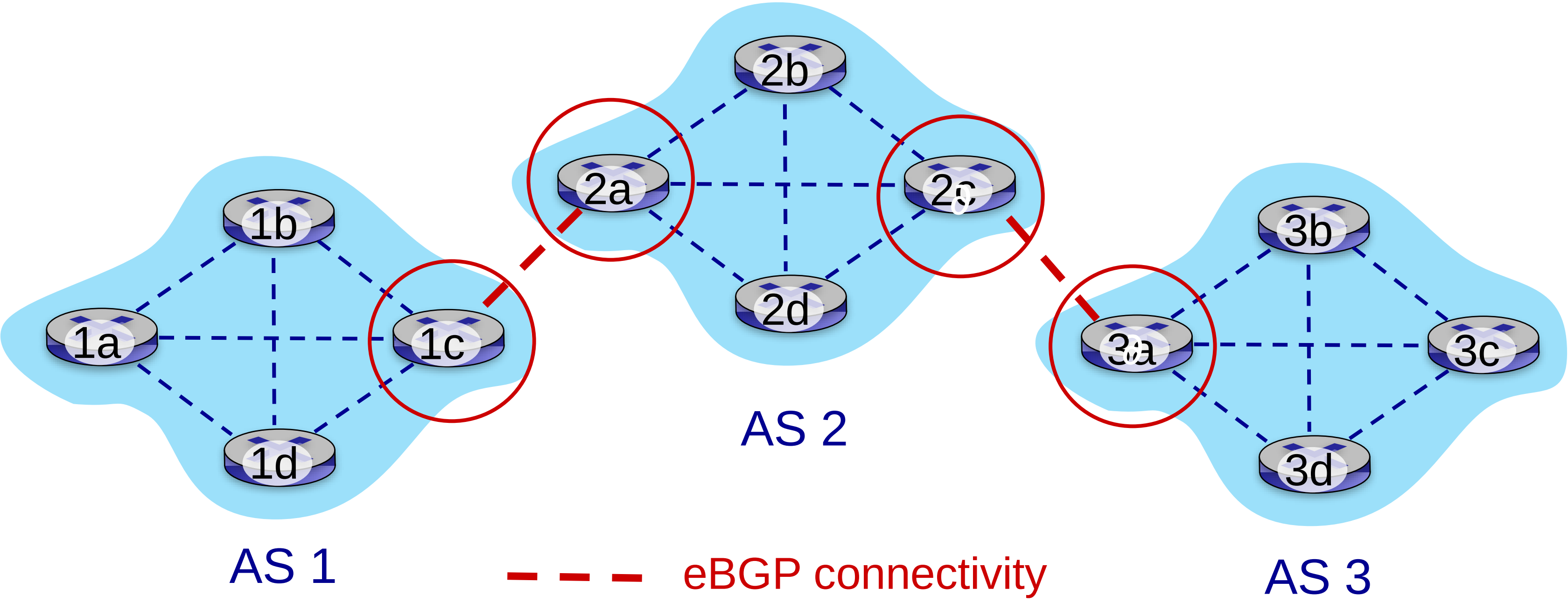
✓ intra-AS (aka "intra-domain"): routing among routers *within same AS ("network")*

➡ inter-AS (aka "inter-domain"): routing *among* AS'es

Internet inter-AS routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *"I am here, here is who I can reach, and how"*
- BGP provides each AS a means to:
  - obtain destination network reachability info from neighboring ASes (eBGP)
  - determine routes to other networks based on reachability information and *policy*
  - propagate reachability information to all AS-internal routers (iBGP)
  - advertise (to neighboring networks) destination reachability info

eBGP, iBGP connections



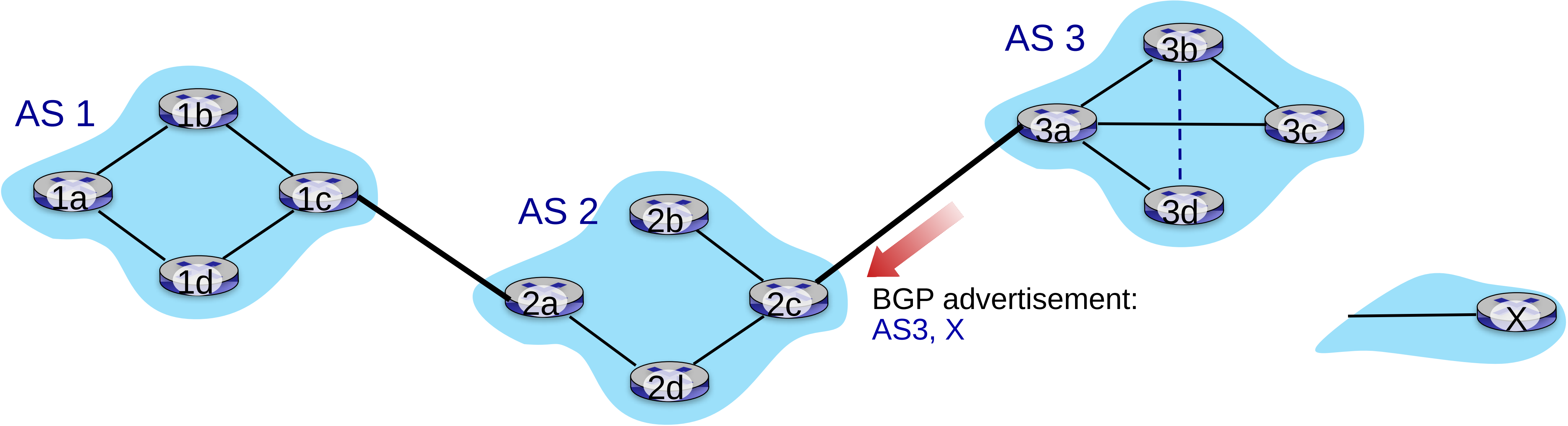
▬ ▬ ▬ eBGP connectivity

- - - - - - logical iBGP connectivity

gateway routers run both eBGP and iBGP protocols

BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway 3a advertises path AS3,X to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



BGP advertisement:
AS3, X

Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

scale:

- hierarchical routing saves table size, reduced update traffic

performance:

- intra-AS: can focus on performance
- inter-AS: policy dominates over performance