

CS5222 Computer Networks and Internets

Network Layer (Control Plane)

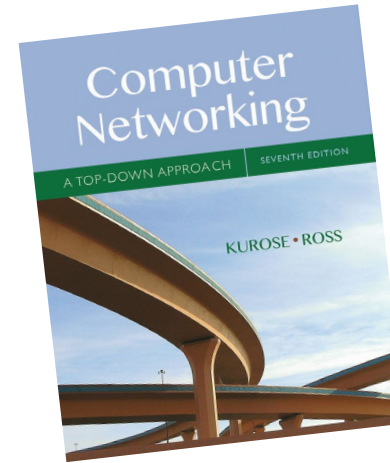
Prof Weifa Liang

Weifa.liang@cityu-dg.edu.cn

Slides based on book *Computer Networking: A Top-Down Approach*.

Network layer: “control plane” roadmap

- introduction
- routing protocols
 - link state
 - distance vector
- intra-ISP routing: RIP & OSPF
- routing among ISPs: BGP



Chapter 5

Network-layer functions

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

data plane

- **routing**: determine route/path taken by packets from source to destination

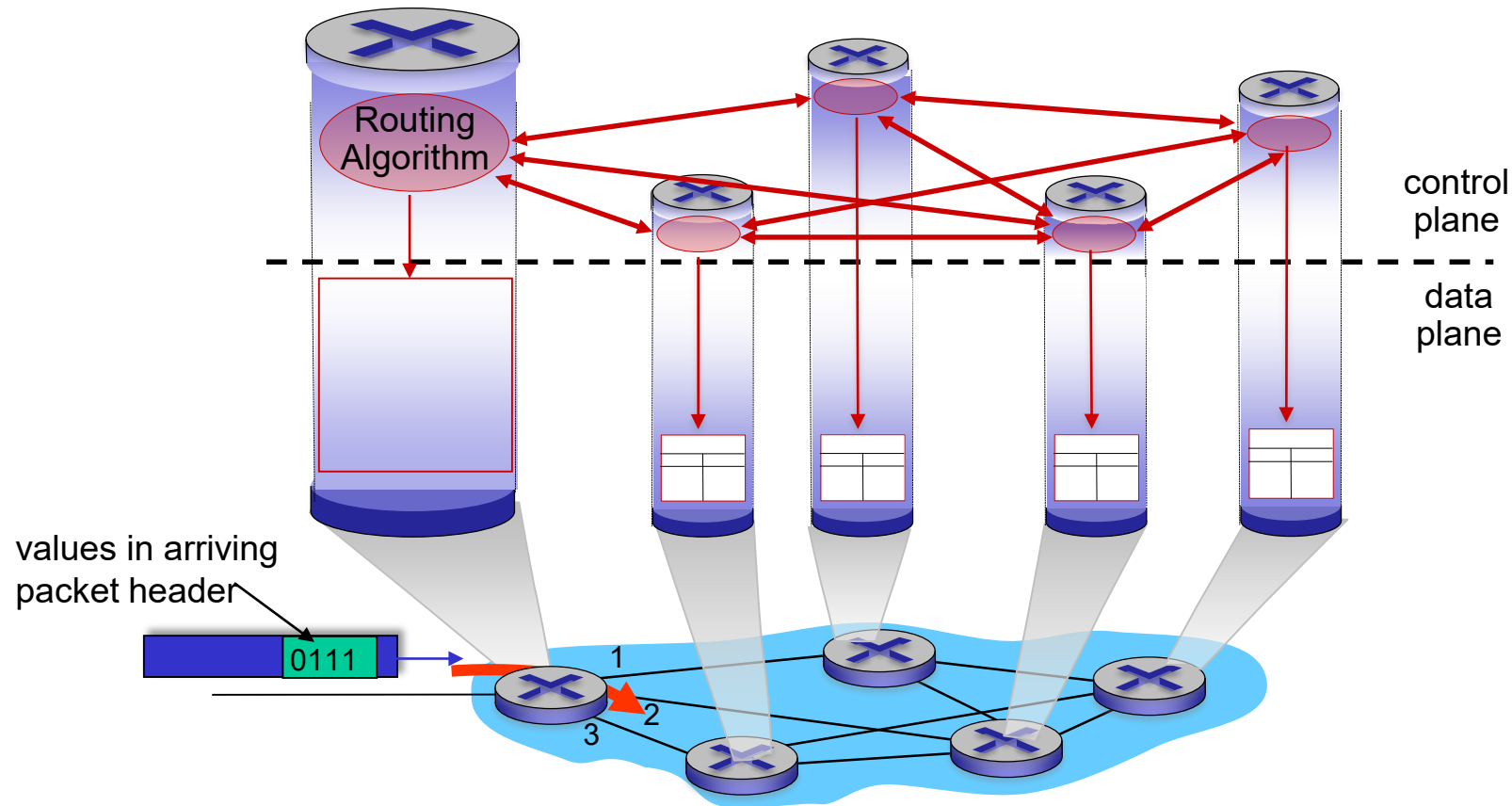
control plane

Two approaches to structuring network control plane:

- per-router control (traditional or table-based routing)
- logically centralized control (software-defined networking or SDN)

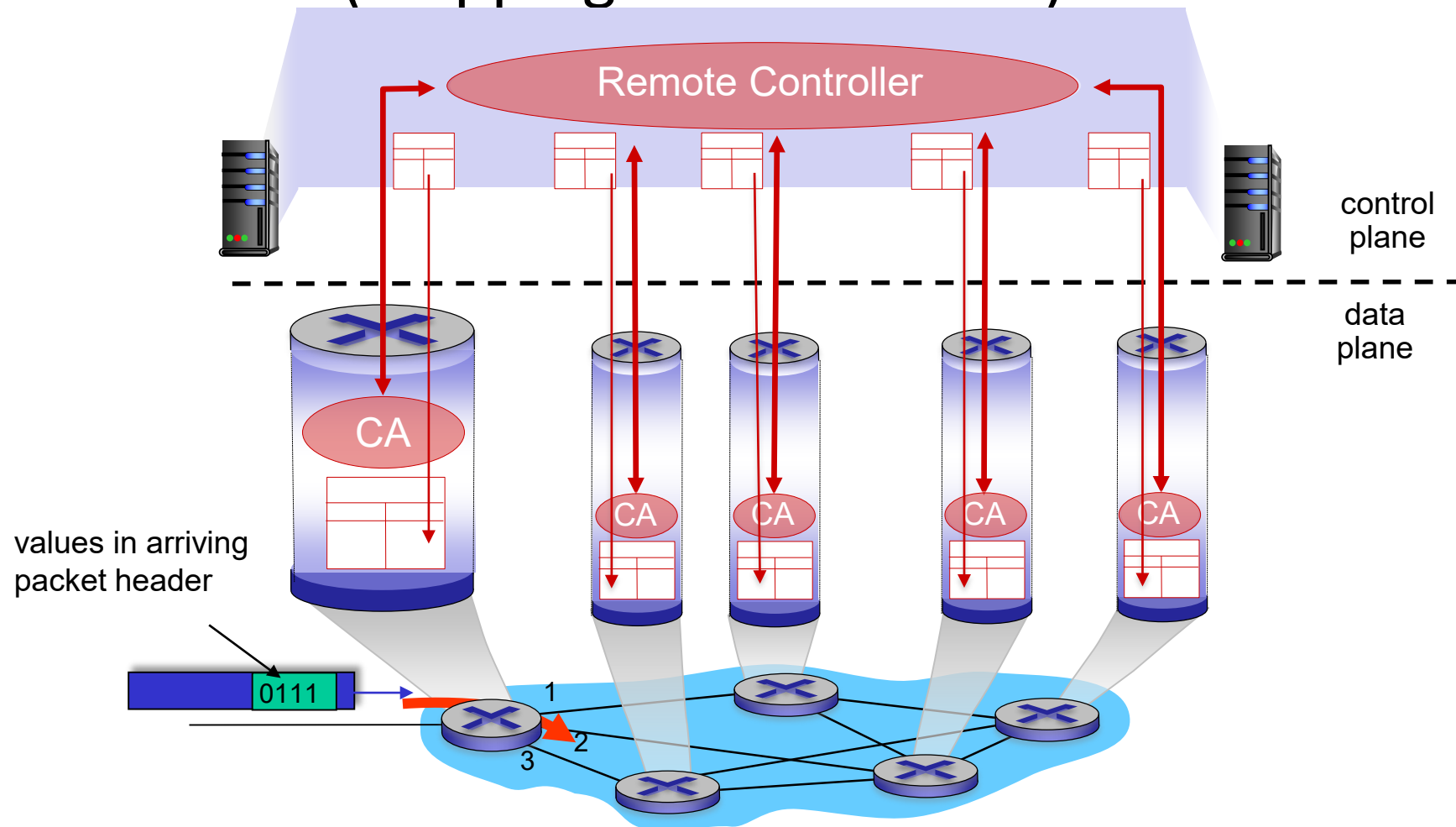
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



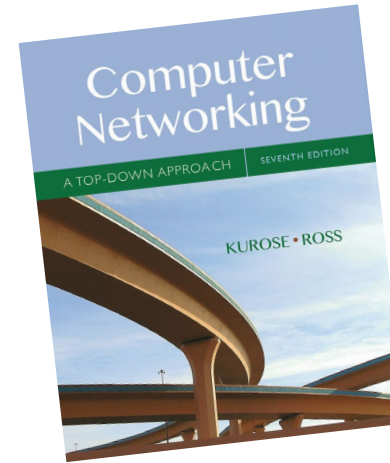
Software-Defined Networking (SDN) control plane

(Remote) controller computes, installs forwarding tables in routers
(mapping to virtual links)



Network layer: “control plane” roadmap

- introduction
- routing protocols
 - link state
 - distance vector
- intra-ISP routing: RIP & OSPF
- routing among ISPs: BGP

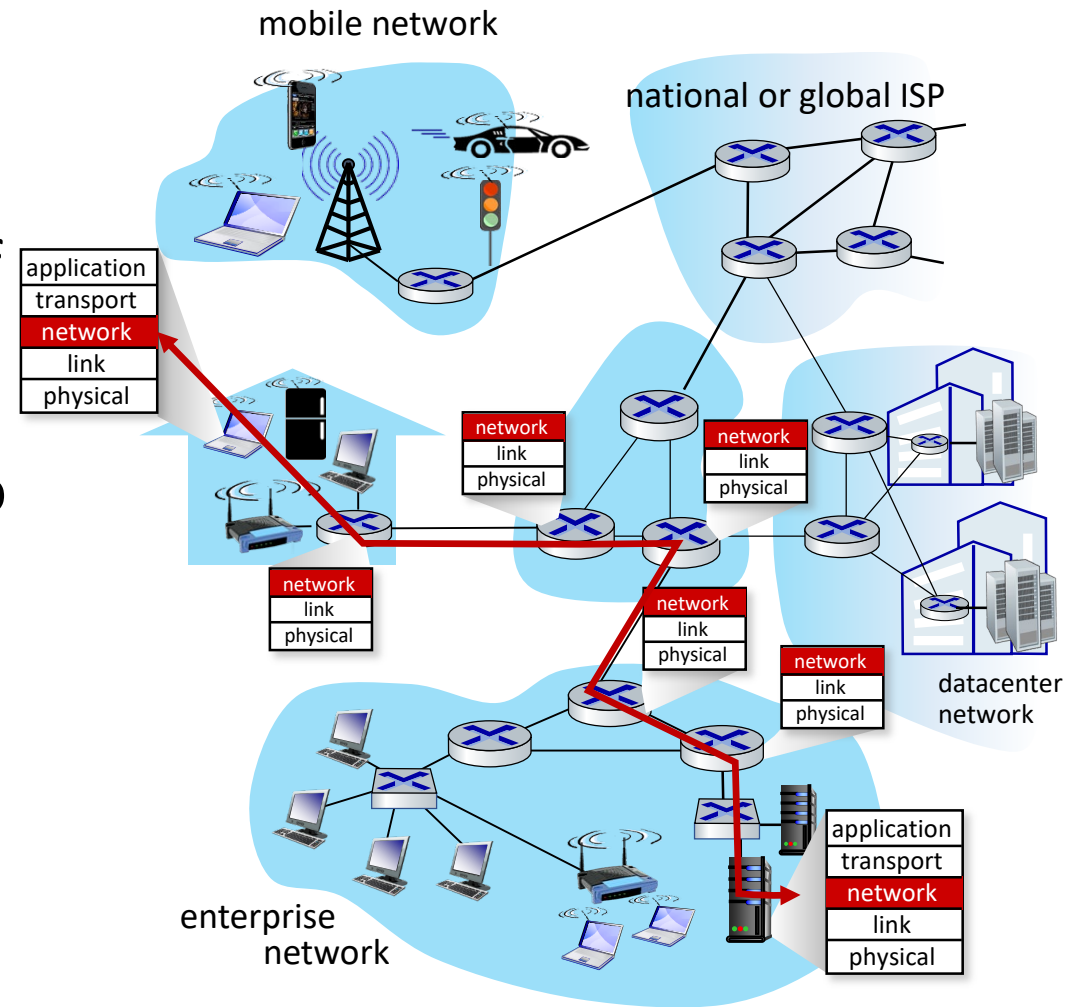


Chapter 5

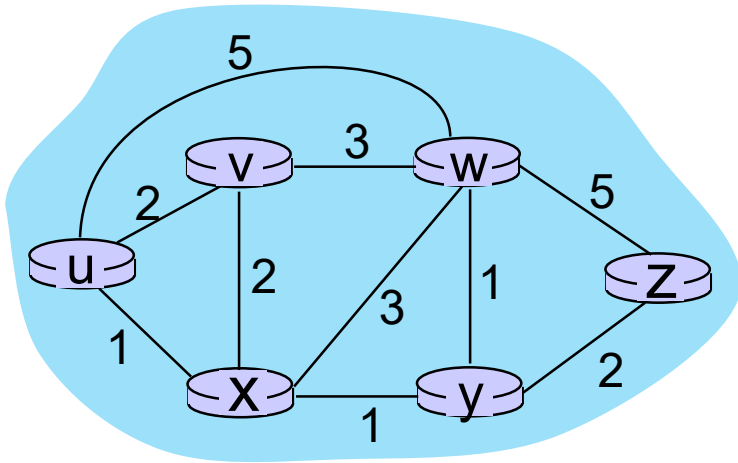
Routing protocols

Routing protocol goal: determine “good” paths (equivalently, routes), from sending host 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 challenges!
 - CISCO (search for them)



Graph abstraction: link costs



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

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

cost defined by network operator:
could always be 1 (**hop**), or inversely
related to bandwidth (**rate**), or
positively related to congestion (**delay**)

graph: $G = (N, E)$

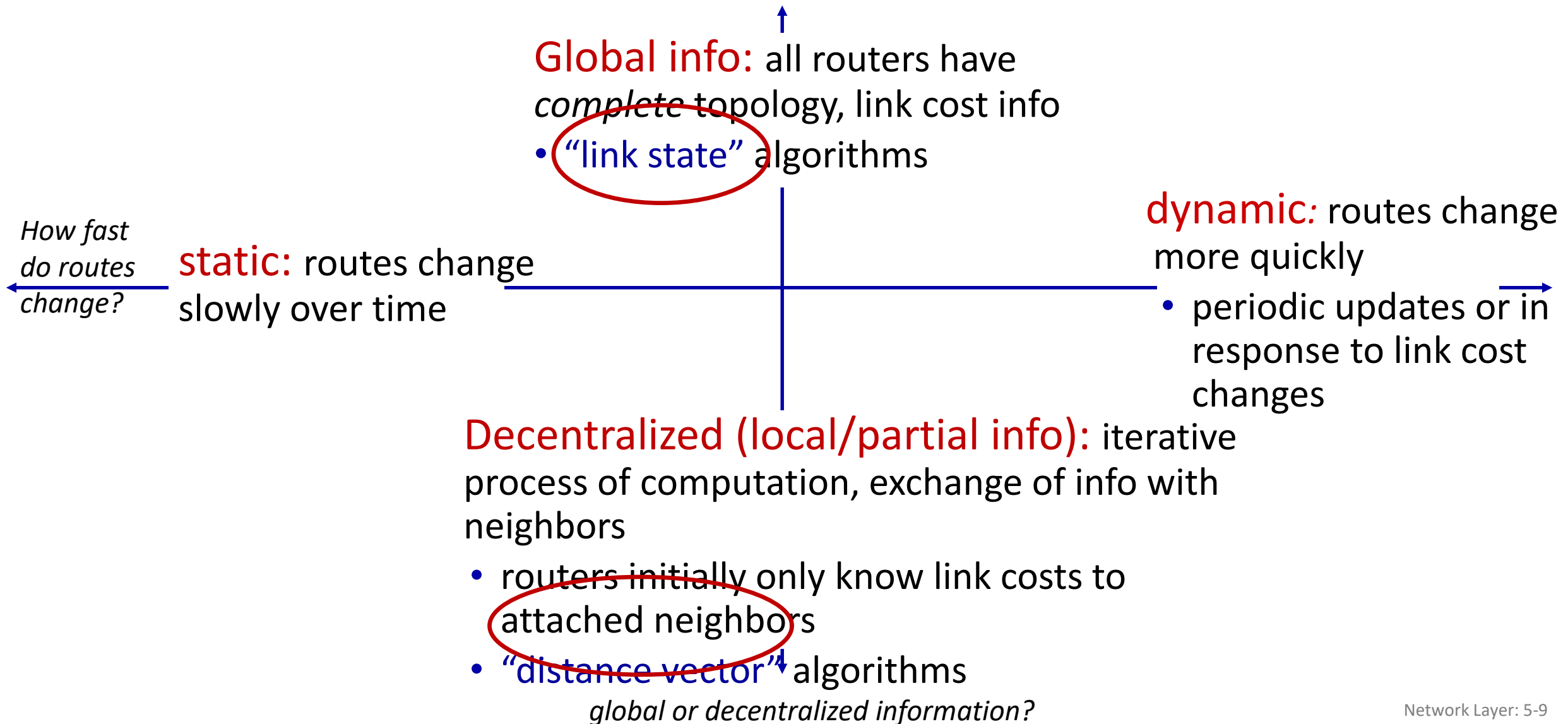
N : set of routers = $\{u, v, w, x, y, z\}$, **vertices**

E : set of links = $\{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$, **edges**

key question: what is the least-cost path between u and z ?

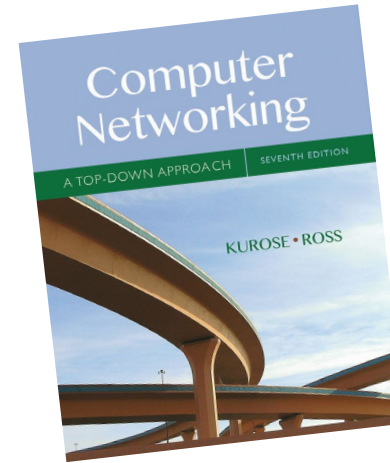
routing algorithm: algorithm that finds the least cost path

Routing algorithm classification



Network layer: “control plane” roadmap

- introduction
- routing protocols
 - link state
 - distance vector
- intra-ISP routing: RIP & OSPF
- routing among ISPs: BGP



Chapter 5

Dijkstra's link-state routing algorithm

- Starting from the **source**!
- Basic idea: packet should come to you from your neighbor, and hence check the best neighbor toward you!
- **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
 - construct *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 ; $= \infty$ if not neighbors
- $D(v)$: *current* estimate of cost of **least-cost-path from source to 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

1 *Initialization:*

2 $N' = \{u\}$ /* compute least cost path from u to all other nodes */

3 for all nodes v

4 if v adjacent to u /* u initially knows direct-path-cost only to direct neighbors */

5 then $D(v) = c_{u,v}$ /* but may not be *minimum* cost! */

6 else $D(v) = \infty$

7



8 *Loop*

9 find w not in N' such that $D(w)$ is a minimum

10 $N' = N' + \{w\}$ --- add w to N'

11 update $D(v)$ for all v adjacent to w and not in N' :

12 $D(v) = \min \{ D(v), D(w) + c_{w,v} \}$ /* check whether cost can be improved via w

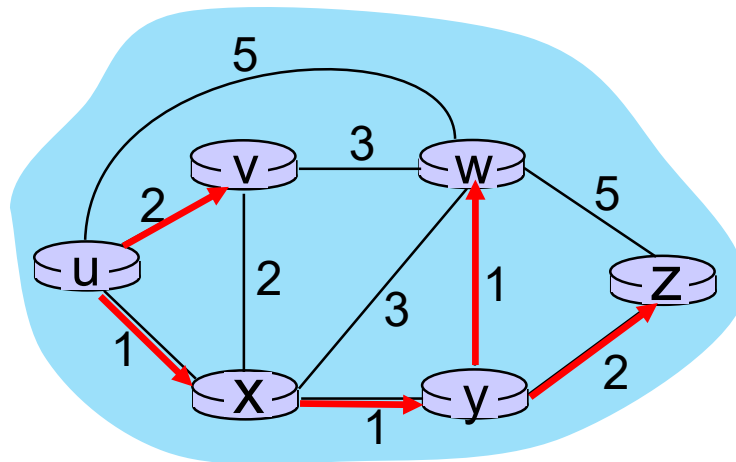
13 /* new least-path-cost to v is either old least-cost-path to v or known

14 least-cost-path to w plus the link cost from neighbor w to v */

15 *until all nodes in N'*

Dijkstra's algorithm: an 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					



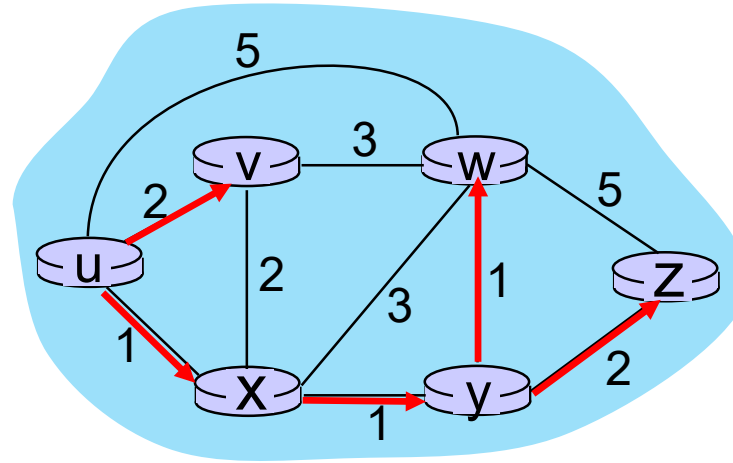
Initialization (step 0): For all a : if a adjacent to u , then $D(a) = c_{u,a}$

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

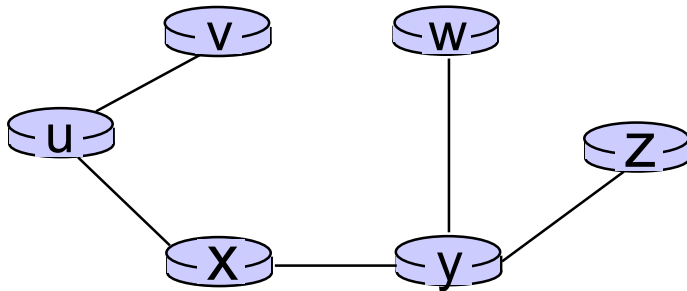
update $D(b)$ for all b adjacent to a and not in N' :

$$D(b) = \min \{ D(b), D(a) + c_{a,b} \}$$

Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

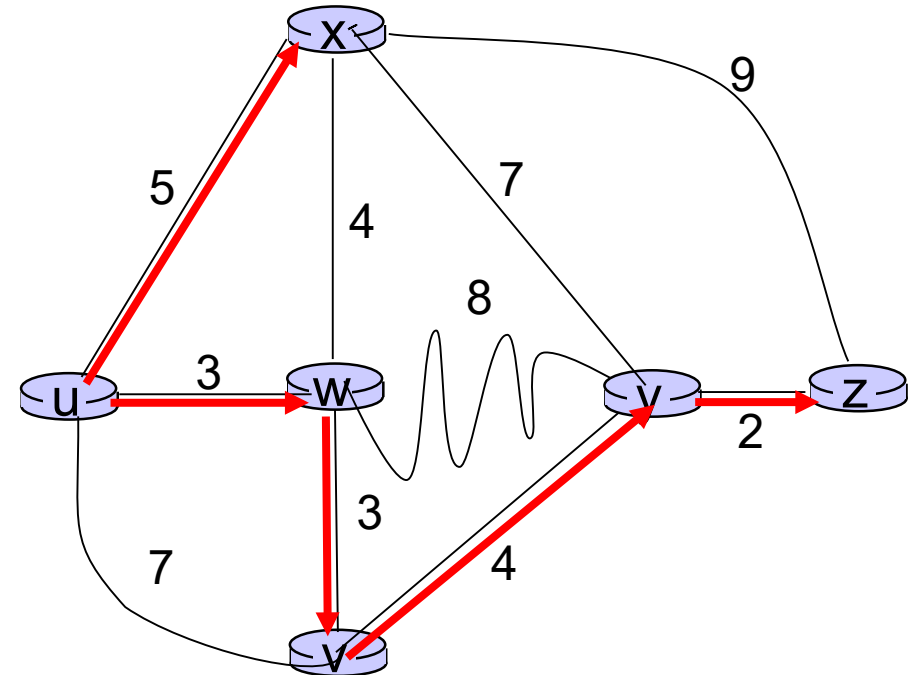
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

route from *u* to *v* directly

route from *u* to all other destinations via *x* (the **next hop**)

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	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwvx	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 (operations needed for convergence): n nodes

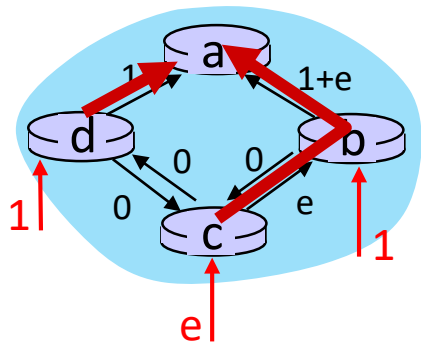
- each of $n-1$ iterations: need to check all nodes, w , not in N'
- At most $n(n-1)$ comparisons: $O(n^2)$ time complexity
- more efficient implementations possible: $O(m \log n)$ where m is the number of links with $m=O(n^2)$.

message complexity (the messages to get global info):

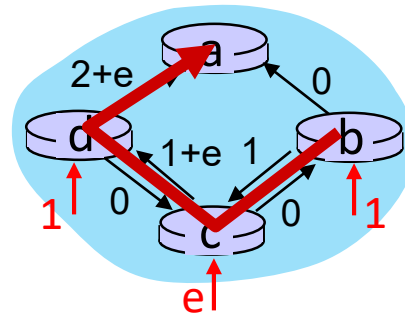
- each router must *broadcast* its link state information to other $(n-1)$ routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

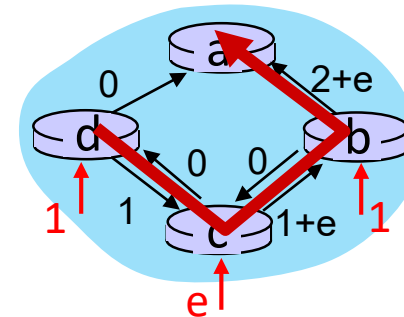
- Dijkstra's algorithm applies to stationary networks. Whenever topology or traffic changes, it has to be rerun!
- when link costs depend on traffic volume, **route oscillations** possible
- **sample scenario:**
 - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



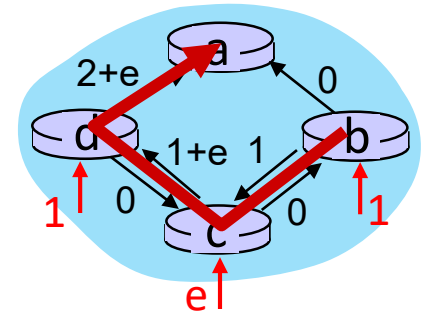
initially



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



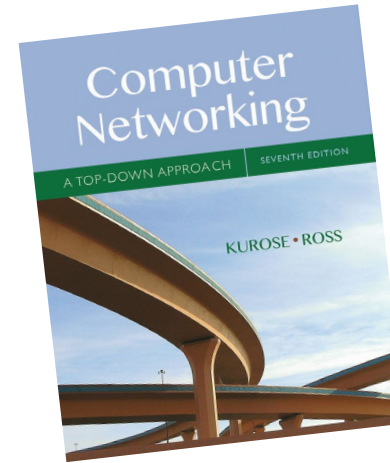
given these costs,
find new routing....
resulting in new costs



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: RIP & OSPF
- routing among ISPs: BGP



Chapter 5

Distance vector algorithm (centralized version)

- Global info is needed, i.e., each node knows all link costs just as in Dijkstra's algorithm
- Trace back from destination: [algorithm starts from destination!](#)
- Basic idea: if my neighbor tells me the cost to the destination, use it and figure out the shortest from me to the destination!
- Based on [Bellman-Ford](#) (BF) equation:

Bellman-Ford equation

Let $D_x(d)$: cost of least-cost path from x to *destination* d .

Then:

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

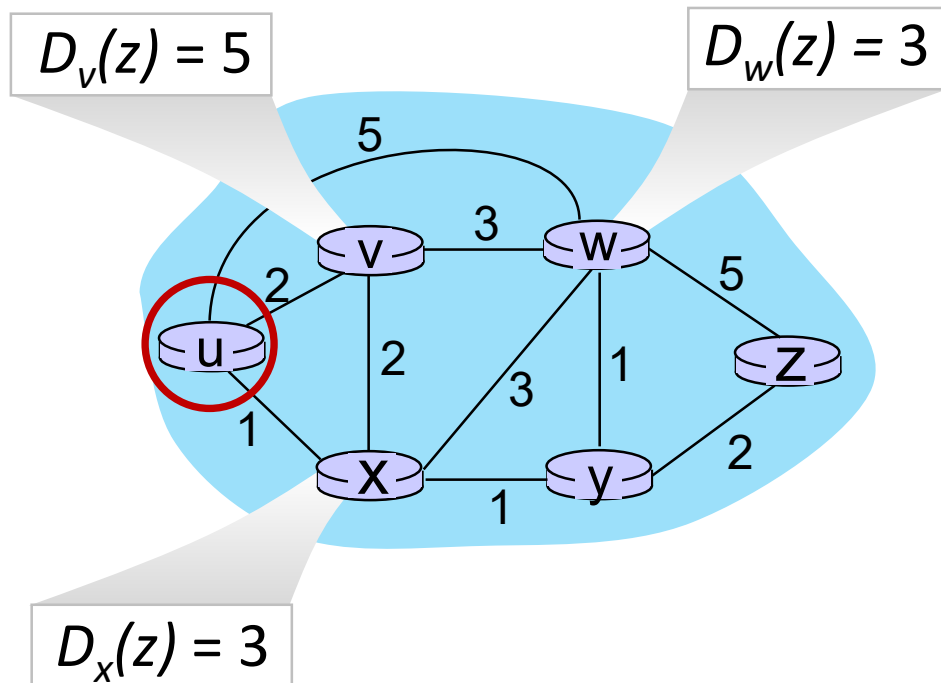
v 's estimated least-cost-path cost to d

cost of link from x to v

min taken over all neighbors v of x ([I have to go through one of my neighbors to destination](#))

Bellman-Ford Example

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



Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,x} + D_x(z), \\ &\quad c_{u,w} + D_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

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

Distributed Distance vector algorithm

key idea:

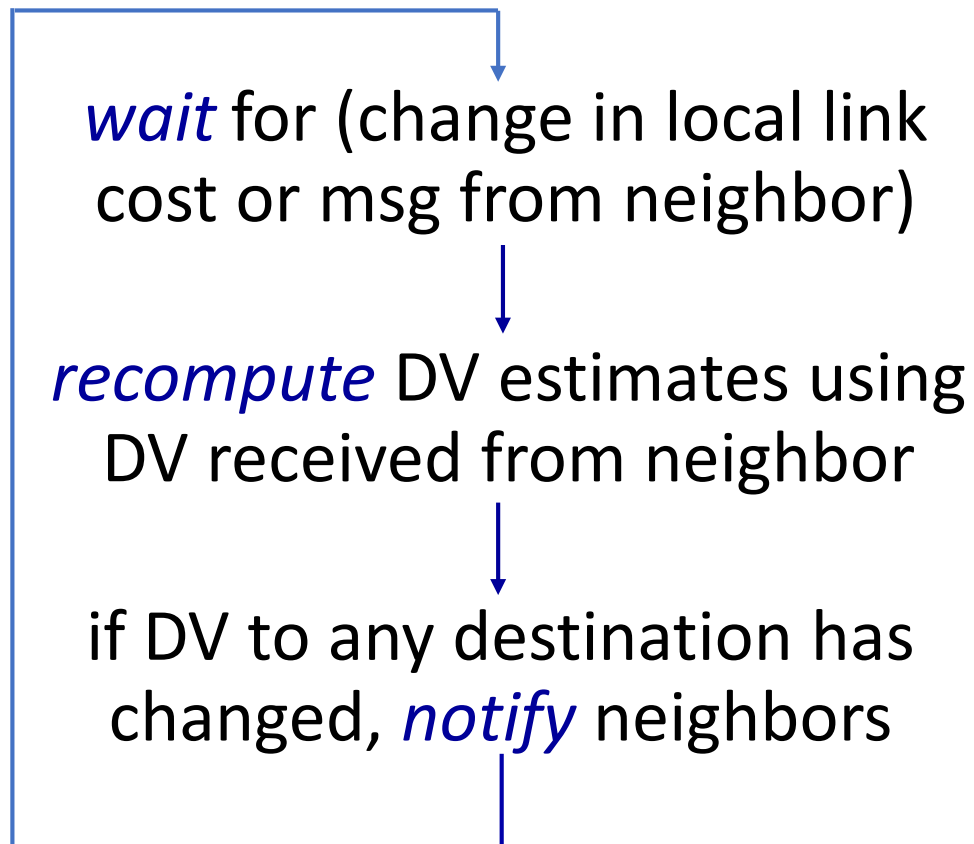
- from time-to-time, each node sends its own distance vector (DV) estimate to neighbors (this is the version of distributed version)
- when x receives a new DV estimate from any neighbor, it updates its own DV using B-F equation:

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

- under natural conditions, the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$

Distributed Distance vector algorithm:

each node:



iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor(s)

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

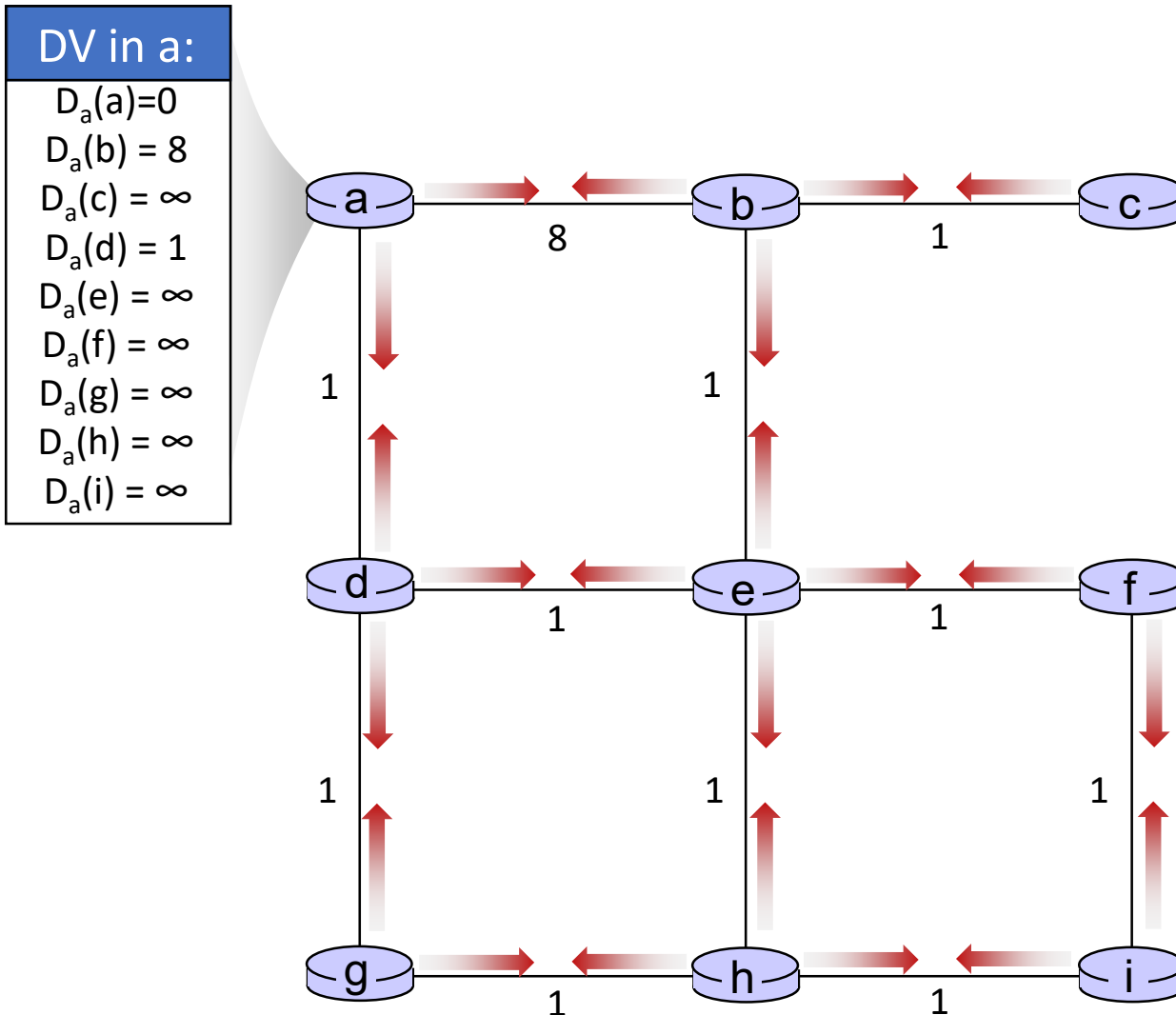
- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

Distance vector: example



t=0

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



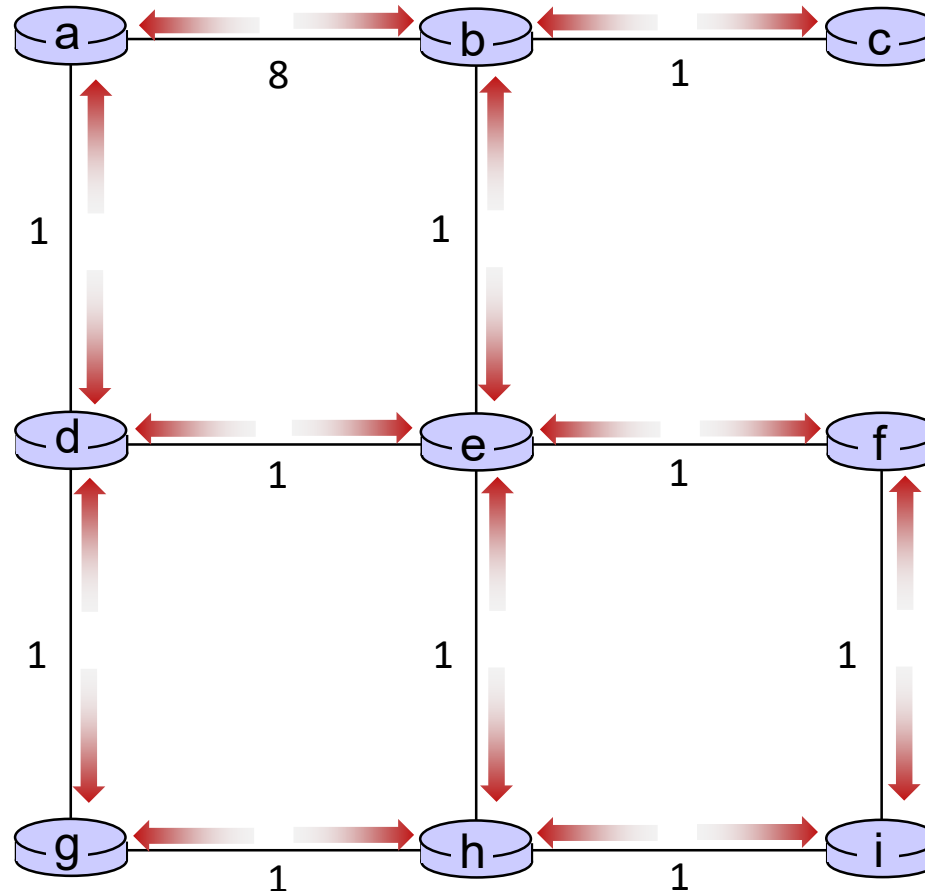
Distance vector example: iteration



$t=1$

All nodes:

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



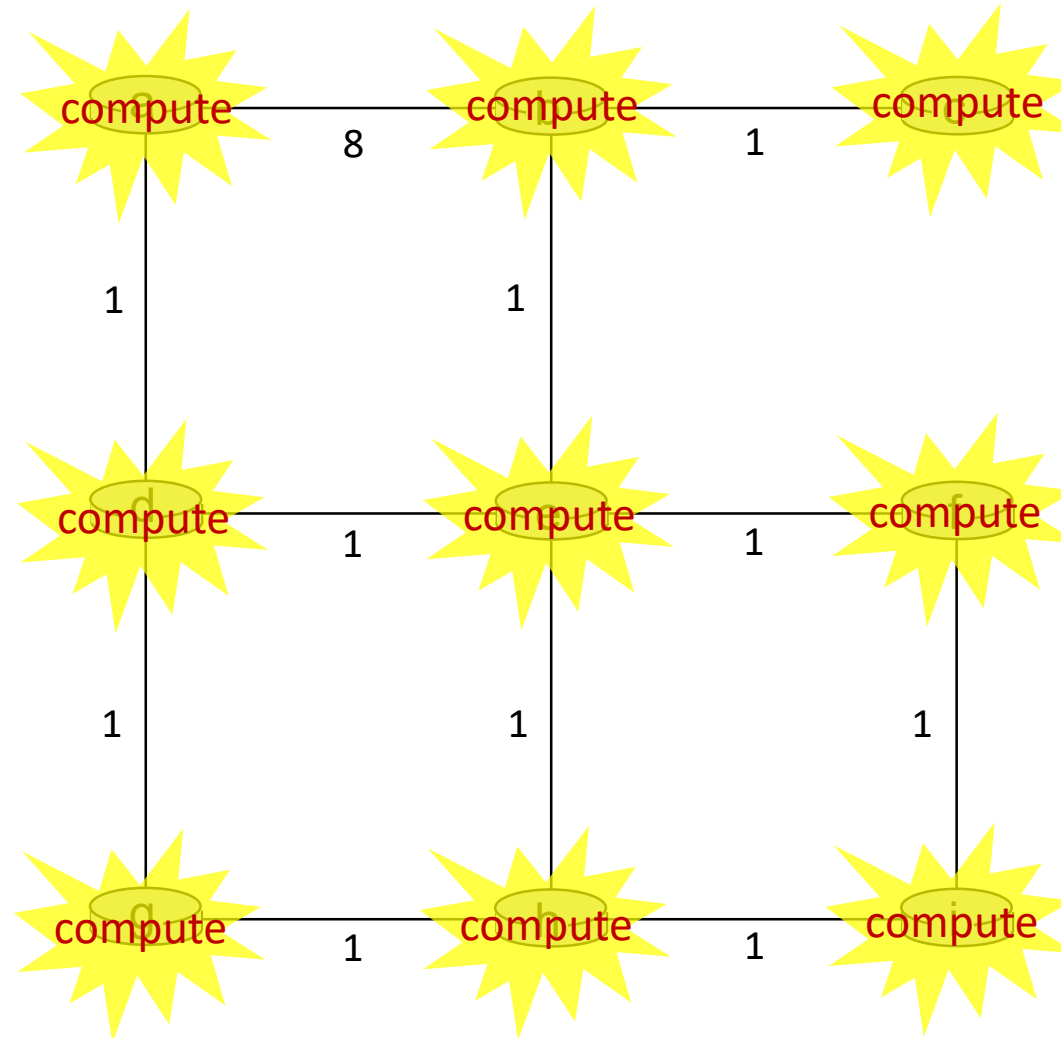
Distance vector example: iteration



t=1

All nodes:

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



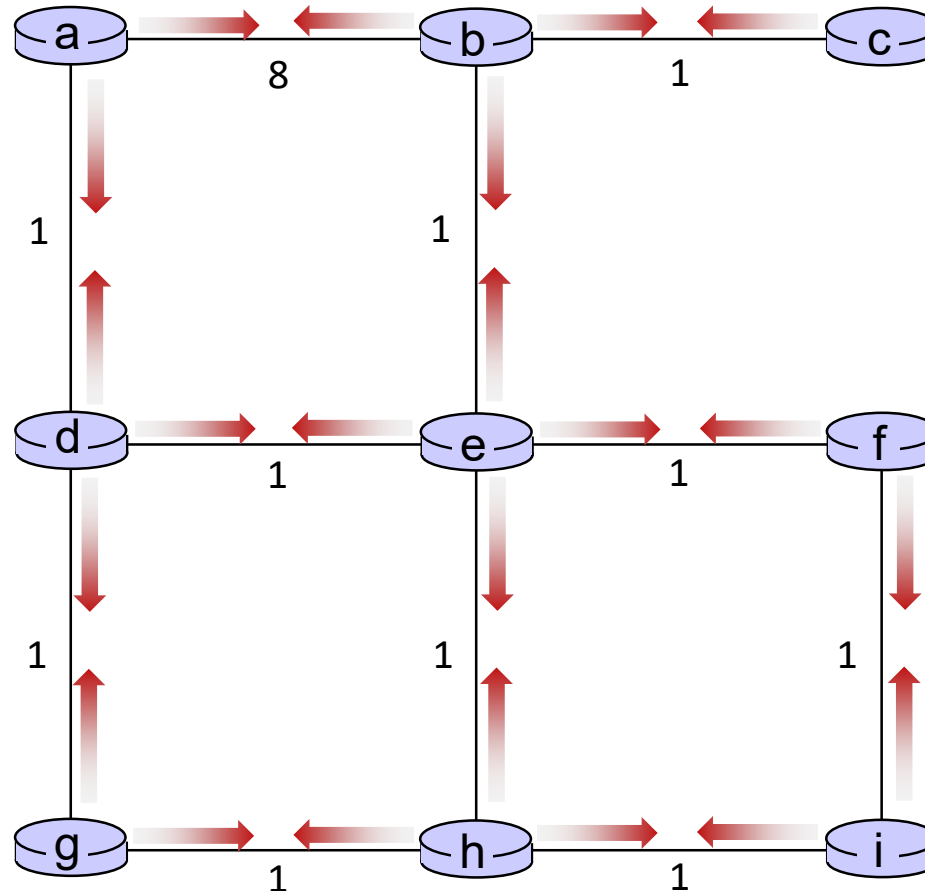
Distance vector example: iteration



t=1

All nodes:

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



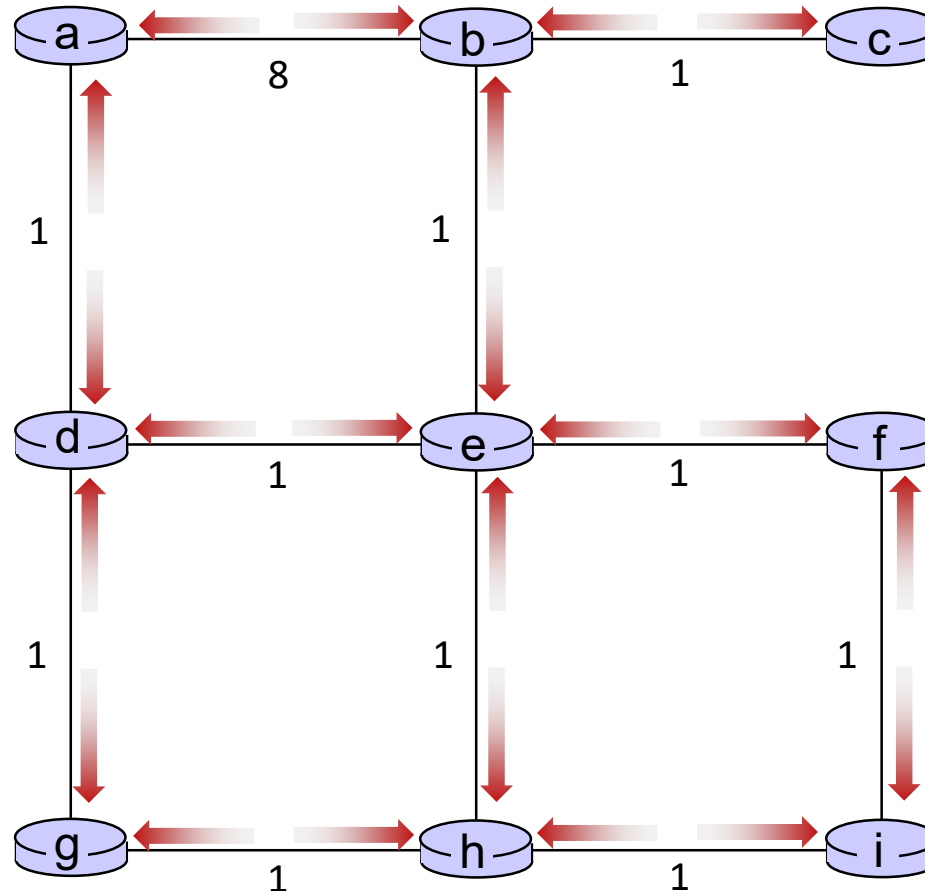
Distance vector example: iteration



$t=2$

All nodes:

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



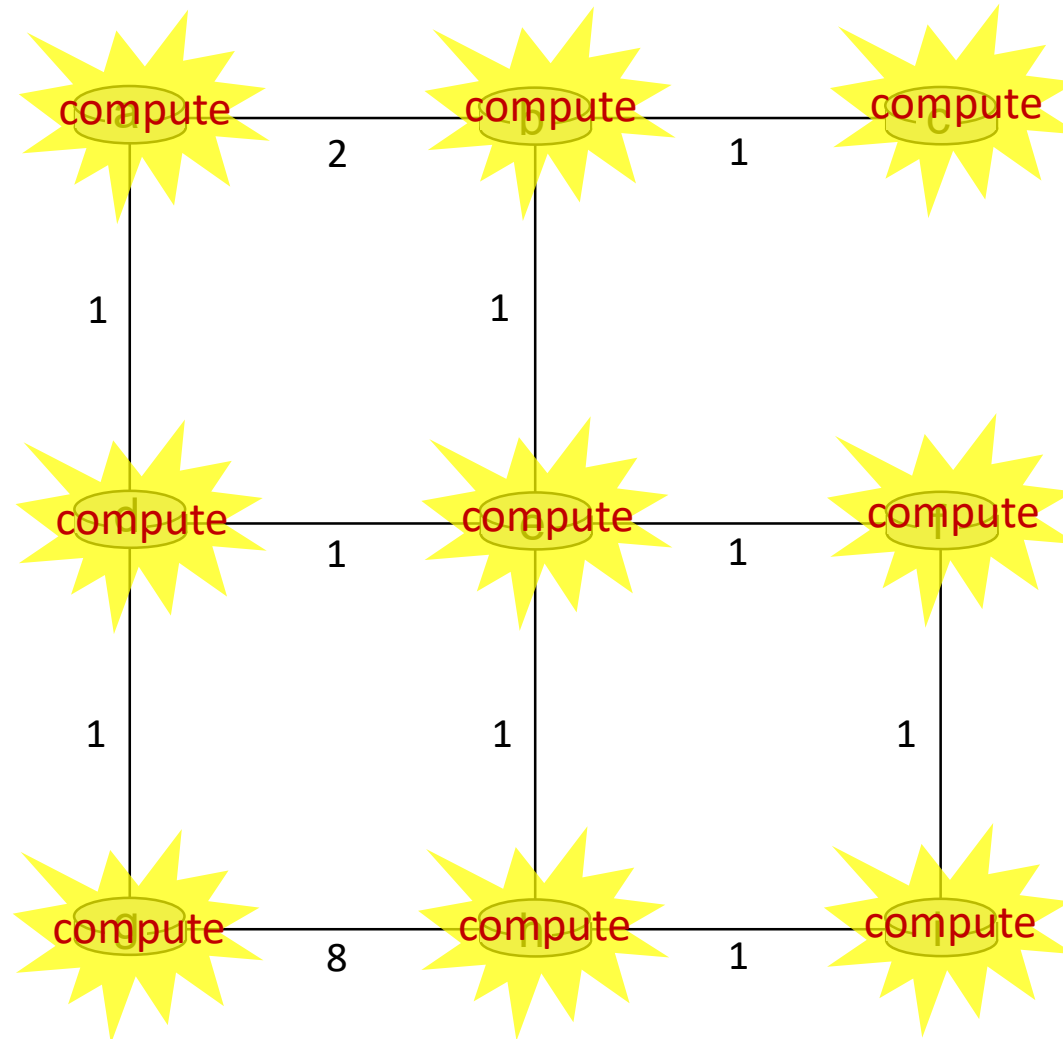
Distance vector example: iteration



t=2

All nodes:

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



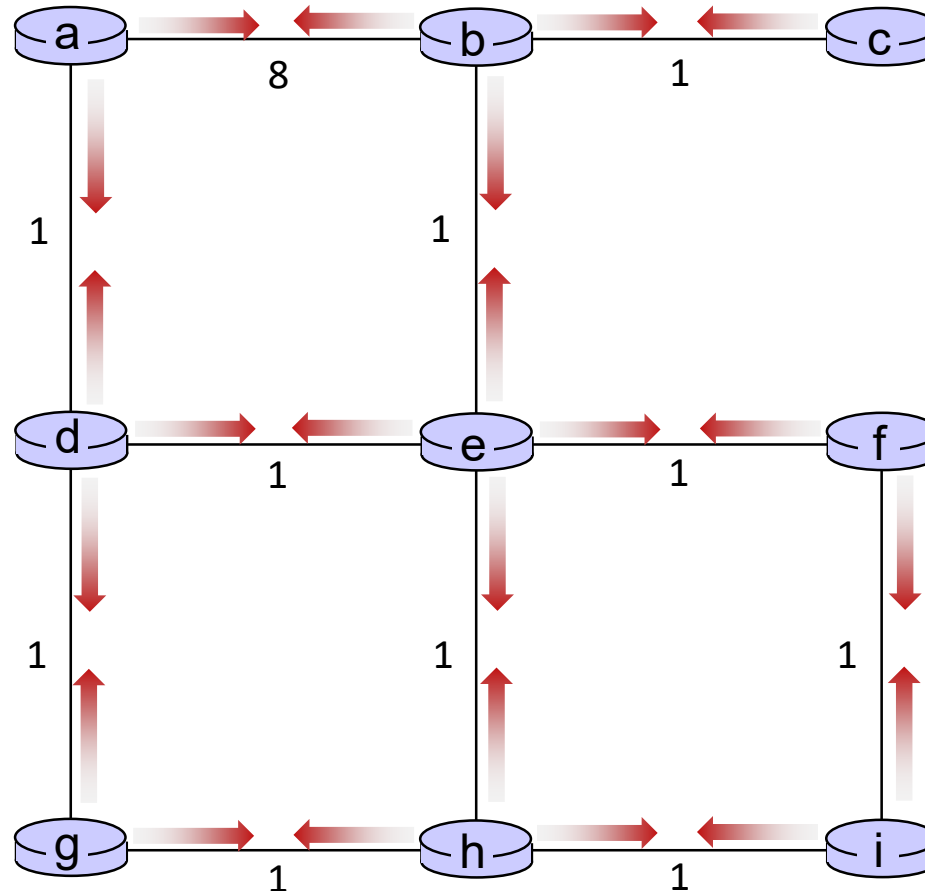
Distance vector example: iteration



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 example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

Distance vector example:



t=1

- b receives DVs from a, c, e

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

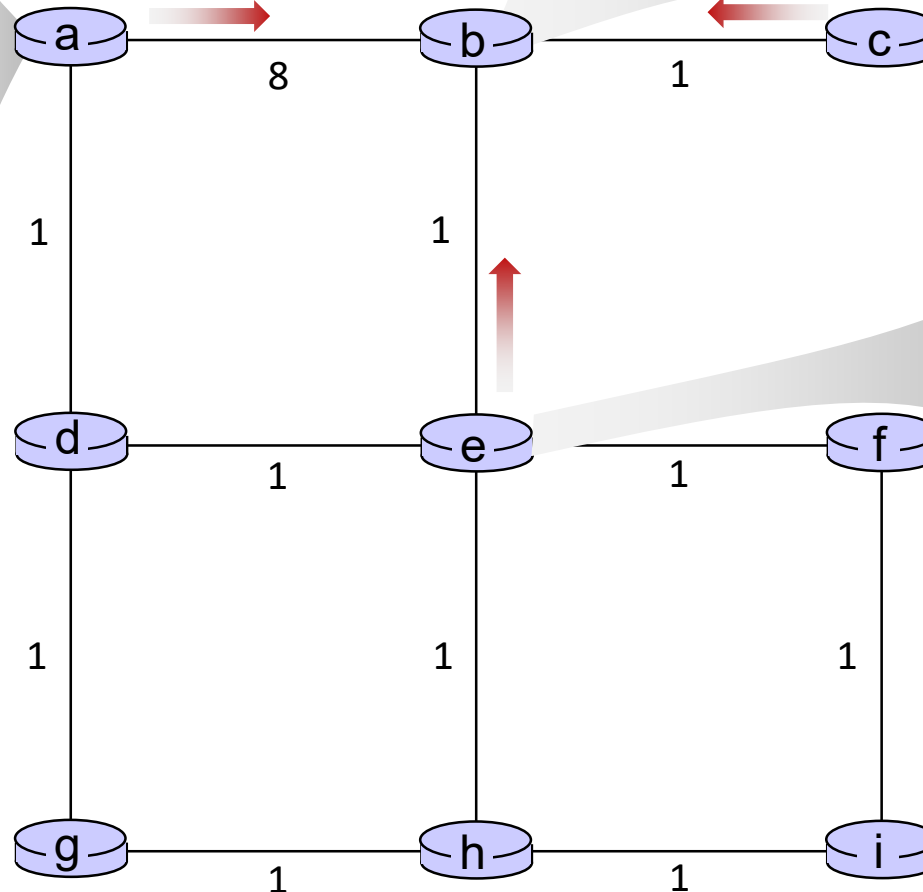
$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$



Distance vector example:

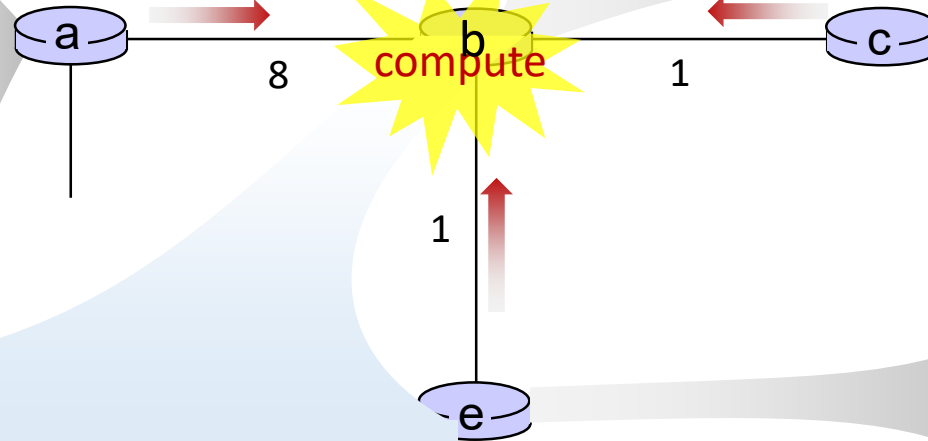


t=1

- b receives DVs from a, c, e, computes:

$$\begin{aligned}
 D_b(a) &= \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8 \\
 D_b(c) &= \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1 \\
 D_b(d) &= \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\} = \min\{9, 2, \infty\} = 2 \\
 D_b(e) &= \min\{c_{b,a} + D_a(e), c_{b,c} + D_c(e), c_{b,e} + D_e(e)\} = \min\{\infty, \infty, 1\} = 1 \\
 D_b(f) &= \min\{c_{b,a} + D_a(f), c_{b,c} + D_c(f), c_{b,e} + D_e(f)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(g) &= \min\{c_{b,a} + D_a(g), c_{b,c} + D_c(g), c_{b,e} + D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty \\
 D_b(h) &= \min\{c_{b,a} + D_a(h), c_{b,c} + D_c(h), c_{b,e} + D_e(h)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(i) &= \min\{c_{b,a} + D_a(i), c_{b,c} + D_c(i), c_{b,e} + D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty
 \end{aligned}$$

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



DV in b:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

DV in b:

$$D_b(a) = 8$$

$$D_b(f) = 2$$

$$D_b(c) = 1$$

$$D_b(g) = \infty$$

$$D_b(d) = 2$$

$$D_b(h) = 2$$

$$D_b(e) = 1$$

$$D_b(i) = \infty$$

Distance vector example:



t=1

- c receives DVs from b

DV in a:
$D_a(a)=0$
$D_a(b)=8$
$D_a(c)=\infty$
$D_a(d)=1$
$D_a(e)=\infty$
$D_a(f)=\infty$
$D_a(g)=\infty$
$D_a(h)=\infty$
$D_a(i)=\infty$

DV in b:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

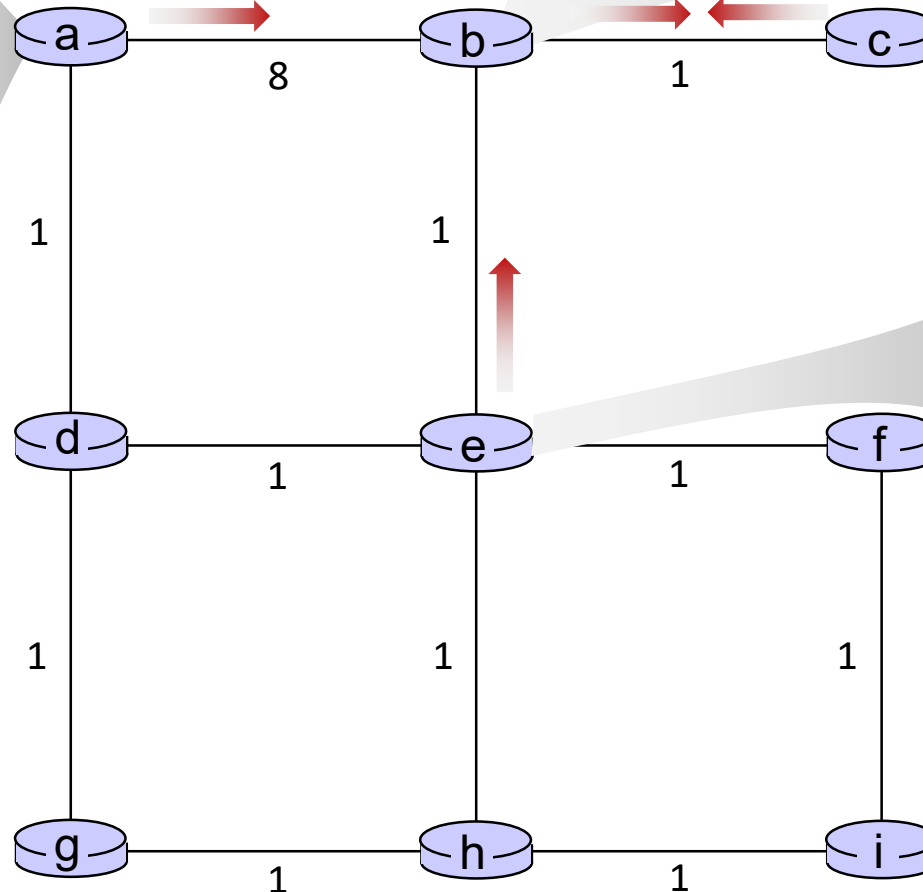
$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

DV in c:
$D_c(a)=\infty$
$D_c(b)=1$
$D_c(c)=0$
$D_c(d)=\infty$
$D_c(e)=\infty$
$D_c(f)=\infty$
$D_c(g)=\infty$
$D_c(h)=\infty$
$D_c(i)=\infty$

DV in e:
$D_e(a)=\infty$
$D_e(b)=1$
$D_e(c)=\infty$
$D_e(d)=1$
$D_e(e)=0$
$D_e(f)=1$
$D_e(g)=\infty$
$D_e(h)=1$
$D_e(i)=\infty$



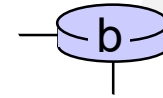
Distance vector example:



t=1

- c receives DVs from b computes:

$$\begin{aligned}D_c(a) &= \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9 \\D_c(b) &= \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1 \\D_c(d) &= \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty \\D_c(e) &= \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2 \\D_c(f) &= \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty \\D_c(g) &= \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty \\D_c(h) &= \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty \\D_c(i) &= \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty\end{aligned}$$



1

compute

DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in c:

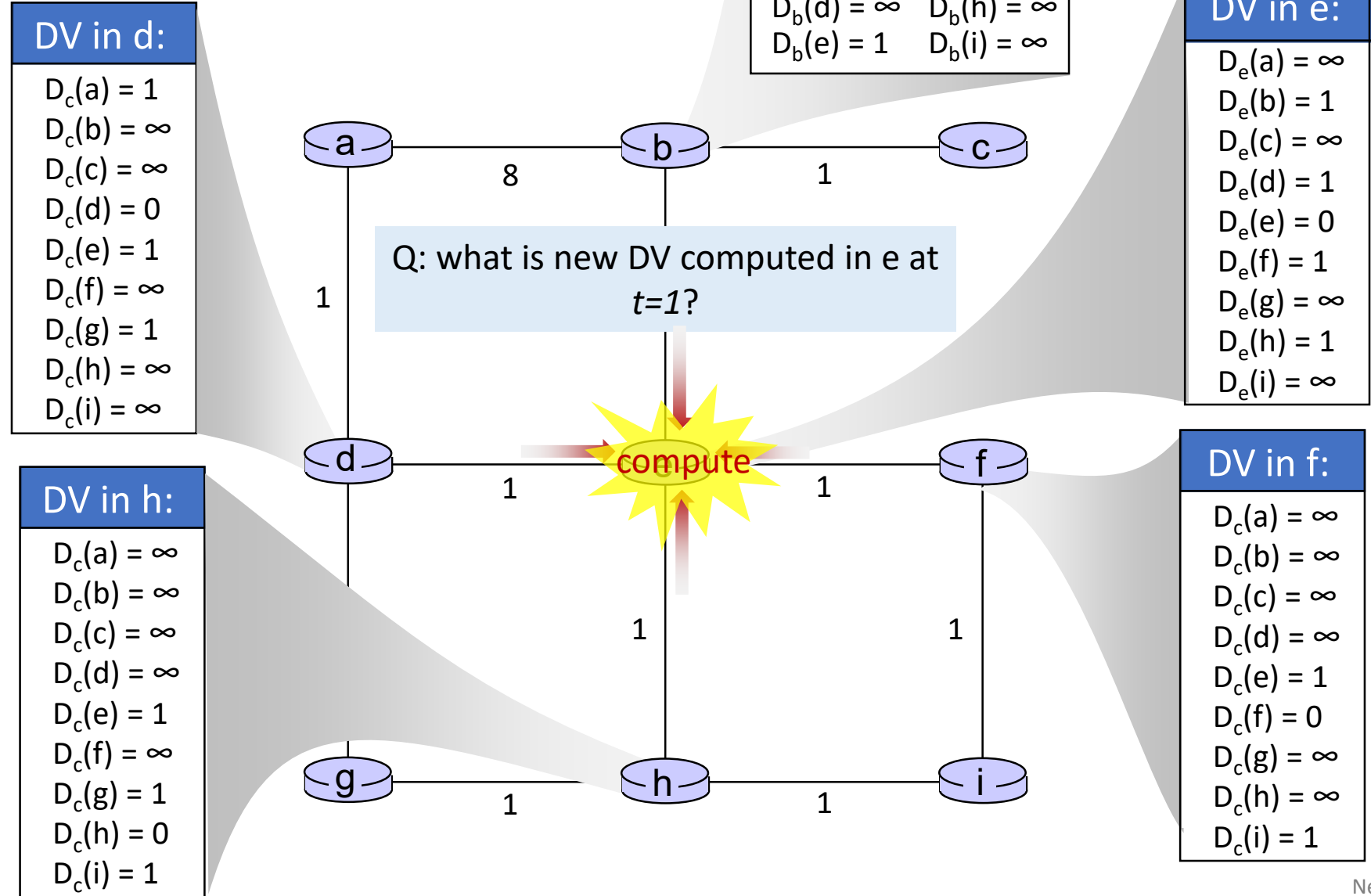
$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

Distance vector example:








t=1

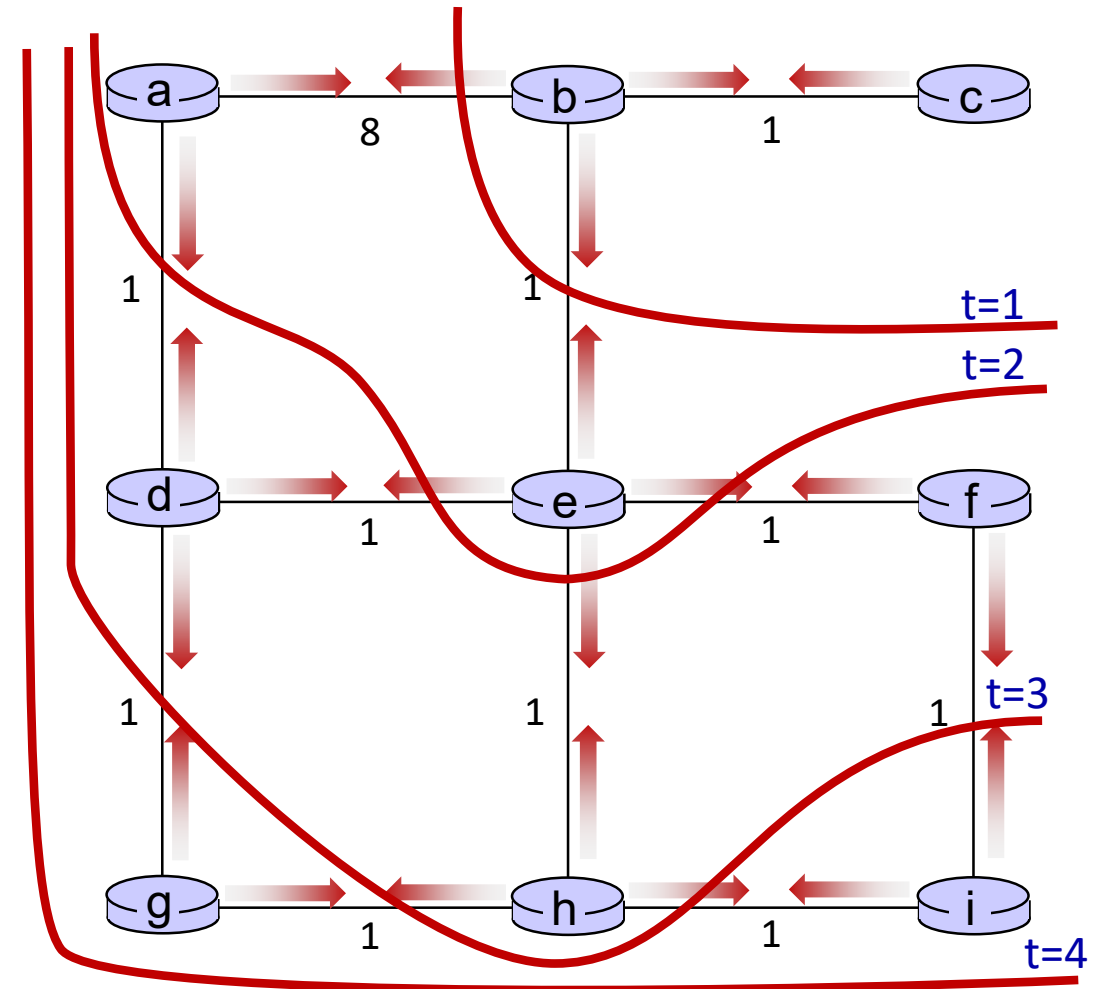
- e receives DVs from b, d, f, h



Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

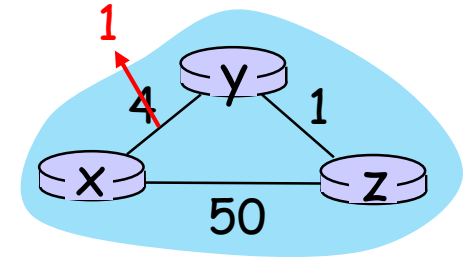
-  $t=0$ c's state at $t=0$ is at c only
-  $t=1$ c's state at $t=0$ has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  $t=2$ c's state at $t=0$ may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  $t=3$ c's state at $t=0$ may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well
-  $t=4$ c's state at $t=0$ may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well



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 table, computes new least cost to x, sends its neighbors its DV.

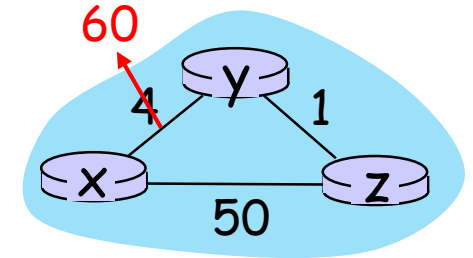
t_2 : y receives z's update, updates its distance table. y's least costs do *not* change, so y does *not* send a message to z, which *means* converges fast!

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- **“bad news travels slow” – count-to-infinity problem:**
 - y sees the 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.
 - ...

➔ *Distributed algorithms are tricky!*



Comparison of LS and DV algorithms

both compute **routing tables**:

- more info than just forwarding table
- gives cost to reach destination

message complexity

LS: n routers, m links $O(nm)$ messages

DV: exchange between neighbors;
convergence time varies

speed of convergence

LS: $O(n^2)$ -time algorithm

- may have oscillations (link cost changes)

DV: convergence time varies, $O(mn)$

- may have routing loops,
- count-to-infinity problem (link cost changes)

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

LS: localize impact!

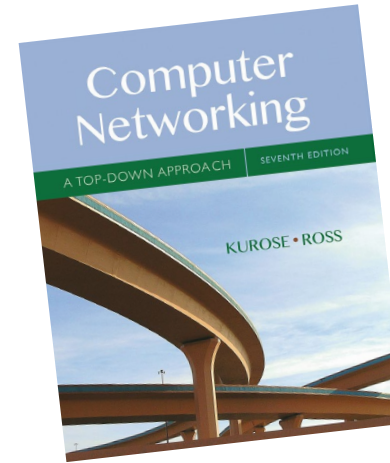
- router can advertise incorrect *link* cost
- But each router computes only its *own* table (computation is more robust)

DV: trust your neighbors!

- DV router can advertise incorrect *path* cost (“I have a *really* low-cost path to everywhere”) → “**blackhole**” attack!
- each router’s table used by others:
→ error propagates through network

Network layer: “control plane” roadmap

- introduction
- routing protocols
- **intra-ISP routing: RIP & OSPF**
- routing among ISPs: BGP
- SDN control plane



Chapter 5

Making routing scalable

our routing study so far - idealized

- all routers identical
- network “flat” (flat topology)

... not true in practice

scale: billions of destinations:

- can't store all destinations in routing tables! (**memory** problem)
- routing table exchange would swamp links! (**bandwidth** problem)

administrative autonomy:

- Mimic Post systems: town/city/state/...
- Internet: a network of networks
- each network admin may want to control routing in its own network

Internet approach to scalable routing

organize routers into regions known as “autonomous systems” (**AS**) a.k.a. “domains”

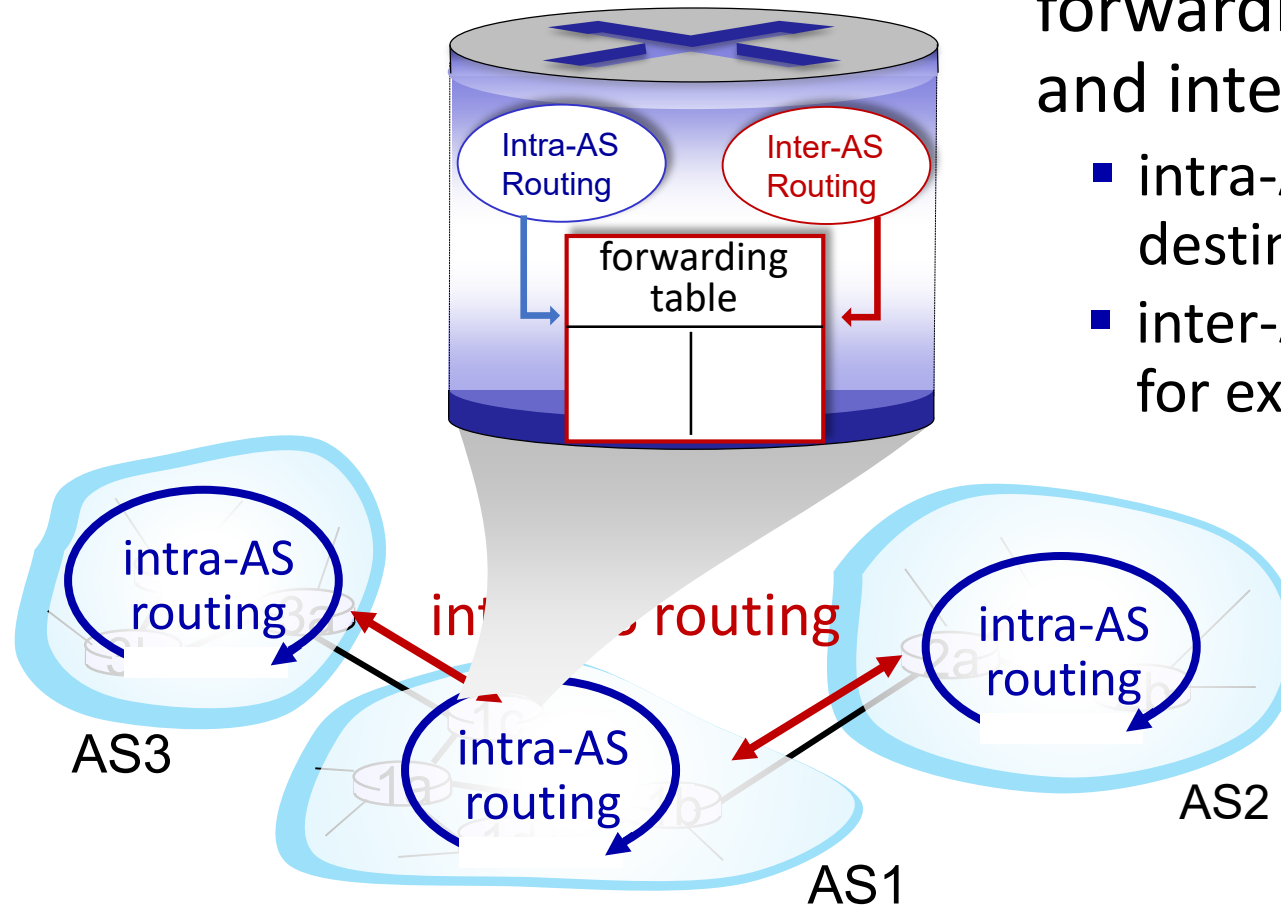
intra-AS (aka “intra-domain”):
routing *within same AS* (“network”)

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

inter-AS (aka “inter-domain”):
routing *among* ASs

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

Interconnected ASs



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

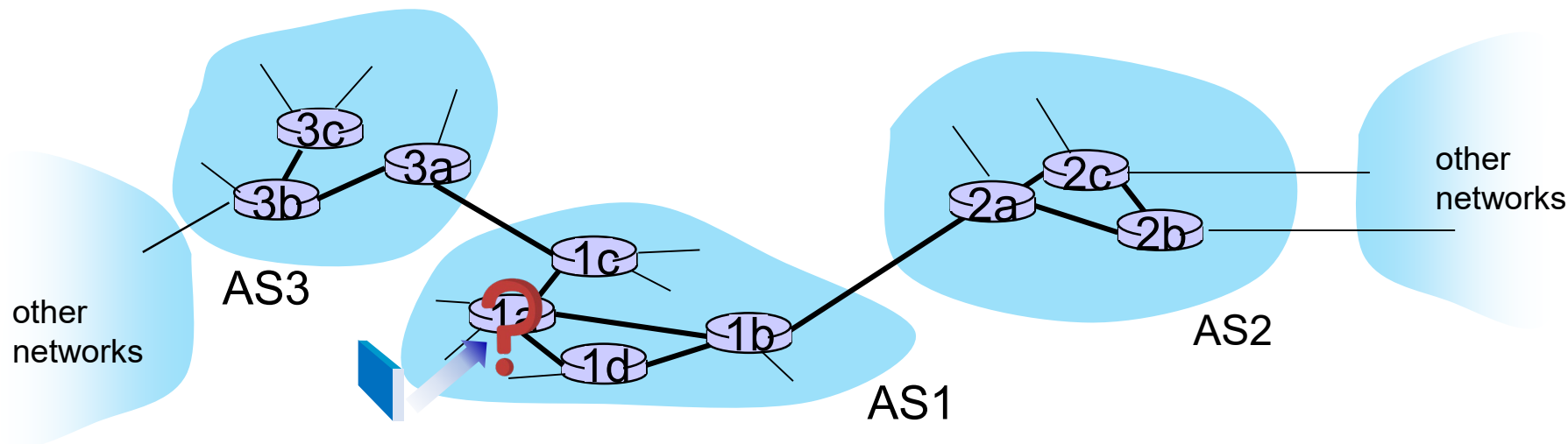
- intra-AS routing determines entries for destinations within an AS
- inter-AS & intra-AS determine entries for external destinations

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



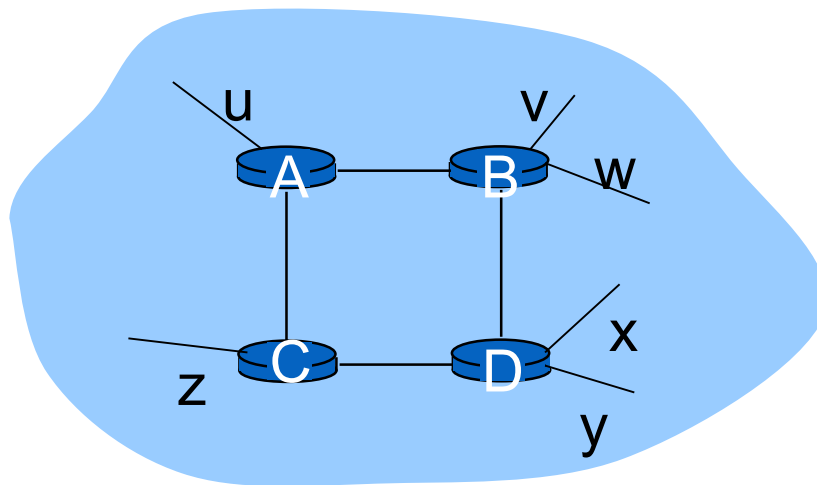
Intra-AS routing: routing within an AS

most common intra-AS routing protocols:

- **RIP: Routing Information Protocol** [RFC 1723]
 - **classical DV**: DVs exchanged every 30 secs over Internet
 - no longer 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 the same as OSPF
 - IS: Intermediate System

RIP (Routing Information Protocol)

- included in BSD-UNIX distribution in 1982
- distance vector algorithm
 - **distance metric**: # hops (max = 15 hops), each link has cost 1
 - DVs exchanged with neighbors every 30 sec in response message (aka **advertisement**)
 - each **advertisement**: list of up to 25 destination **subnets** (*in IP addressing sense*) and their distances from sender

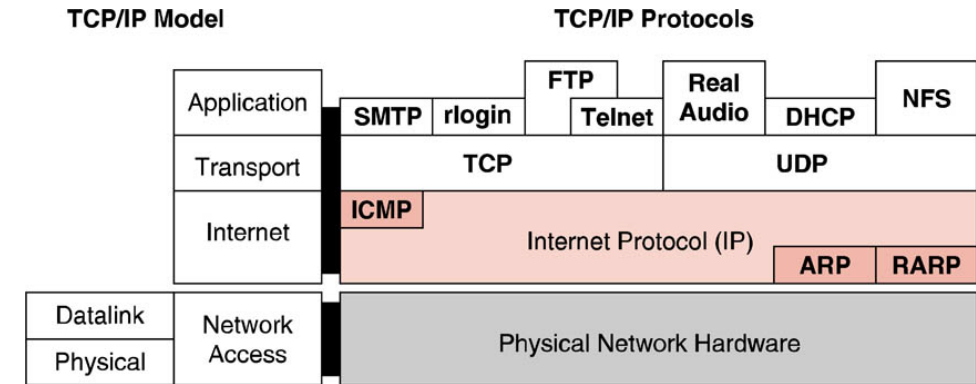


from router A to destination **subnets**:

<u>subnet</u>	<u>hops</u>
u	1
v	2
w	2
x	3
y	3
z	2

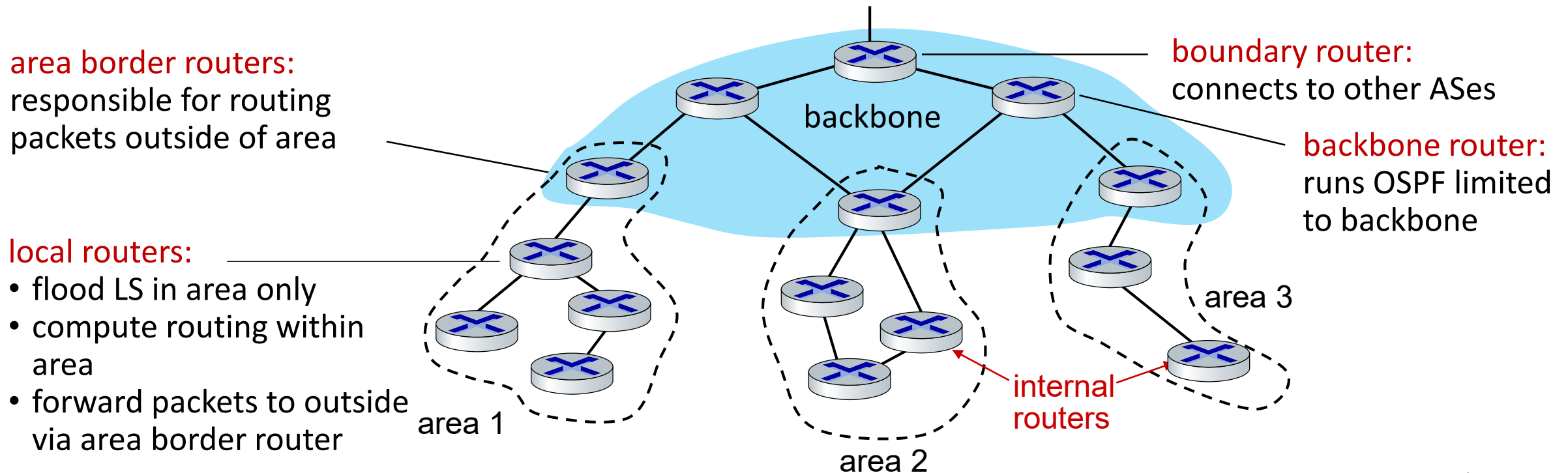
OSPF (Open Shortest Path First) routing

- “open”: publicly available
- classical link-state
 - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP, e.g., ICMP) to all other routers in entire AS
 - multiple link costs metrics possible: bandwidth, delay, energy
 - each router has **full topology (global info)**, uses Dijkstra’s algorithm to compute forwarding table
- **security**: all OSPF messages authenticated (to prevent malicious intrusion)



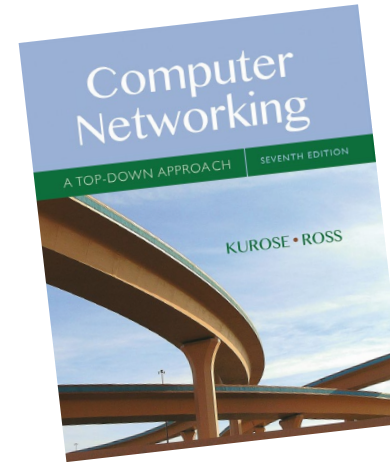
Hierarchical OSPF

- **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



Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: RIP & OSPF
- **routing among ISPs: BGP**

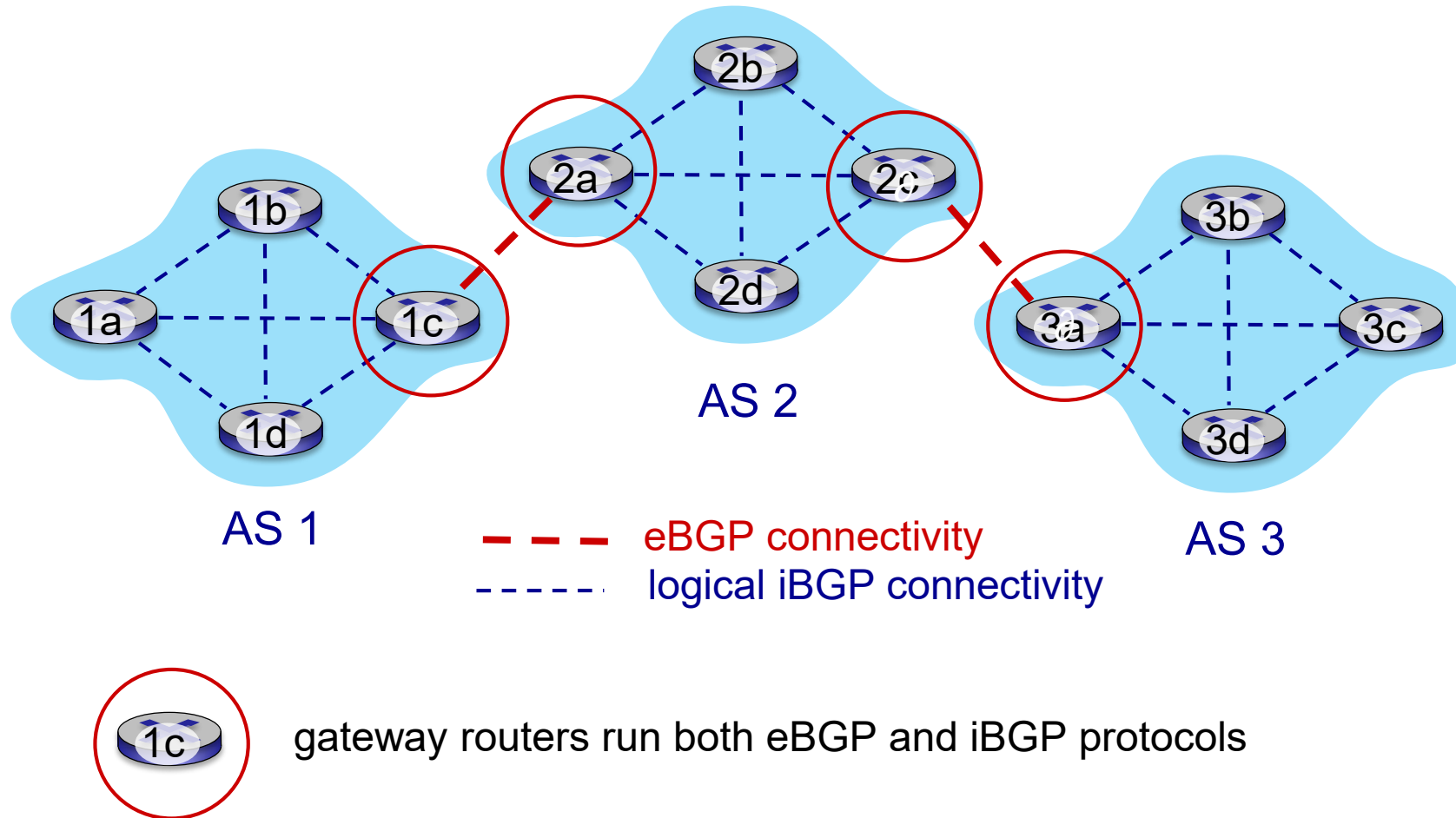


Chapter 5

Internet inter-AS routing: BGP

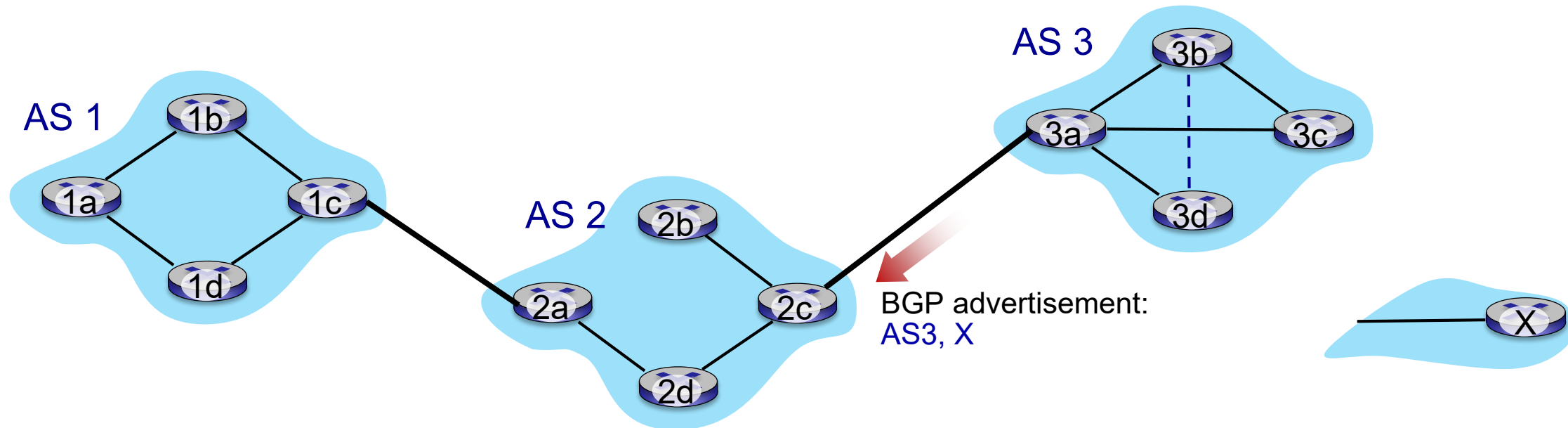
- **BGP (Border Gateway Protocol):** *the* de facto inter-domain routing protocol
 - “glue that holds the Internet together”
- allows each subnet to advertise its existence, and the destinations it can reach to rest of Internet: *“I am here, here is who I can reach, and how”*
- BGP provides each AS a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

eBGP, iBGP connections



BGP basics

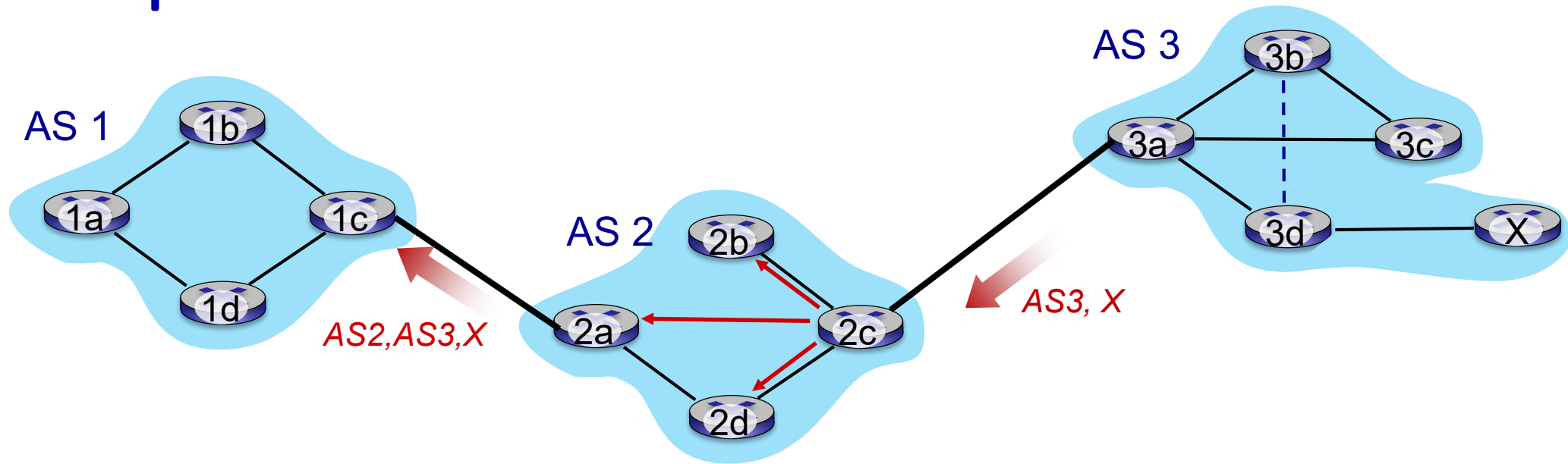
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
 - advertising *paths* to different destination *network prefixes* (BGP is a “path vector” protocol)
- when AS3 gateway 3a advertises *path AS3,X* to AS2 gateway 2c:
 - AS3 *promises* to AS2 it will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



Path attributes and BGP routes

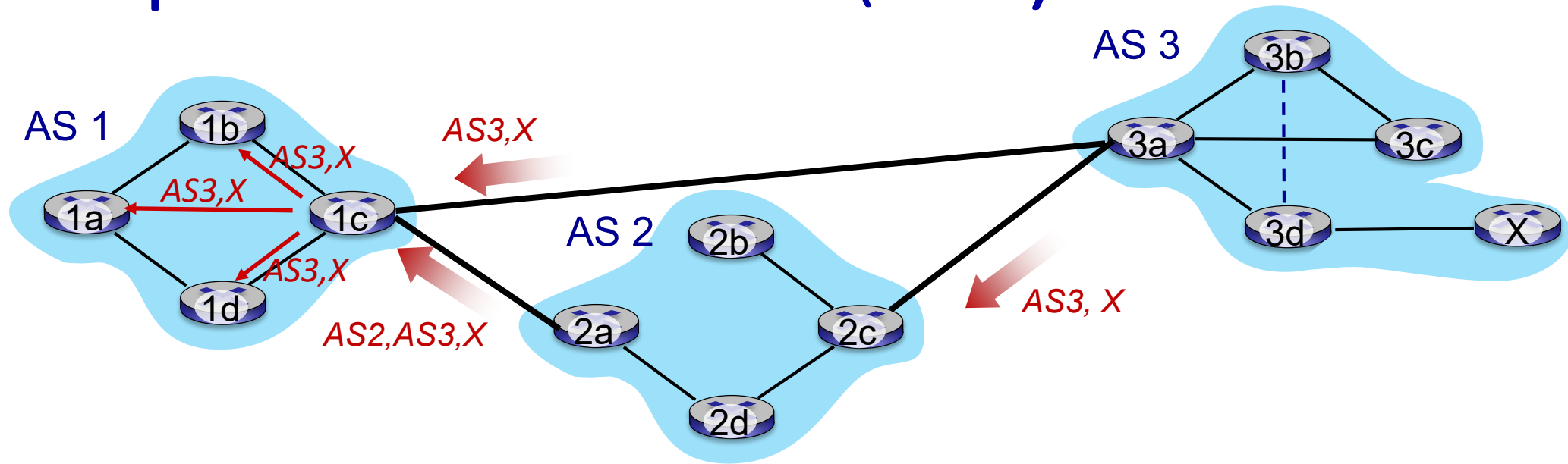
- BGP advertised route: **prefix + attributes**
 - prefix: destination being advertised
 - two important attributes:
 - **AS-PATH**: list of ASs through which prefix advertisement has passed
 - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
- **policy-based routing**:
 - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
 - AS policy also determines whether to *advertise* path to other neighboring ASs

BGP path advertisement



- AS2 router 2c receives path advertisement **AS3,X** (via **eBGP**) from AS3 router 3a
- based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via **iBGP**) to all AS2 routers
- based on AS2 policy, AS2 router 2a advertises (via **eBGP**) path **AS2, AS3, X** to AS1 router 1c

BGP path advertisement (more)



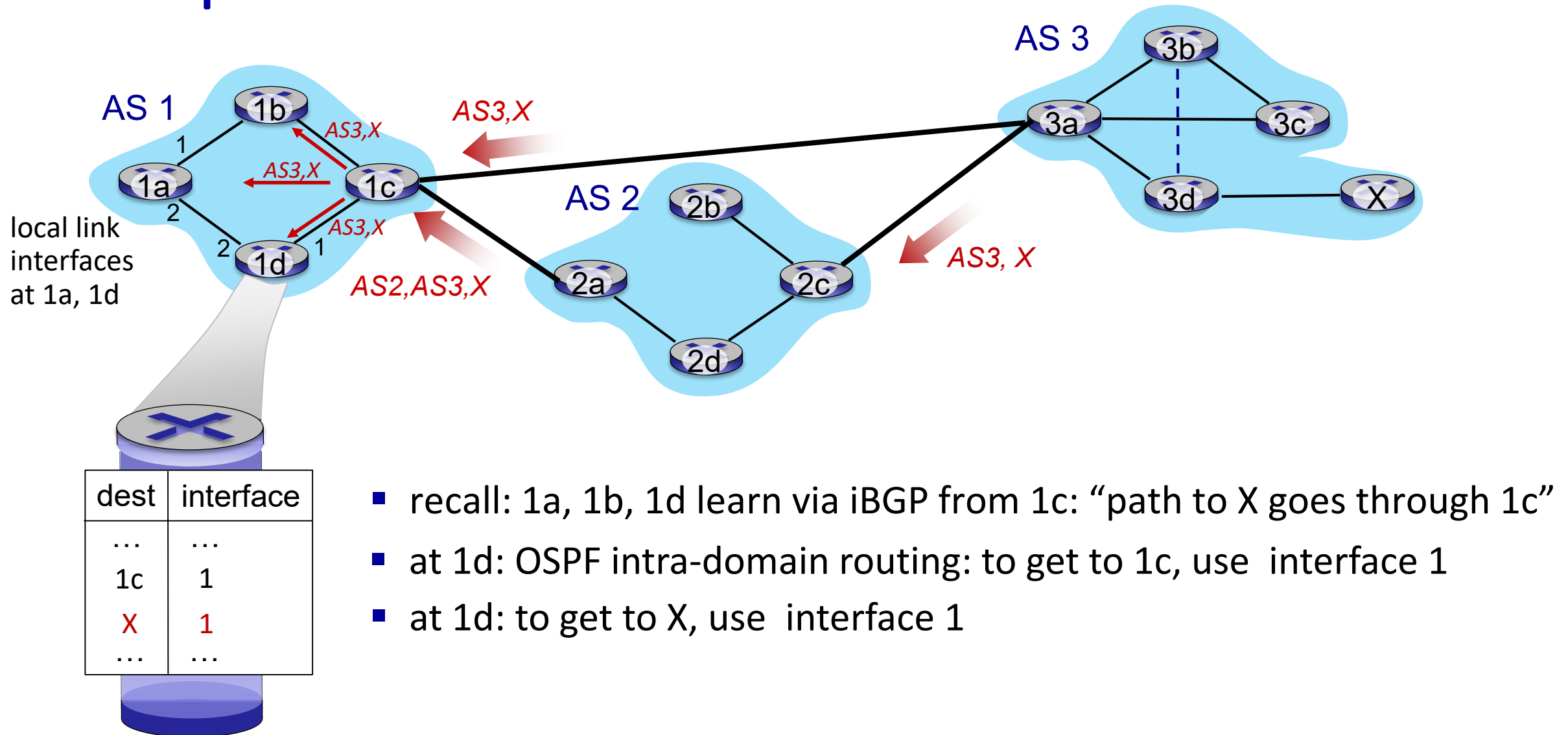
gateway router may learn about **multiple** paths to destination:

- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a
- AS1 gateway router 1c learns path **AS3,X** from 3a
- based on **policy**, AS1 gateway router 1c chooses path **AS3,X** and advertises path within AS1 via iBGP

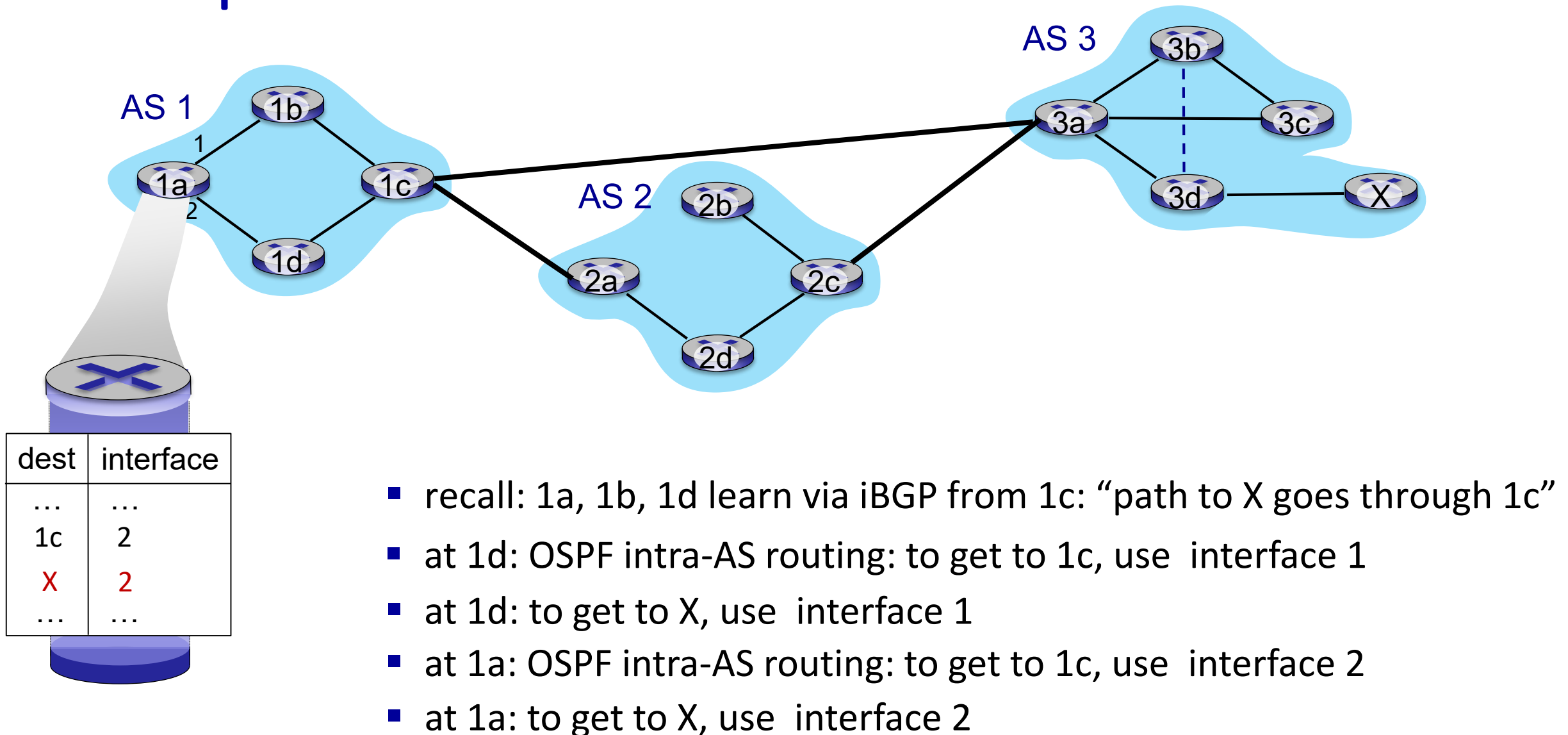
BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages:
 - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
 - **UPDATE**: advertises new path (or withdraws old)
 - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
 - **NOTIFICATION**: reports errors in previous msg; also used to close connection

BGP path advertisement



BGP path advertisement



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

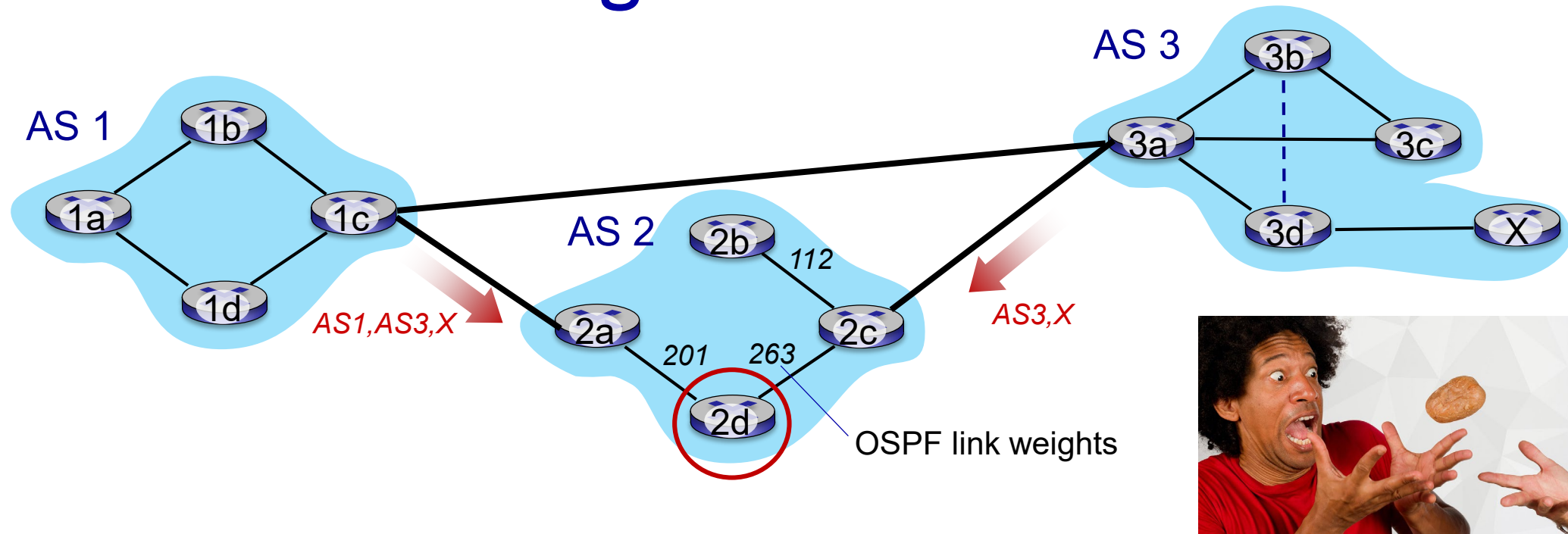
scale:

- hierarchical routing saves the forwarding table size (memory), reduces update traffic (bandwidth)

performance:

- intra-AS: can focus on performance (optimality, stability, ...)
- inter-AS: policy dominates over performance

Hot Potato Routing

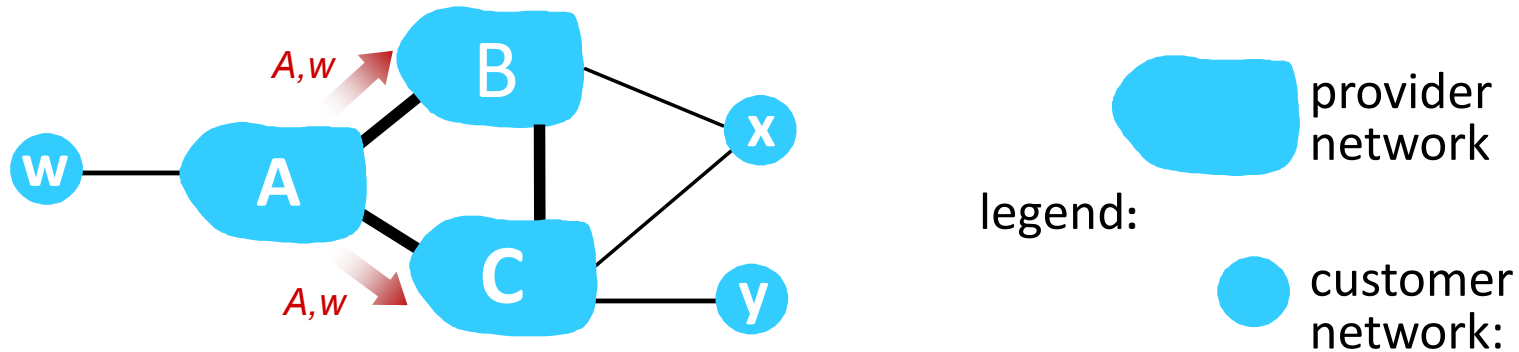


- 2d learns (via iBGP) it can route to X via 2a or 2c
- **hot potato routing**: choose local gateway that has least *intra-domain* cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

BGP route selection algorithm

- More complicated than just hot potato routing
- Router selects route based on:
 1. local preference value attribute: policy decision
 2. shortest AS-PATH
 3. closest NEXT-HOP router: use **hot potato routing**
 4. Any casting
 5. additional criteria...

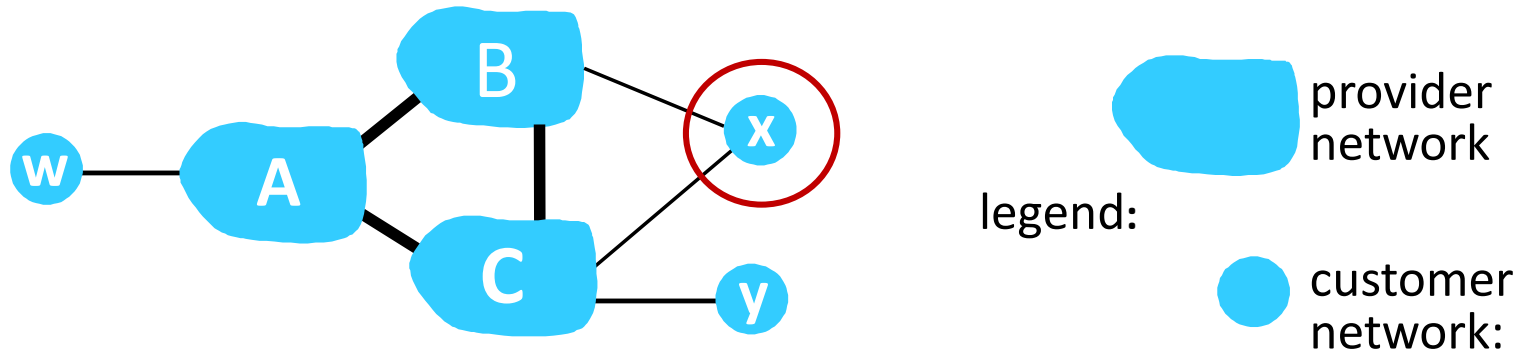
BGP: achieving policy via advertisements



ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic for other ISPs – a typical “real world” policy)

- A advertises path Aw to B and C
- B *chooses not to advertise* BA_w to C!
 - B gets no “revenue” for routing CBA_w, since none of C, A, w are B’s customers
 - C does *not* learn about CBA_w path
- C will use route CA_w (not using B) to get to w

BGP: achieving policy via advertisements (more)

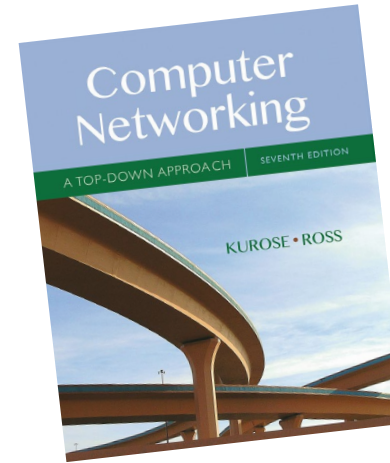


ISP only wants to route traffic to/from its own customer networks (does not want to carry transit traffic for other ISPs – a typical “real world” policy)

- A,B,C are **provider networks**
- x,w,y are **customers** (of provider networks)
- x is **dual-homed**: attached to two ISPs
- **policy to enforce**: x does not want to route from B to C via x
 - .. so x will not advertise to B a route to C

Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: RIP & OSPF
- routing among ISPs: BGP
- **SDN control plane**



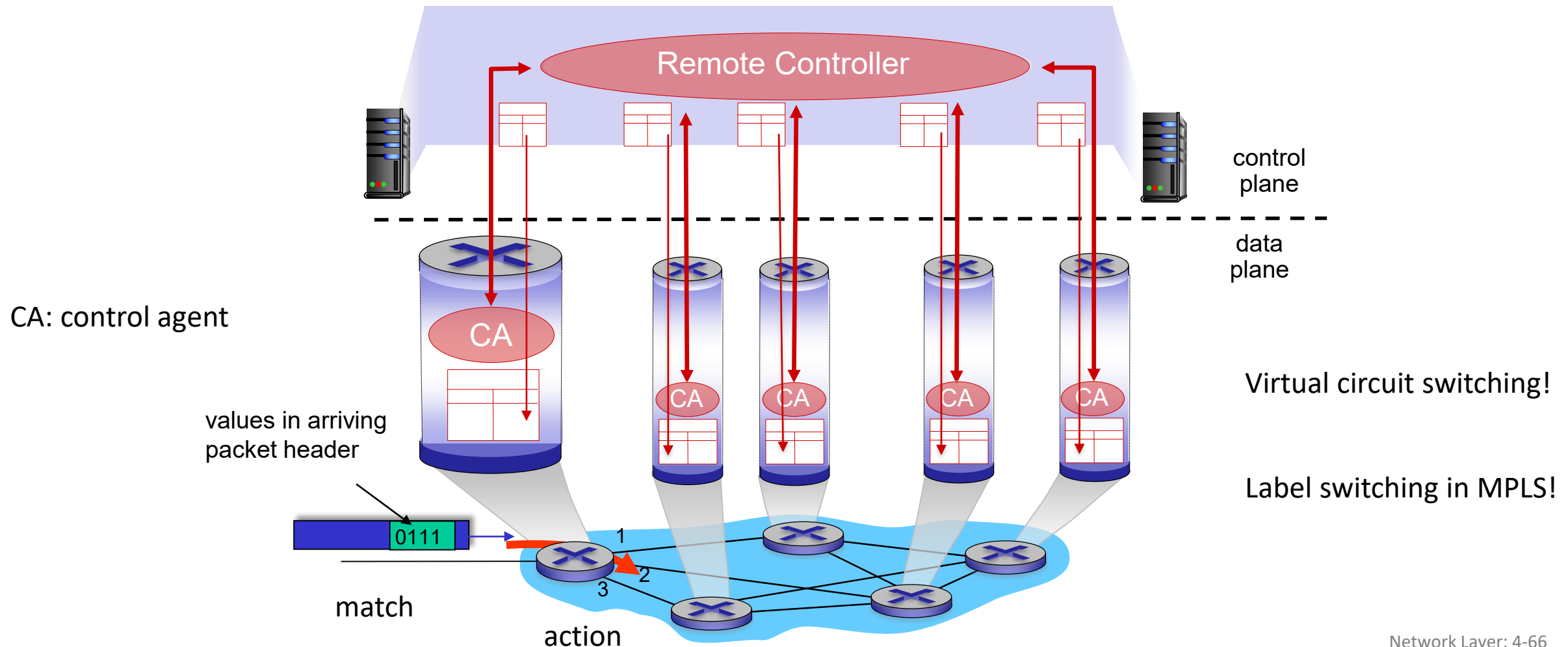
Chapter 5

Software-defined networking (SDN)

- Internet network layer: historically implemented via distributed, **per-router** control approach:
 - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
 - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes,...
- ~2005: renewed interest in rethinking network control plane
 - Open Flow reenergizes SDN, originated from **ATM** switching
 - Clean slate drive from US NSF
 - Learn from telephone industries

Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

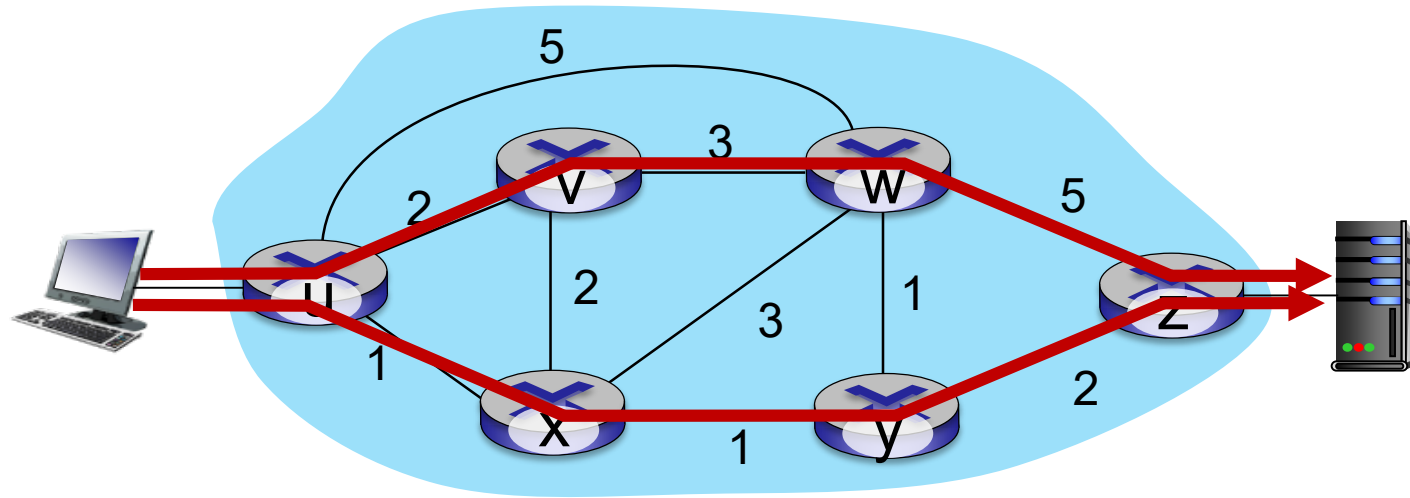


Software-defined networking (SDN)

Why a *logically centralized* control plane?

- **Continuously monitoring and fast action-taking:** e.g., slow electronic controls fast optical
- **easier network management:** avoid router misconfigurations, greater flexibility of traffic flows
- table-driven forwarding (recall OpenFlow API) allows “programming” routers (“**controllable**”)
 - **centralized** “programming” easier: compute tables centrally and distribute: does not need everybody to compute!
 - **distributed** “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- **open** (non-proprietary) implementation of the control plane
 - foster innovation

Traffic engineering: difficult with traditional routing

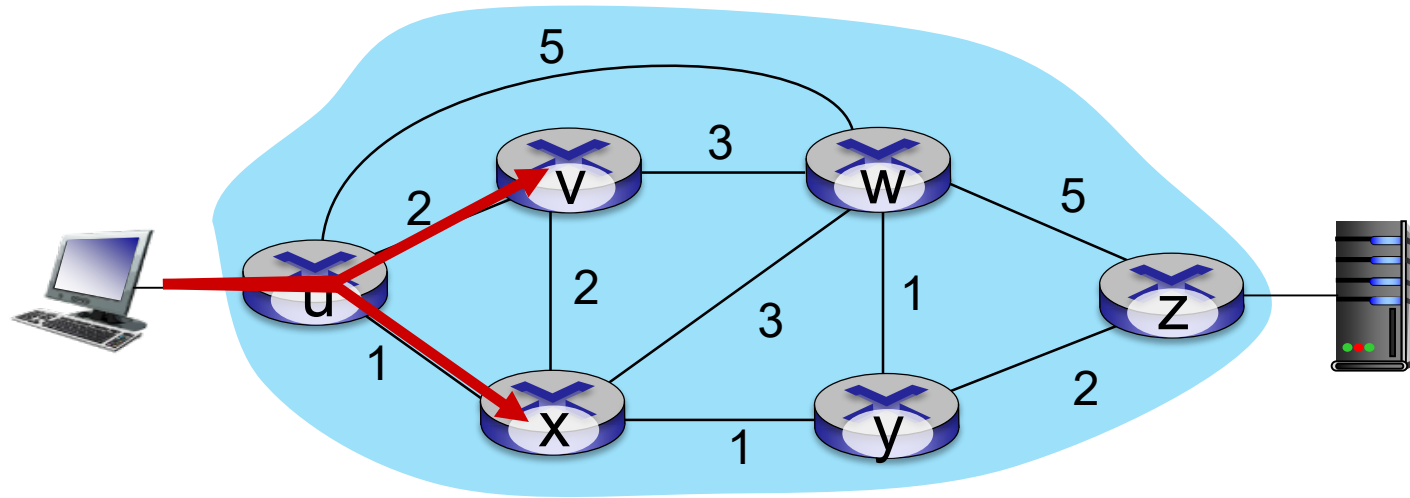


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

link weights are only control “knobs”: not much control, but regulate!

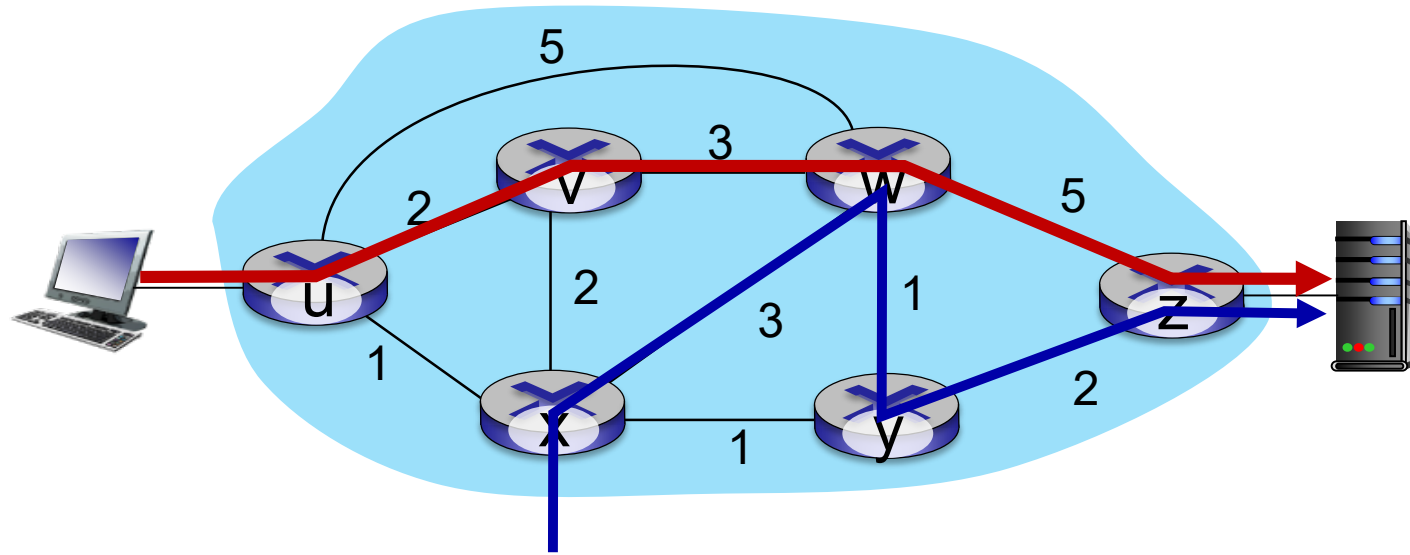
Traffic engineering: difficult with traditional routing



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm, i.e., multipath routing with routing strategies like random routing)

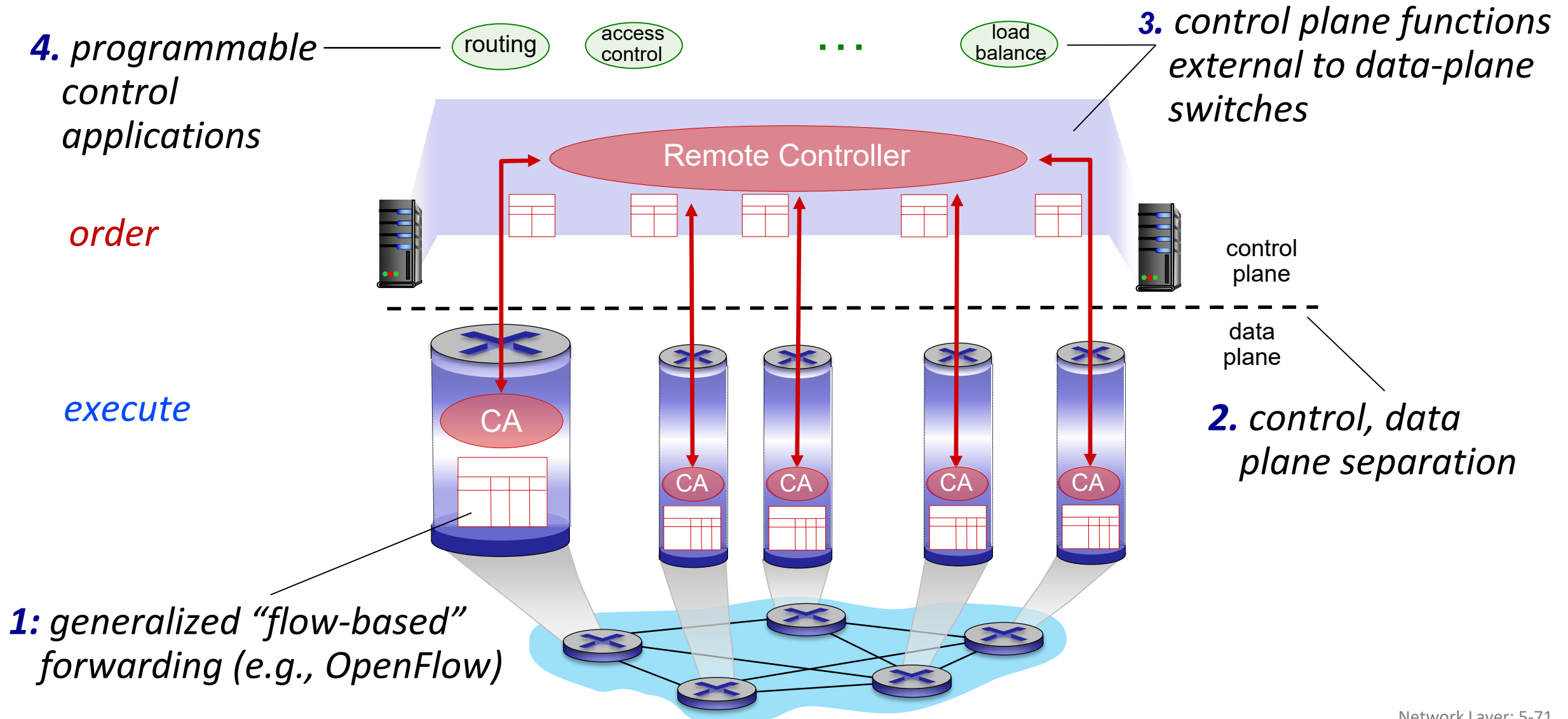
Traffic engineering: difficult with traditional routing



Q: What if w wants to route blue and red traffic differently from w to z?

A: Can't do it (with destination-based forwarding, and LS, DV routing)

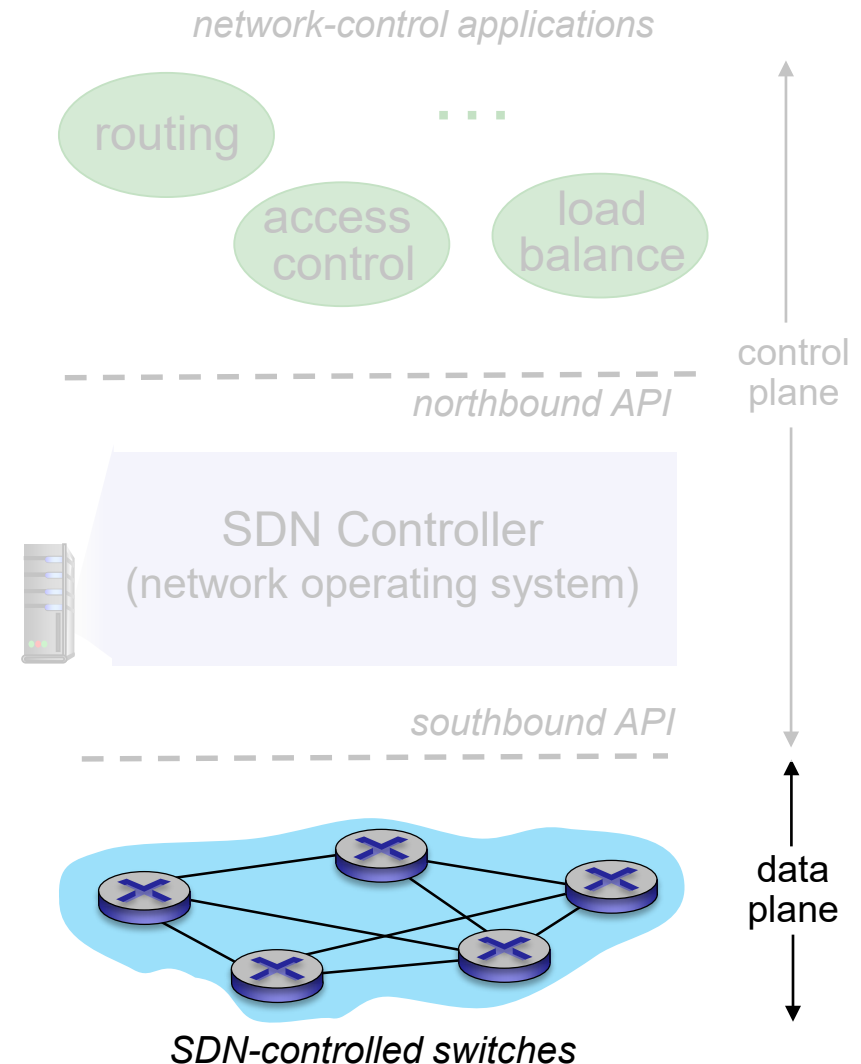
Software-defined networking (SDN)



Software-defined networking (SDN)

Data-plane switches:

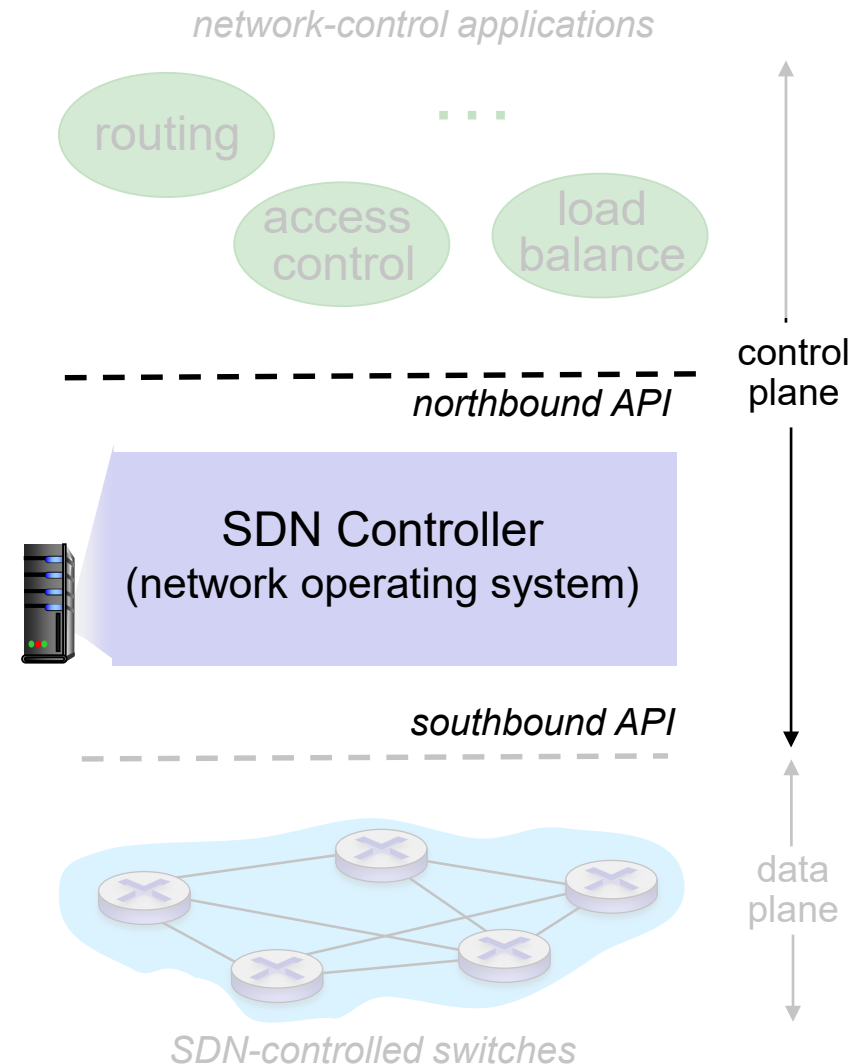
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
 - Analogy: train track switches!
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)



Software-defined networking (SDN)

SDN controller (network OS):

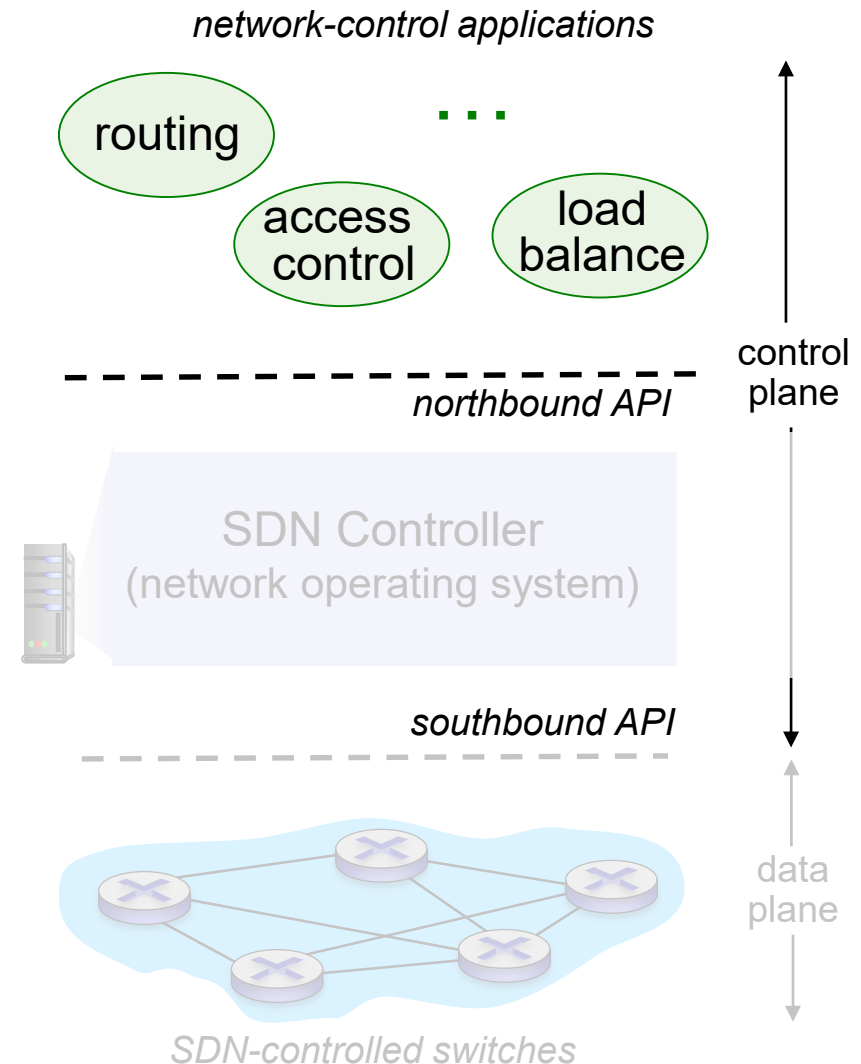
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



Software defined networking (SDN)

network-control apps:

- “brains” of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled*: can be provided by 3rd party: distinct from routing vendor, or SDN controller
- Hardware switching fabrics do not have to change, but programmable controller (software) takes actions

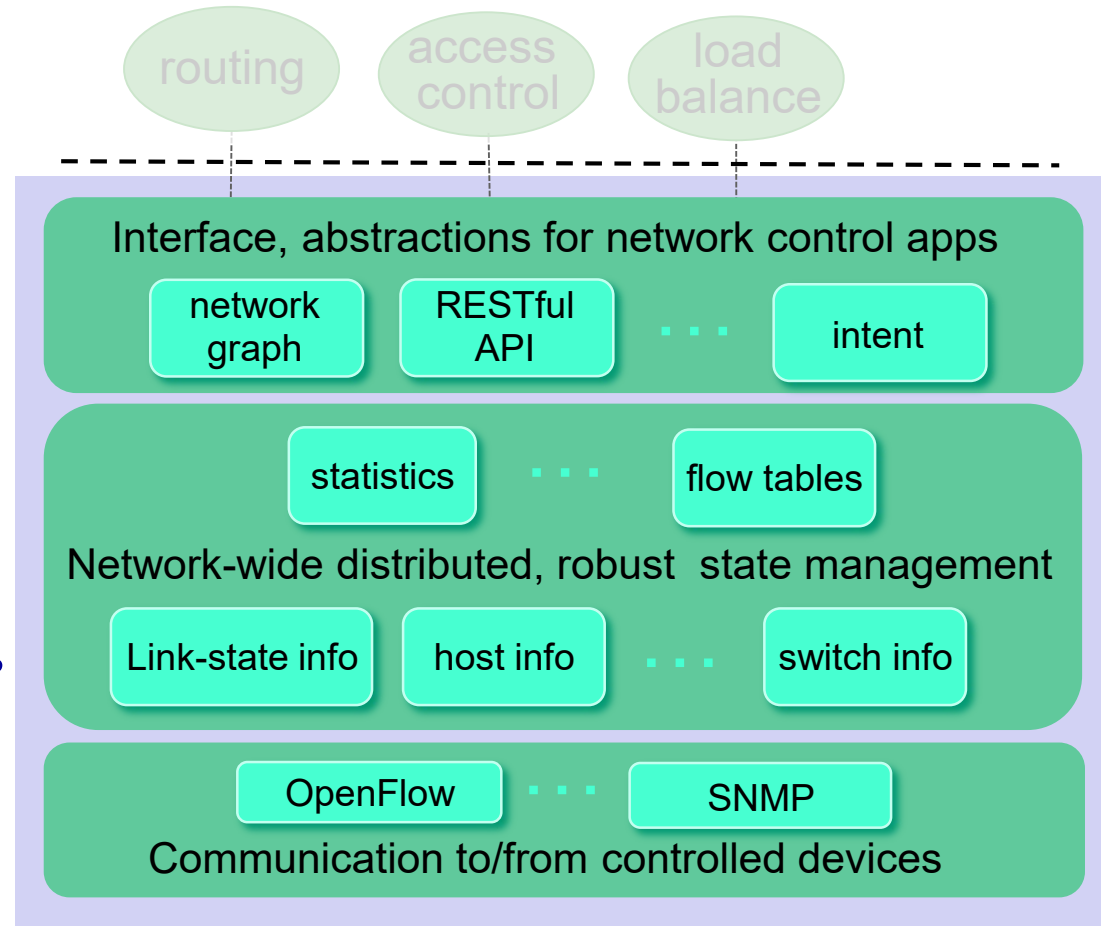


Components of SDN controller

interface layer to network control apps: abstractions API

network-wide state management : state of networks links, switches, services: a *distributed database*

communication: communicate between SDN controller and controlled switches



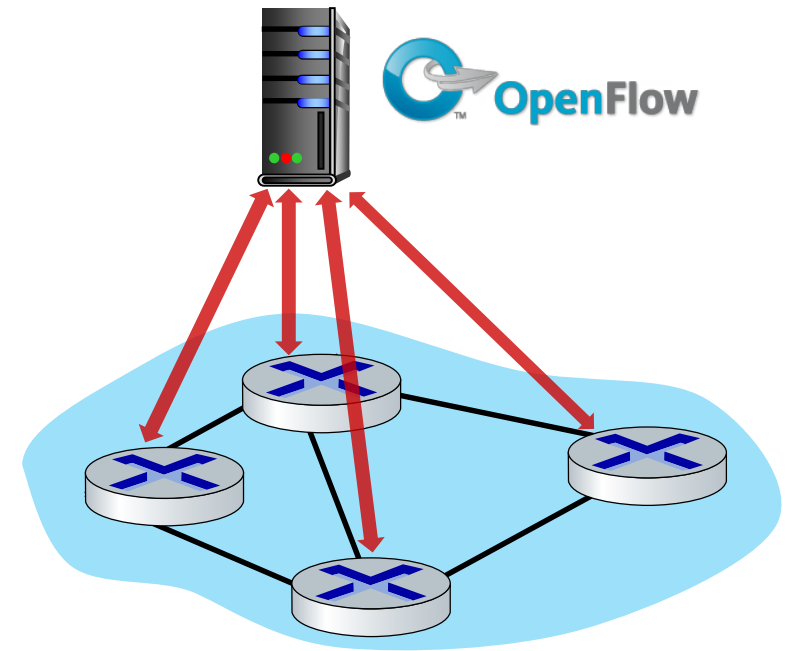
SDN
Controller:
info to be
collected
for control



OpenFlow protocol

- operates between controller and switch
- TCP used to exchange messages
 - optional encryption
- three classes of OpenFlow messages:
 - controller-to-switch
 - asynchronous (switch to controller)
 - symmetric (misc.)
- distinct from OpenFlow API
 - API used to specify generalized forwarding actions

OpenFlow Controller

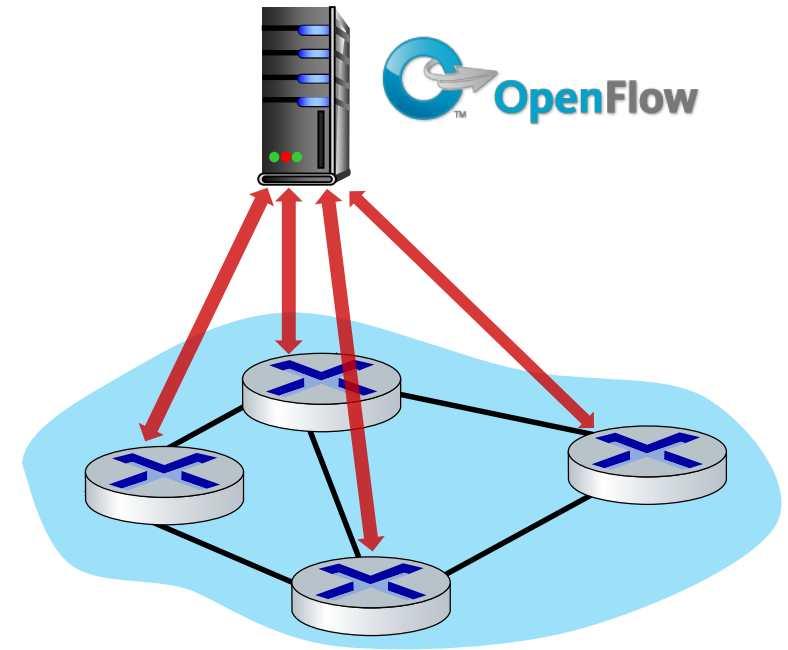


OpenFlow: controller-to-switch messages

Key controller-to-switch messages

- *features*: controller queries switch features, switch replies
- *configure*: controller queries/sets switch configuration parameters
- *modify-state*: add, delete, modify flow entries in the OpenFlow tables
- *packet-out*: controller can send this packet out of specific switch port

OpenFlow Controller

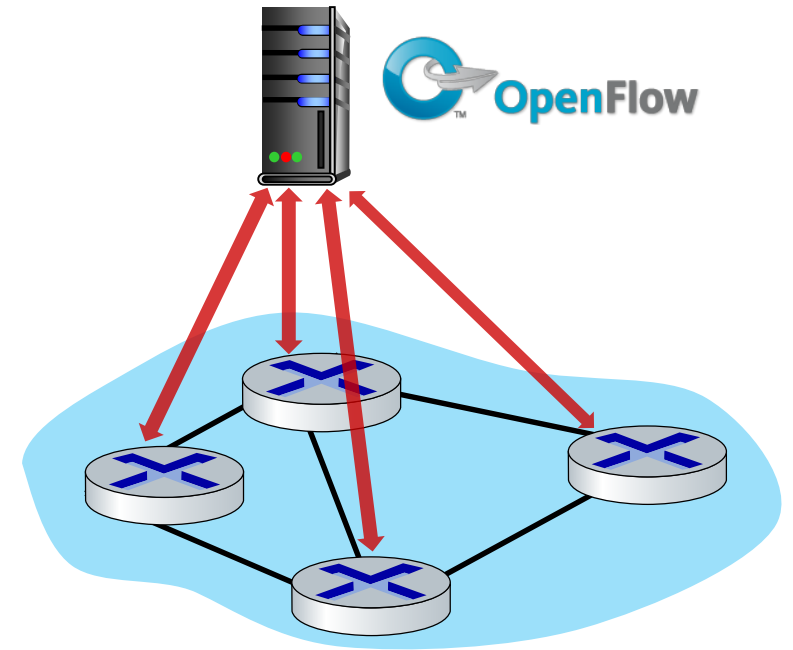


OpenFlow: switch-to-controller messages

Key switch-to-controller messages

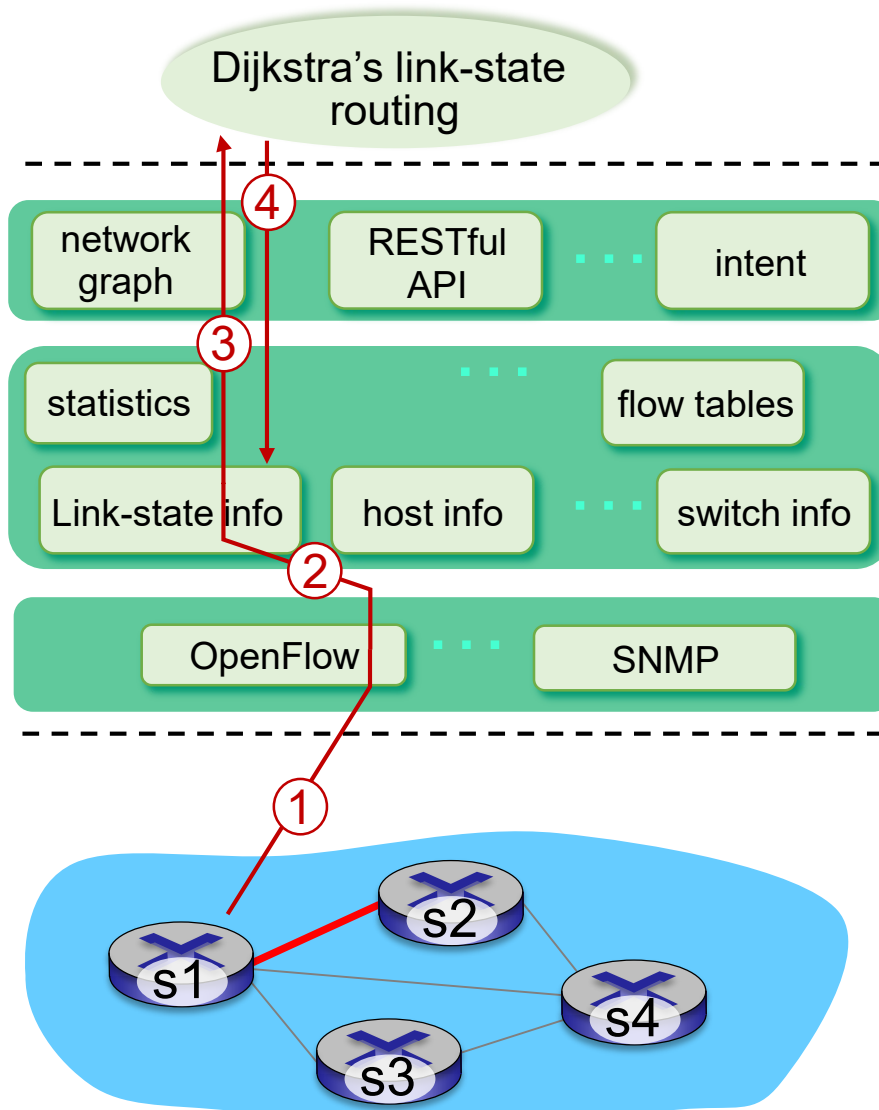
- *packet-in*: transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed*: flow table entry deleted at switch
- *port status*: inform controller of a change on a port.

OpenFlow Controller



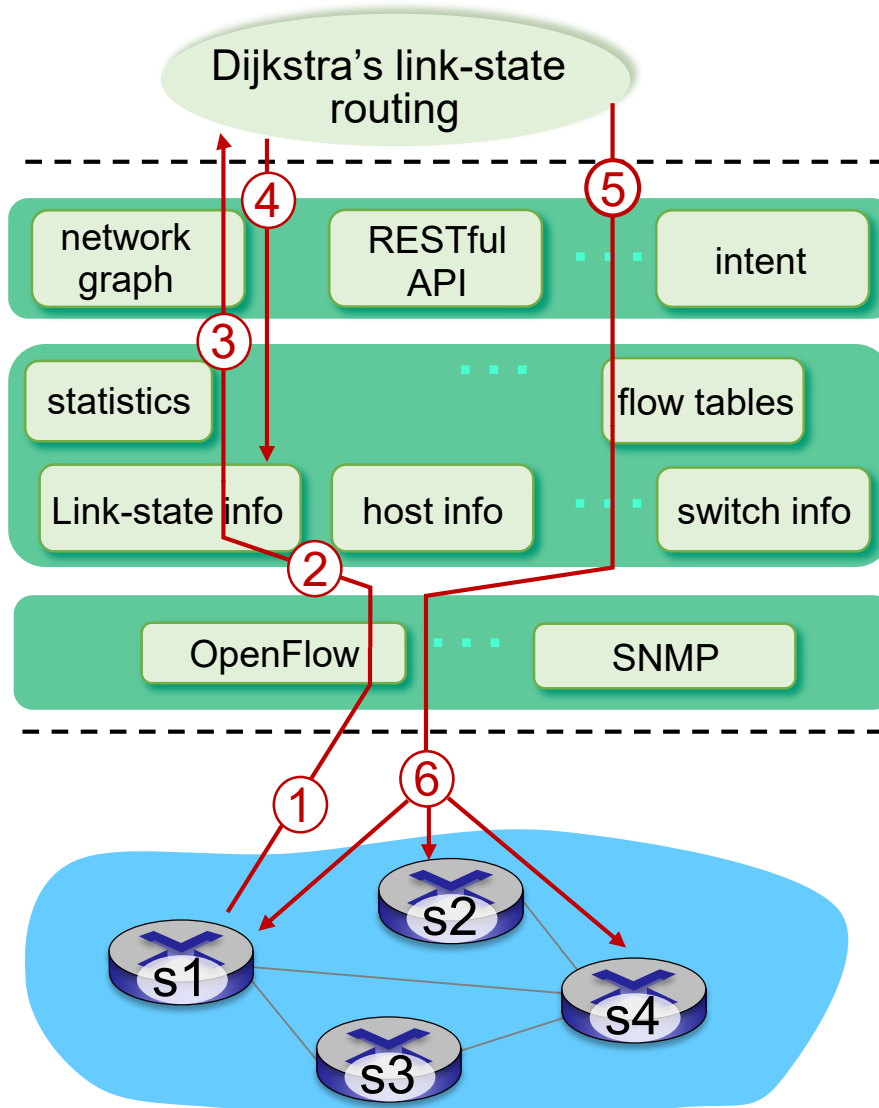
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

SDN: control/data plane interaction example



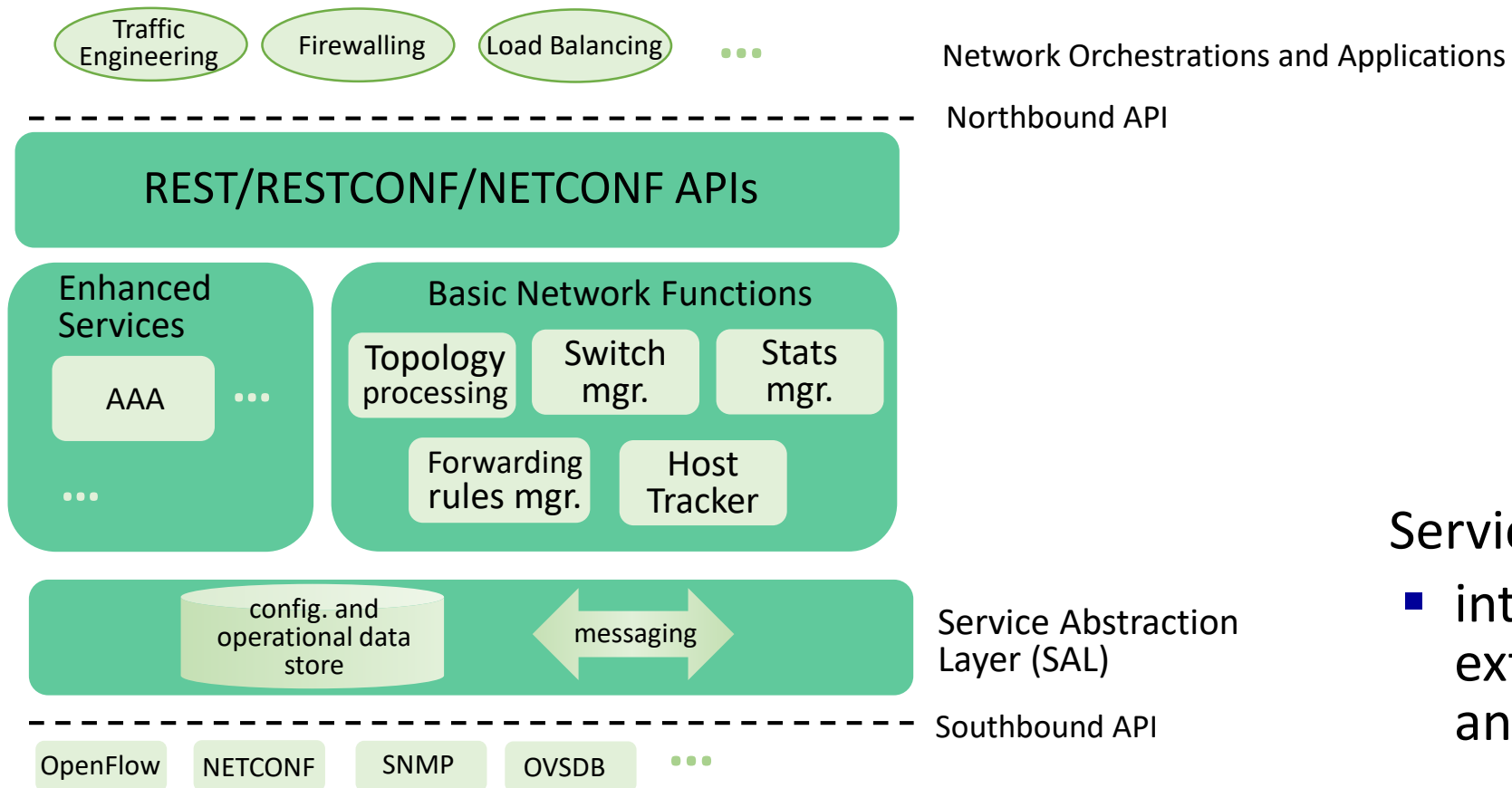
- ① S1, experiencing link failure uses OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

SDN: control/data plane interaction example



- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ controller uses OpenFlow to install new flow tables in switches that need updating

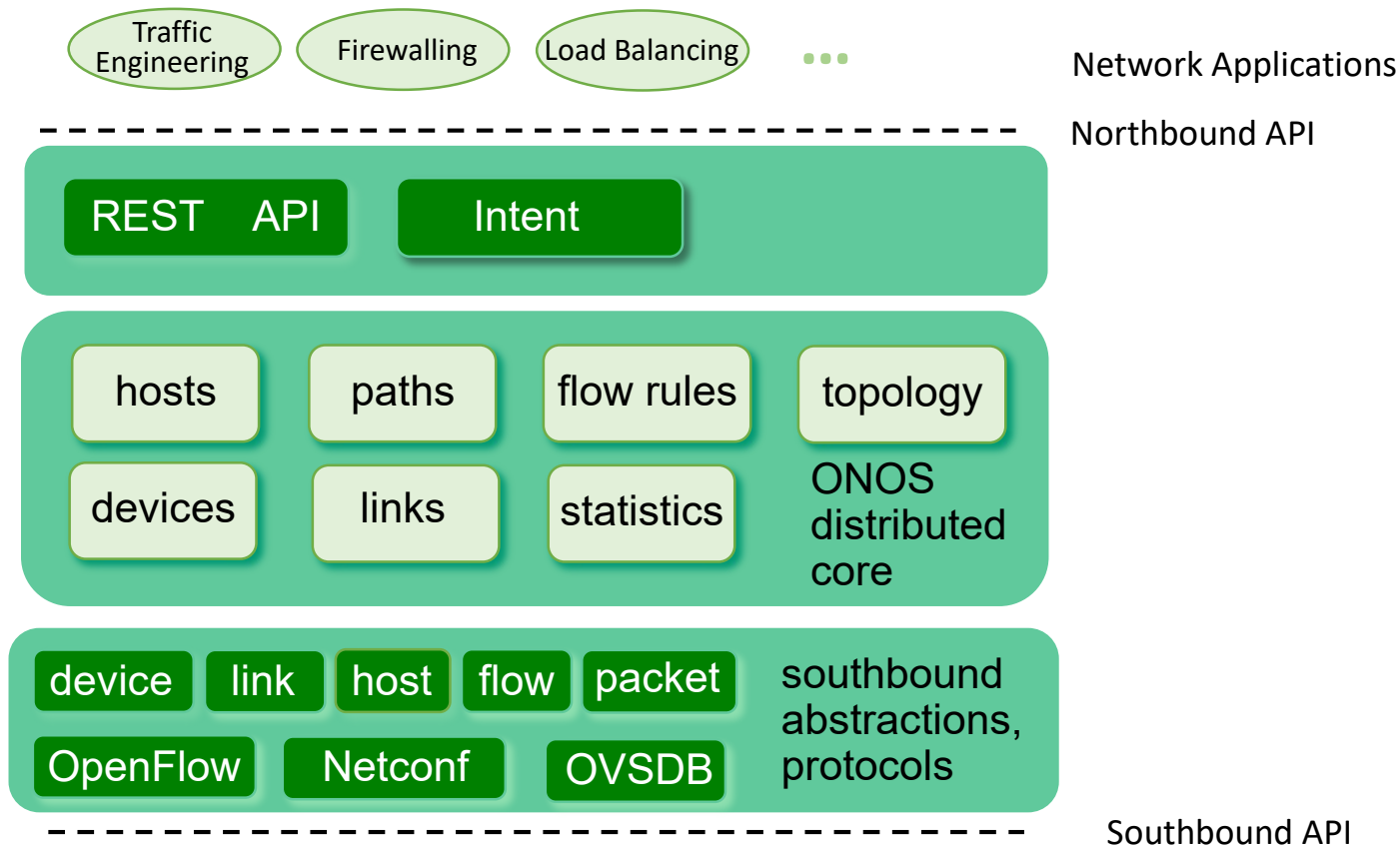
OpenDaylight (ODL) controller



Service Abstraction Layer:

- interconnects internal, external applications and services

ONOS controller



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

SDN: selected challenges

- **Hardening the control plane:** dependable, reliable, performance-scalable, and secure distributed system
 - **robustness to failures:** leverage strong theory of reliable distributed system for control plane
 - **Dependability and security:** “baked in” from day one?
- **Quality of Service (QoS):** networks and protocols meeting mission-specific requirements
 - e.g., real-time (low latency), ultra-reliable, ultra-secure
- **Scalability:** beyond a single AS
- SDN critical in 5G cellular networks (e.g., network slicing)

SDN and the future of traditional network protocols

- SDN-computed (at **some routers**) versus router-computed(at all routers) forwarding tables:
 - just one example of logically-centralized-computed versus physically distributed computed
- one could imagine SDN-computed congestion control:
 - controller sets sender rates based on router-reported (to controller) congestion levels → make network more controllable



How will implementation of network functionality (SDN versus protocols) evolve?

