

Chapter 5

Network Layer:

Control Plane

Network layer control plane: our goals

- understand principles behind network control plane:
 - traditional routing algorithms
 - SDN controllers
 - network management, configuration
- instantiation, implementation in the Internet:
 - OSPF, BGP
 - OpenFlow, ODL and ONOS controllers
 - Internet Control Message Protocol: ICMP
 - SNMP, YANG/NETCONF

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

Network-layer functions

- **forwarding:** move packets from router's input to appropriate router output
- **routing:** determine route taken by packets from source to destination

data plane

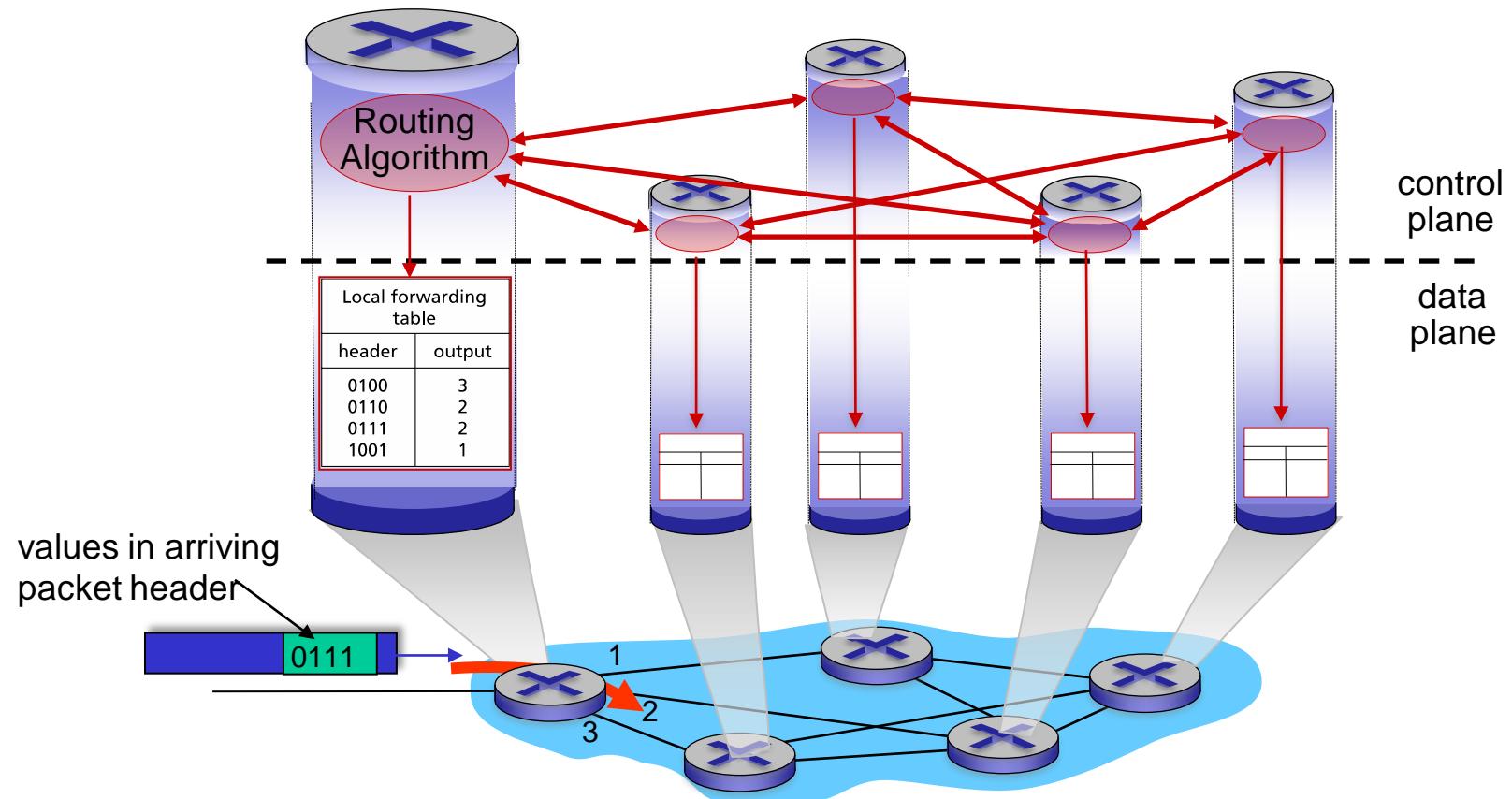
control plane

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

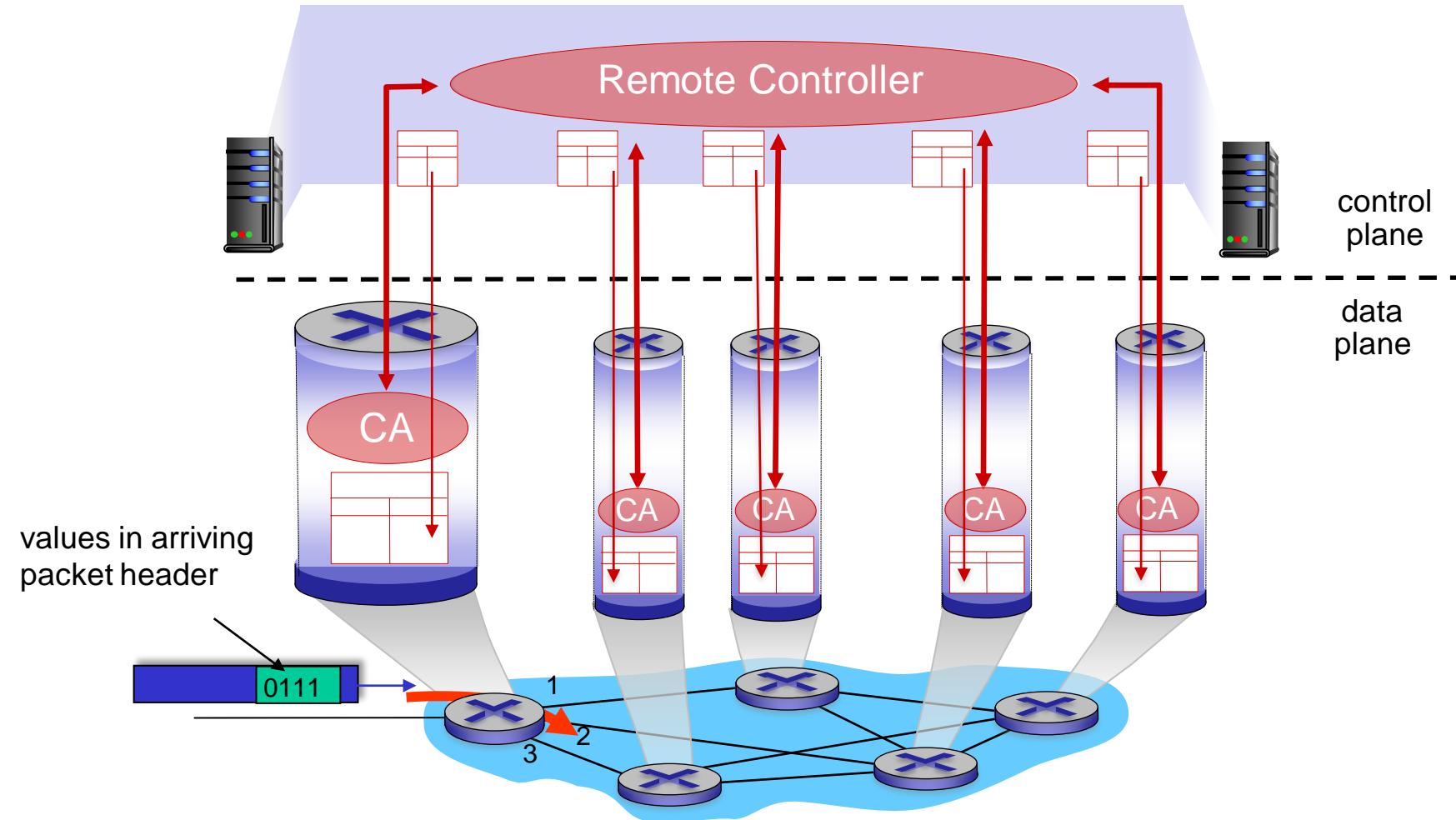
Per-router control plane

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



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

lab1

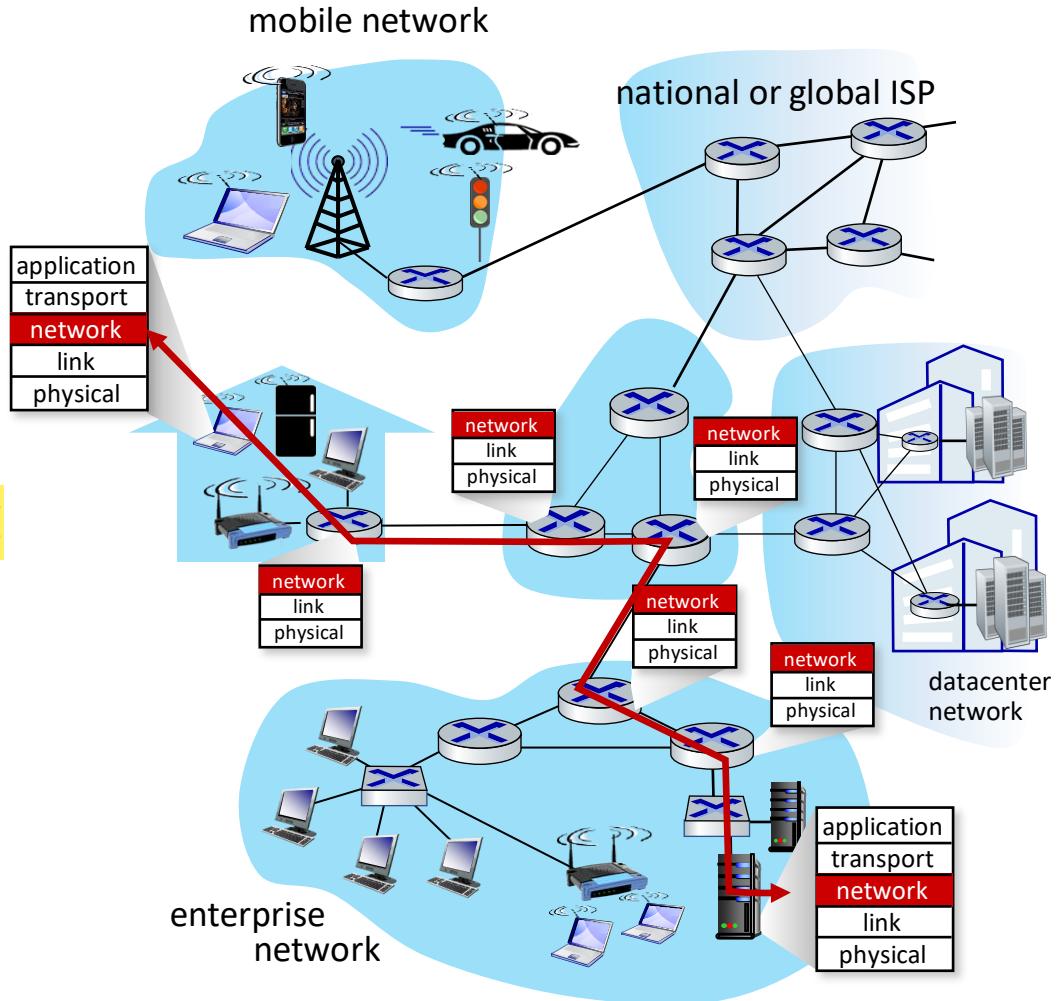


- network management, configuration
 - SNMP
 - NETCONF/YANG

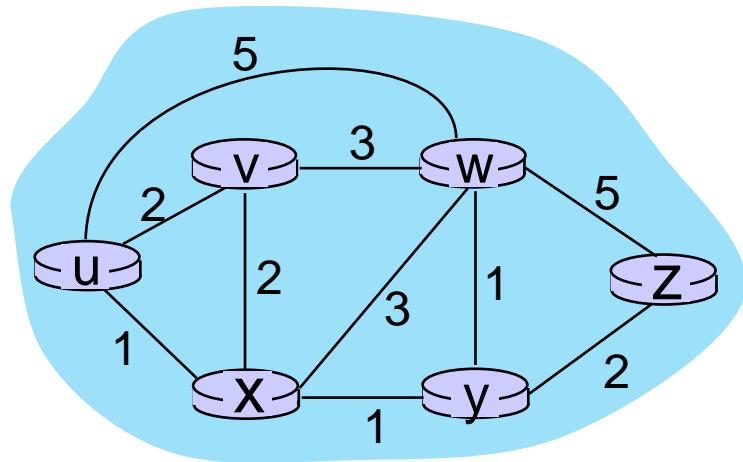
Routing protocols

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



graph: $G = (N, E)$

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

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

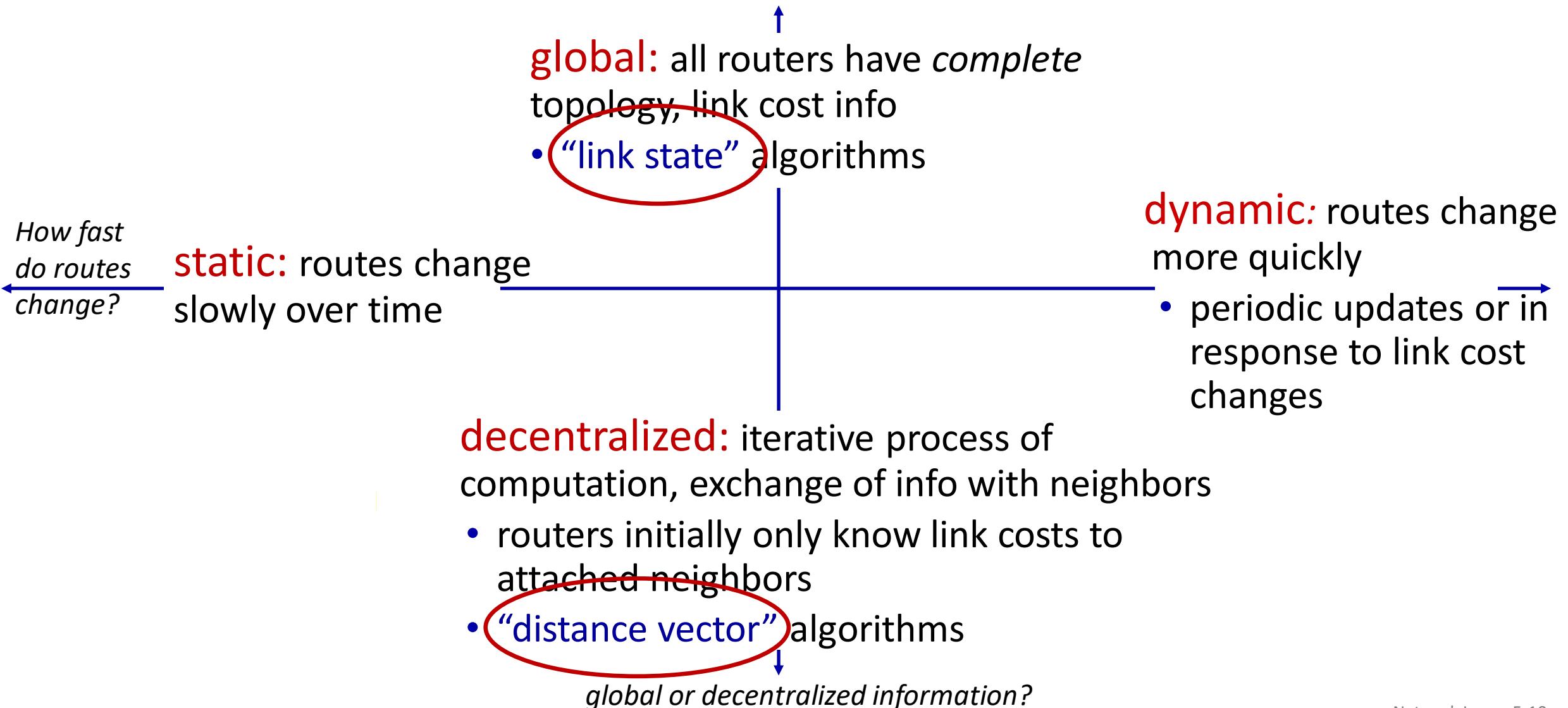
$c_{a,b}$: cost of *direct* link connecting a and b

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

Routing algorithm classification



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

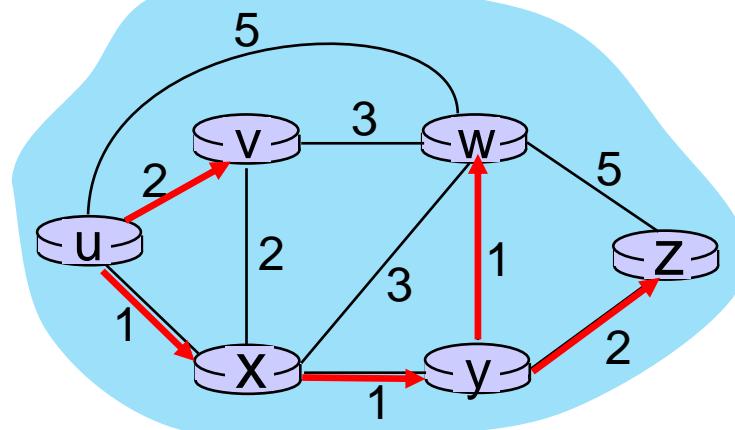
- $c_{x,y}$: direct link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: *current* estimate of cost of least-cost-path from source to destination v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least-cost-path *definitively* known

Dijkstra's link-state routing algorithm

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

Dijkstra's algorithm: an example

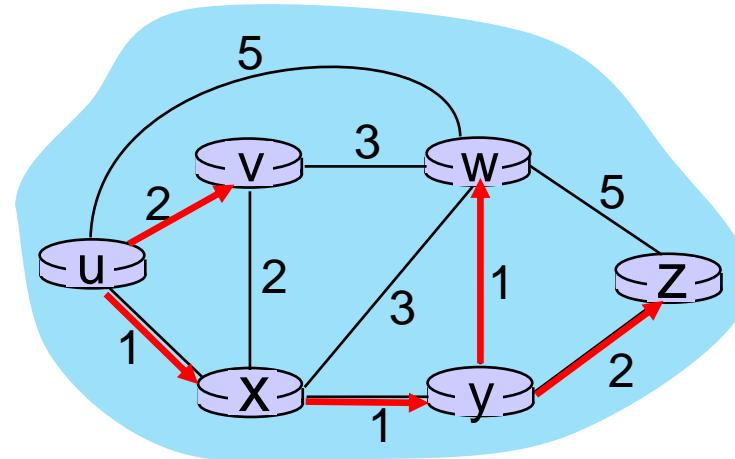
Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	u, x	2, u	4, x	2, x	∞	∞
2	u, x, y	2, u	3, y		4, y	
3	u, x, y, v		3, y		4, y	
4	u, x, y, v, w					4, y
5	u, x, y, v, w, z					



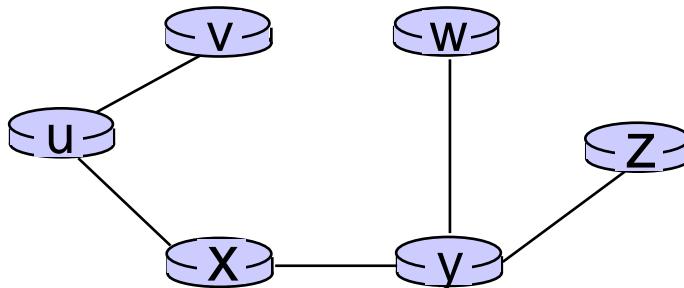
Initialization (step 0): For all a : if a adjacent to u then $D(a) = c_{u,a}$

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:

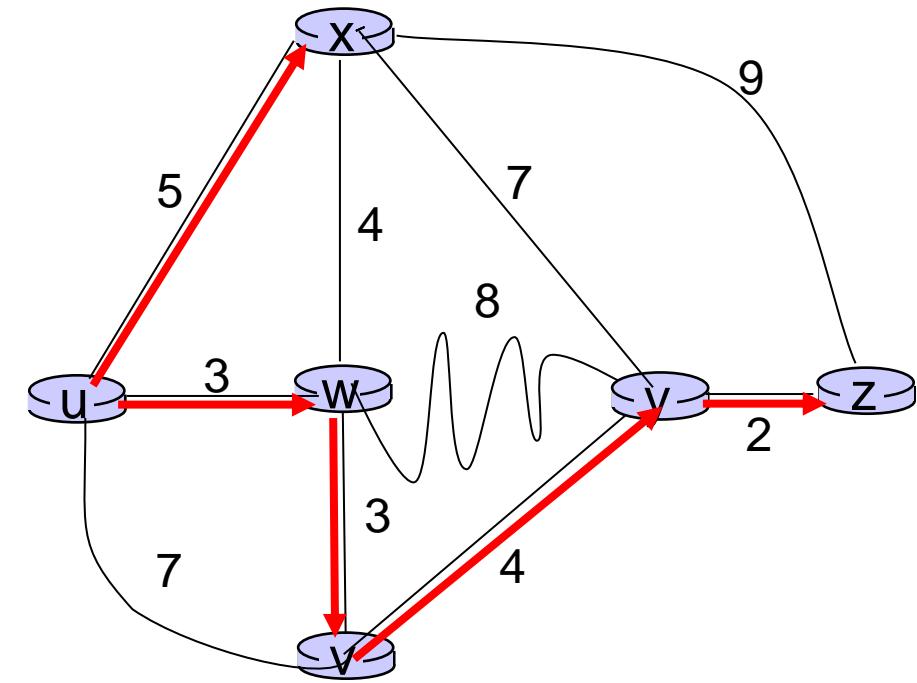
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from u to v directly

route from u to all other destinations via x

Dijkstra's algorithm: another example

Step	N'	v	w	x	y	z
0	u	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
1	uw	$7, u$	$3, u$	$5, u$	∞	∞
2	uwx	$6, w$	$5, u$	$11, w$	∞	∞
3	$uwxv$	$6, w$	$11, w$	$14, x$	∞	∞
4	$uwxvy$	$10, v$	$14, x$	∞	$12, y$	∞
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

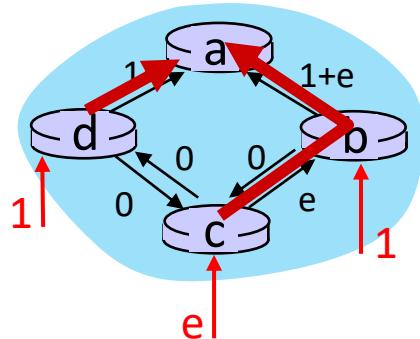
- each of n iteration: need to check all nodes, w , not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n \log n)$

message complexity:

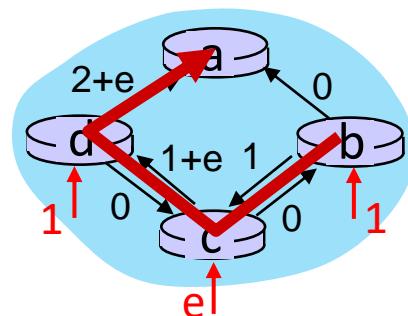
- each router must *broadcast* its link state information to other n routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

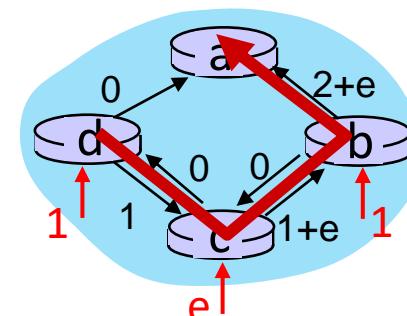
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
 - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



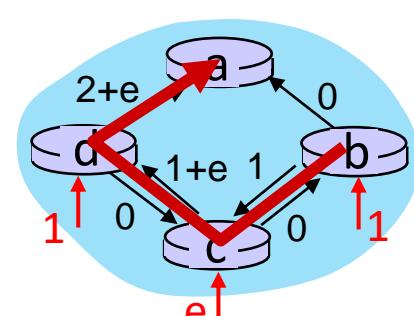
initially



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:

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

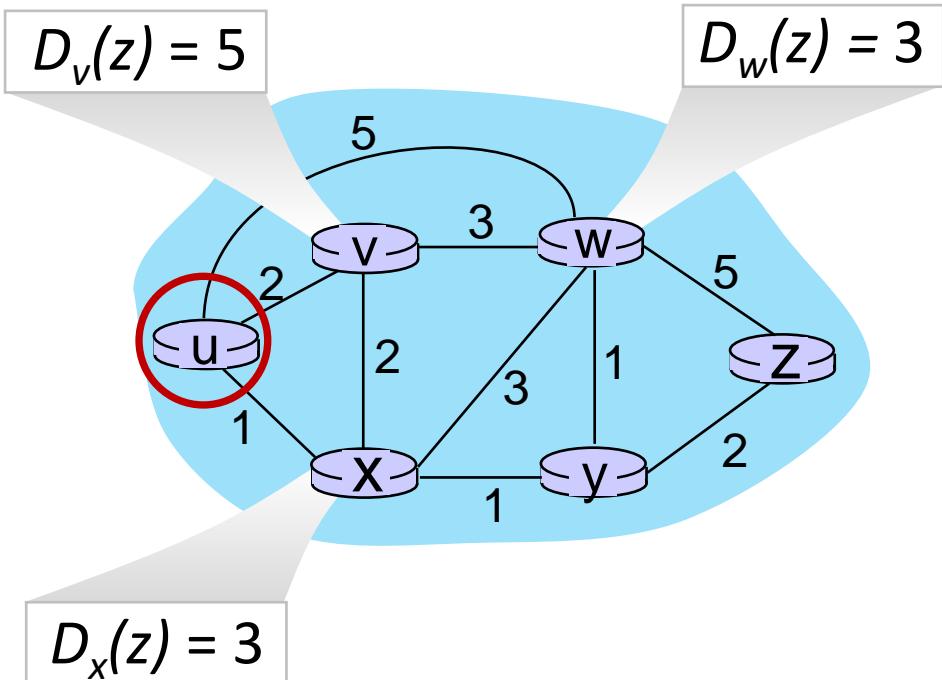
\min taken over all neighbors v of x

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

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:

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

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

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 $d_x(y)$

Distance vector algorithm:

each node:

-
- ```
graph TD; A[wait for (change in local link cost or msg from neighbor)] --> B[recompute DV estimates using DV received from neighbor]; B --> C;if DV to any destination has changed, notify neighbors
```
- 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!

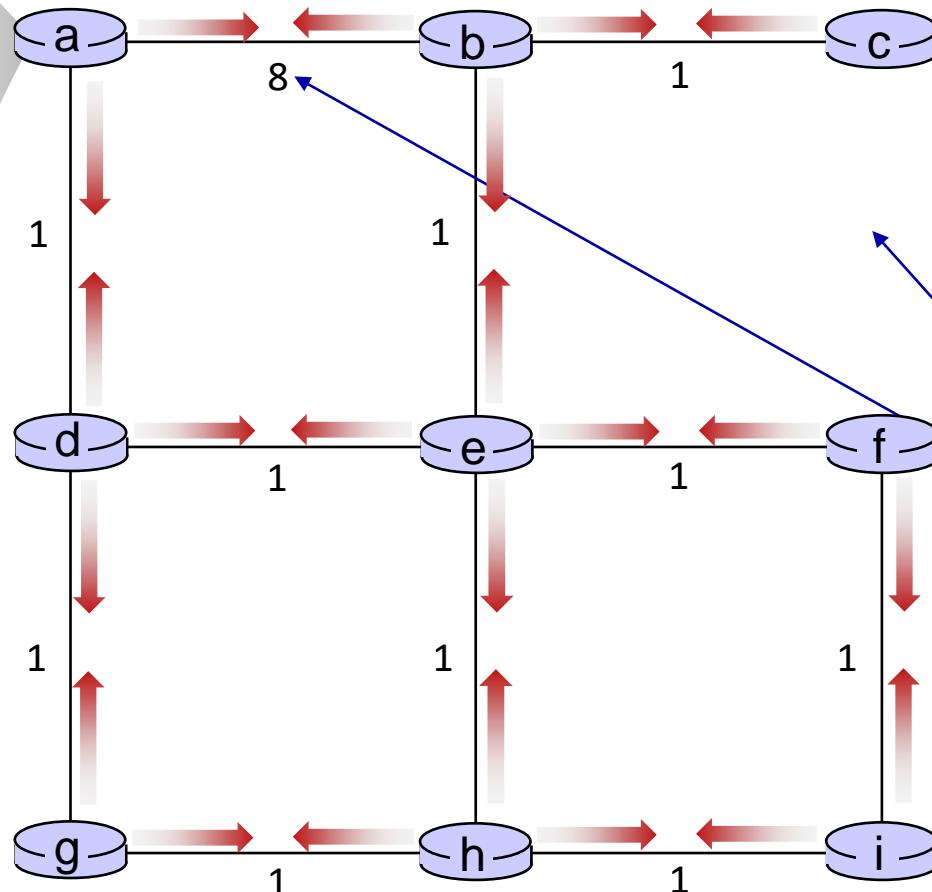
# Distance vector: example



$t=0$

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

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



- A few asymmetries:
- missing link
  - larger cost

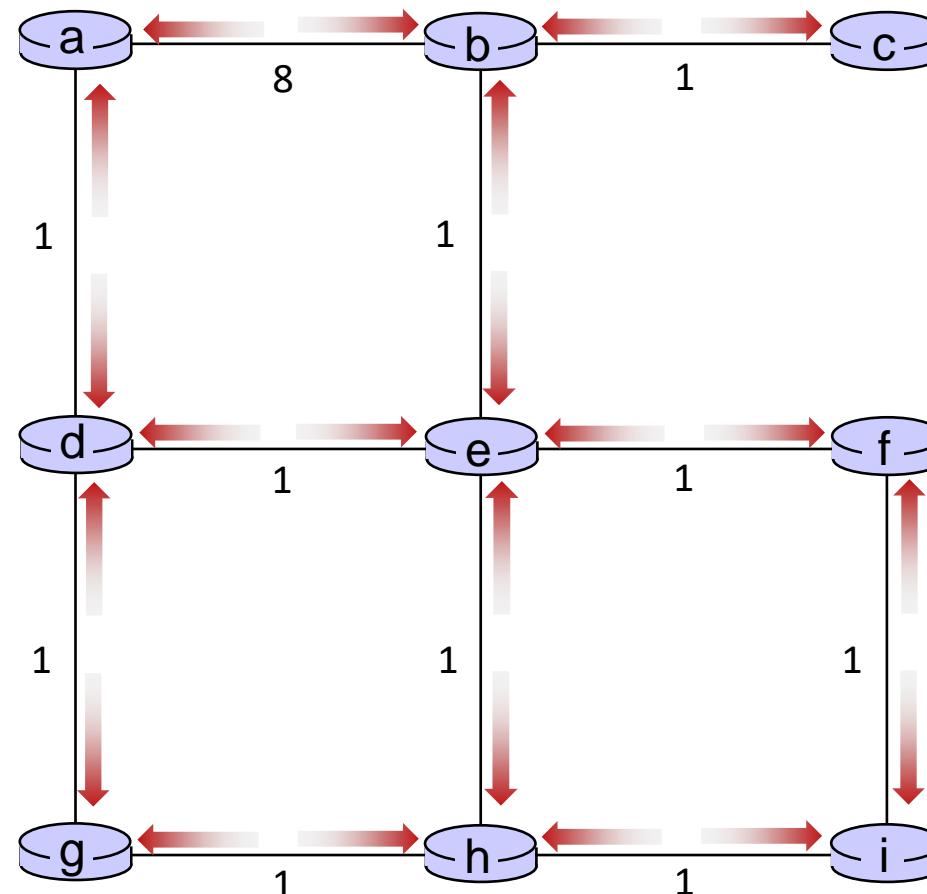
# Distance vector example: iteration



$t=1$

All nodes:

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



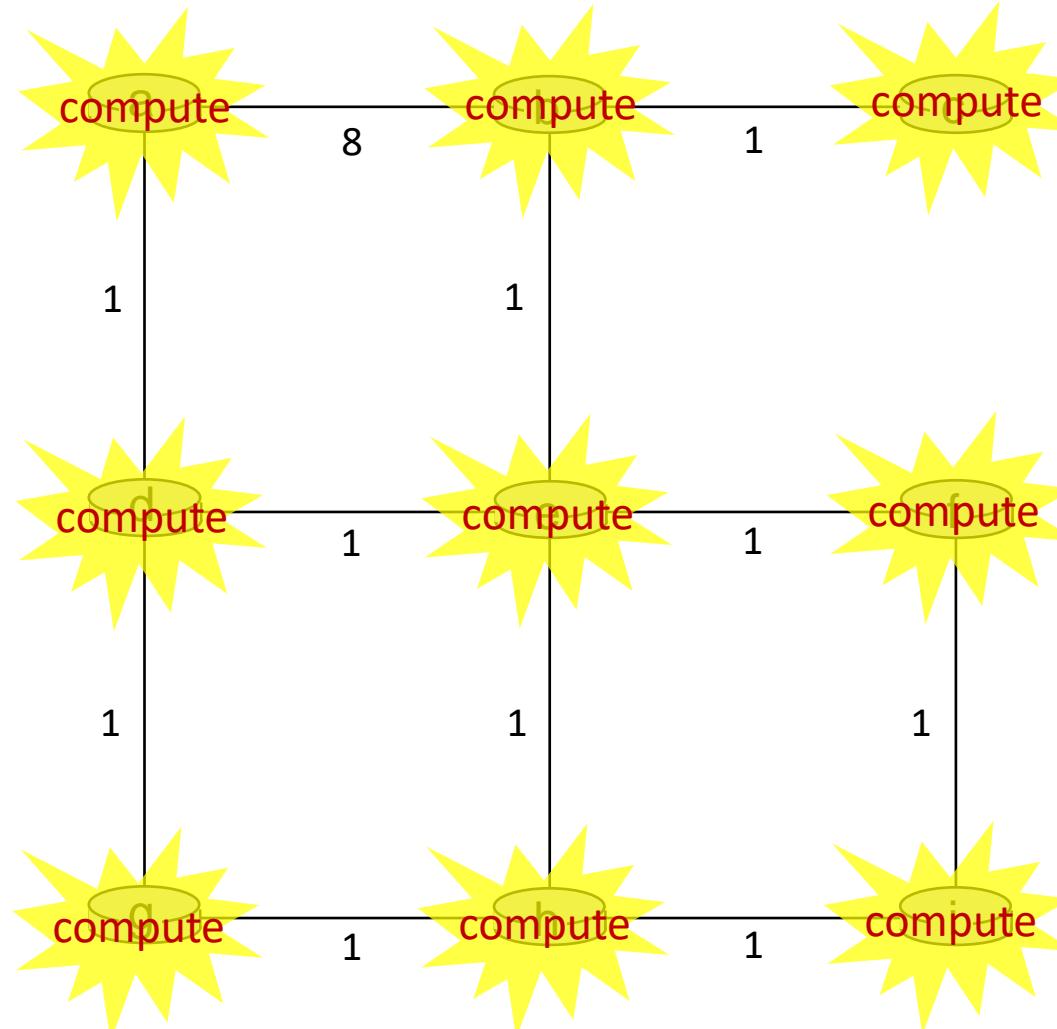
# Distance vector example: iteration



**t=1**

All nodes:

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



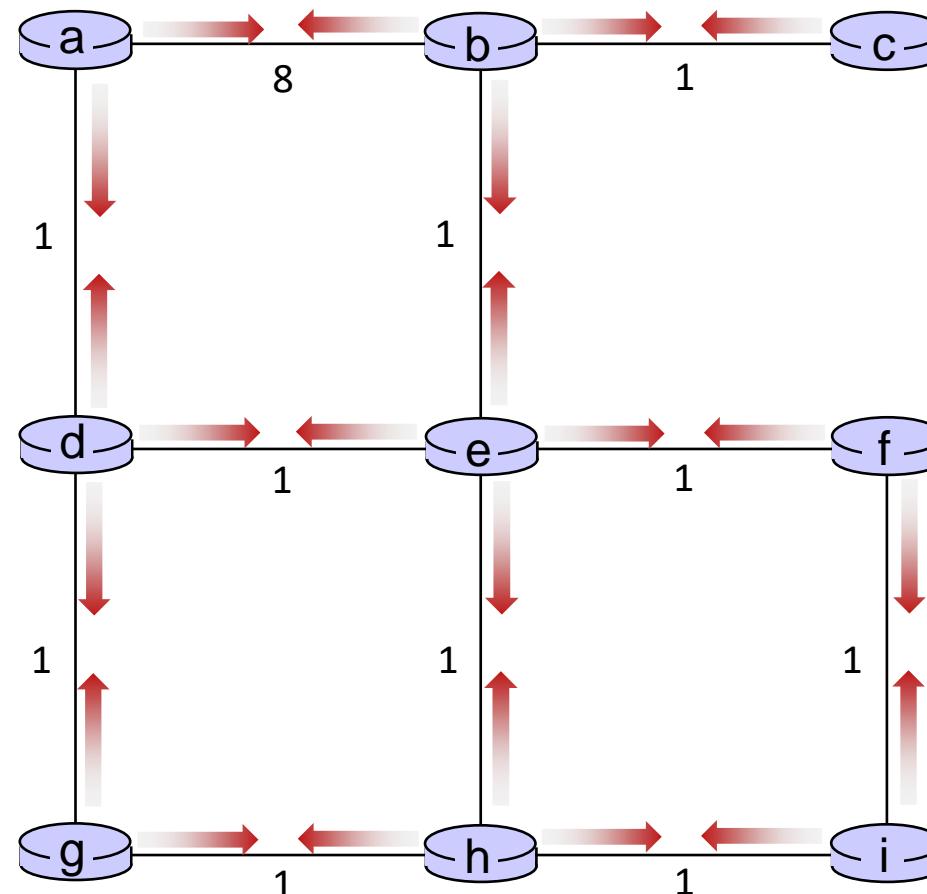
# Distance vector example: iteration



**t=1**

All nodes:

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



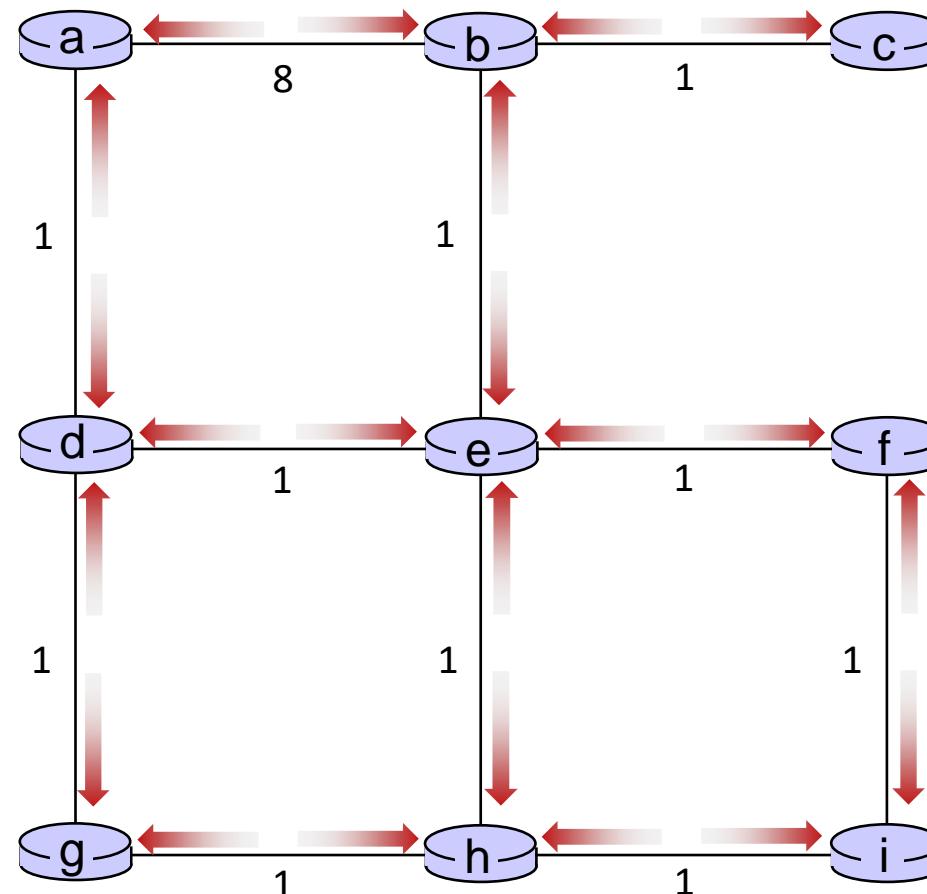
# Distance vector example: iteration



$t=2$

All nodes:

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



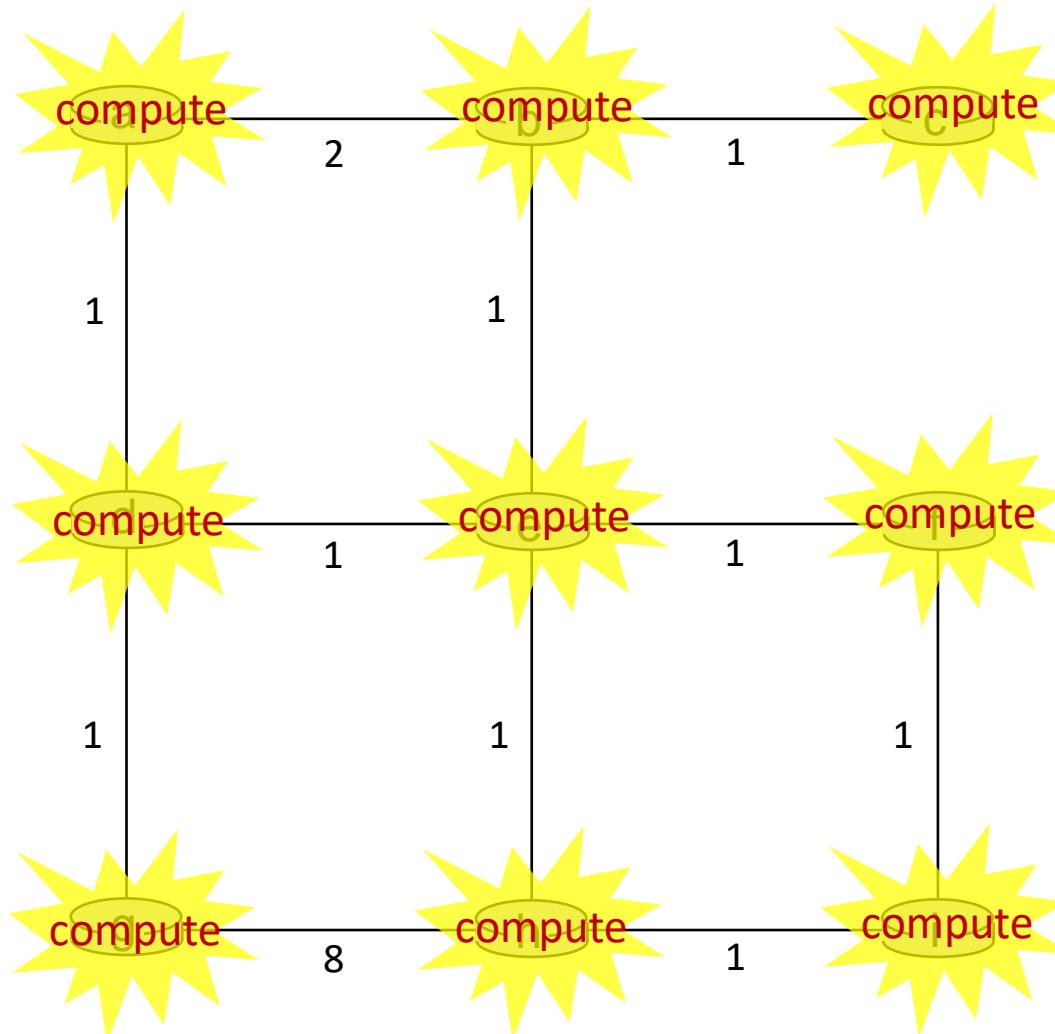
# Distance vector example: iteration



$t=2$

All nodes:

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



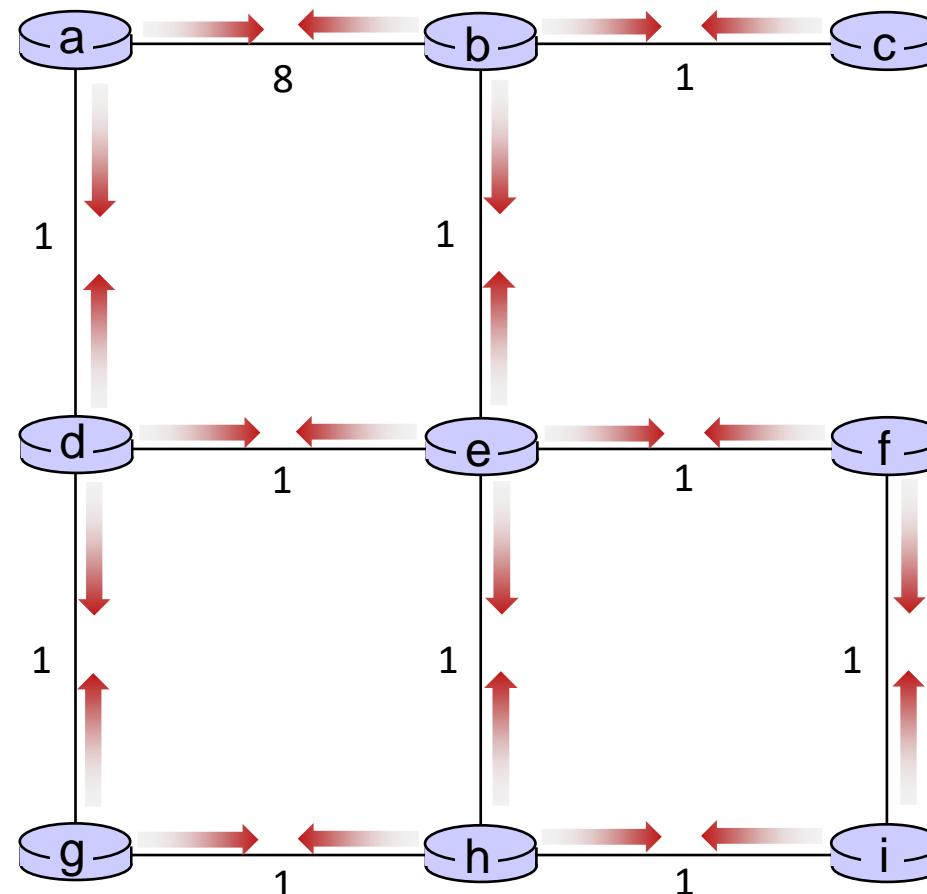
# Distance vector example: iteration



$t=2$

All nodes:

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



# Distance vector example: iteration

.... and so on

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

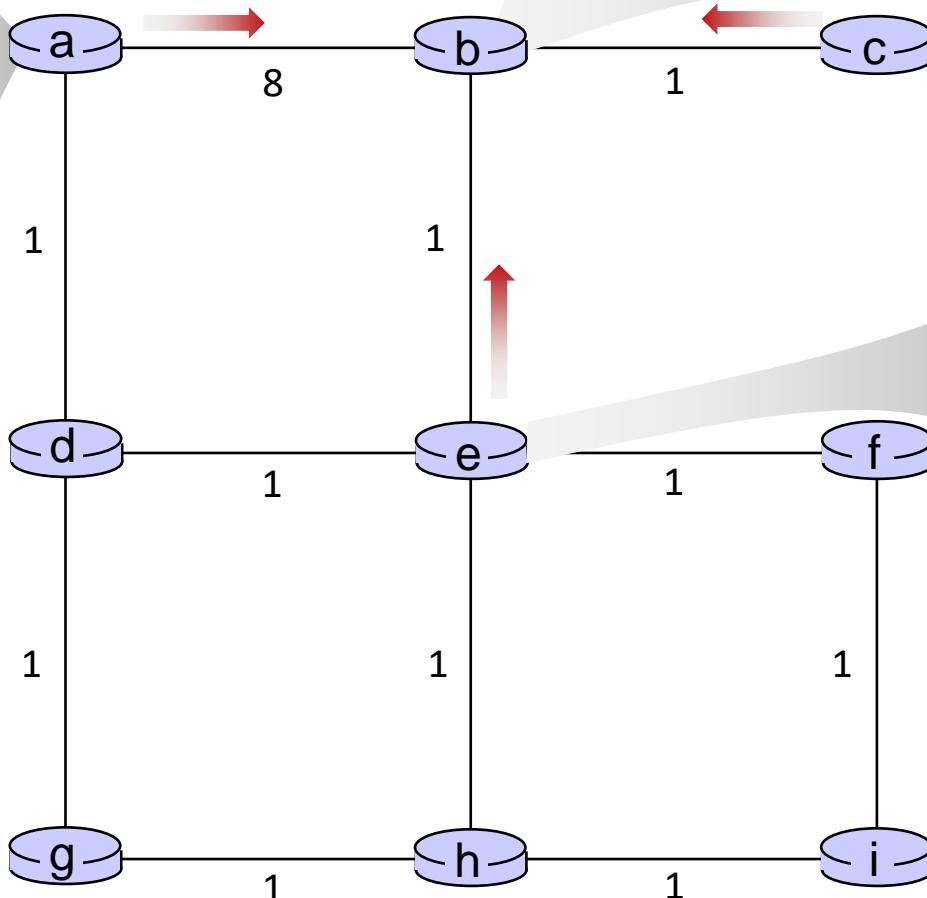
# Distance vector example: t=1



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

| 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 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

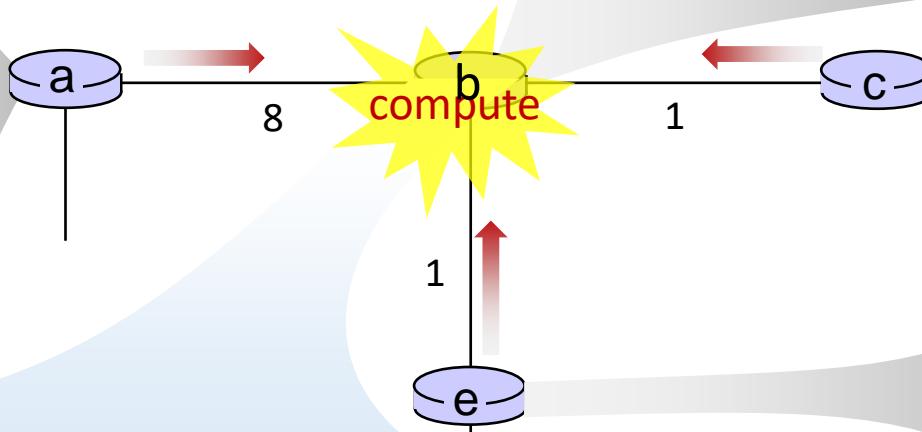


**t=1**

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

| 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 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:     |                   |
|--------------|-------------------|
| $D_b(a) = 8$ | $D_b(f) = 2$      |
| $D_b(c) = 1$ | $D_b(g) = \infty$ |
| $D_b(d) = 2$ | $D_b(h) = 2$      |
| $D_b(e) = 1$ | $D_b(i) = \infty$ |

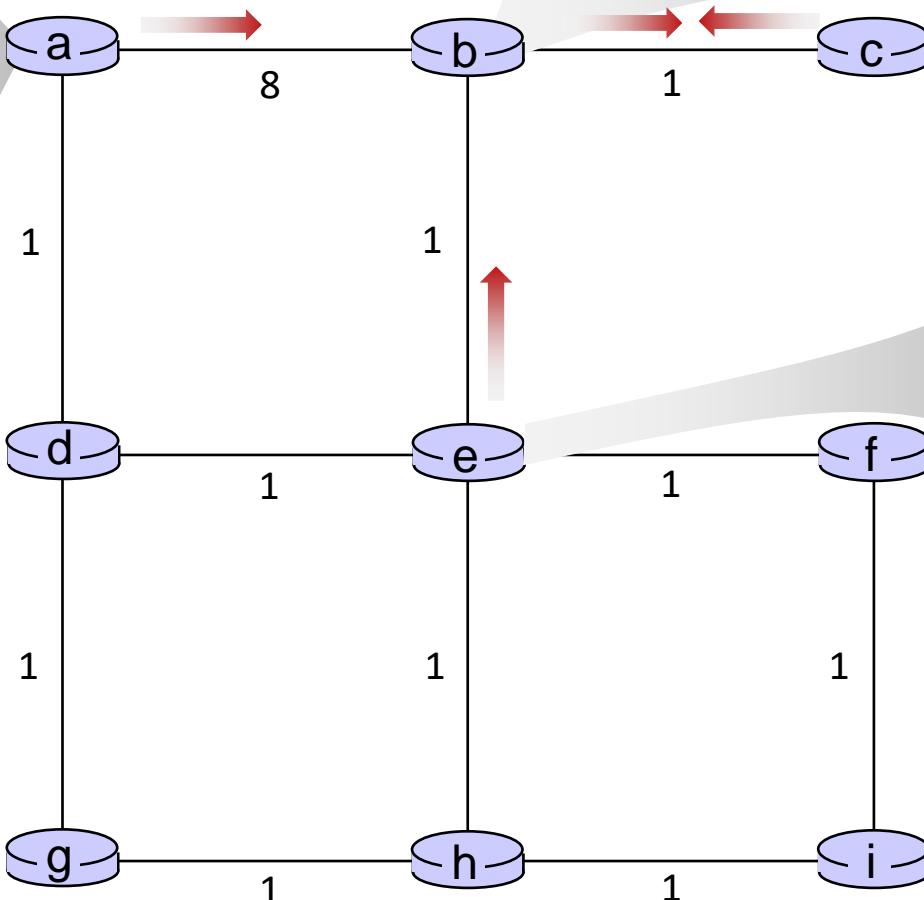
# Distance vector example: t=1



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

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

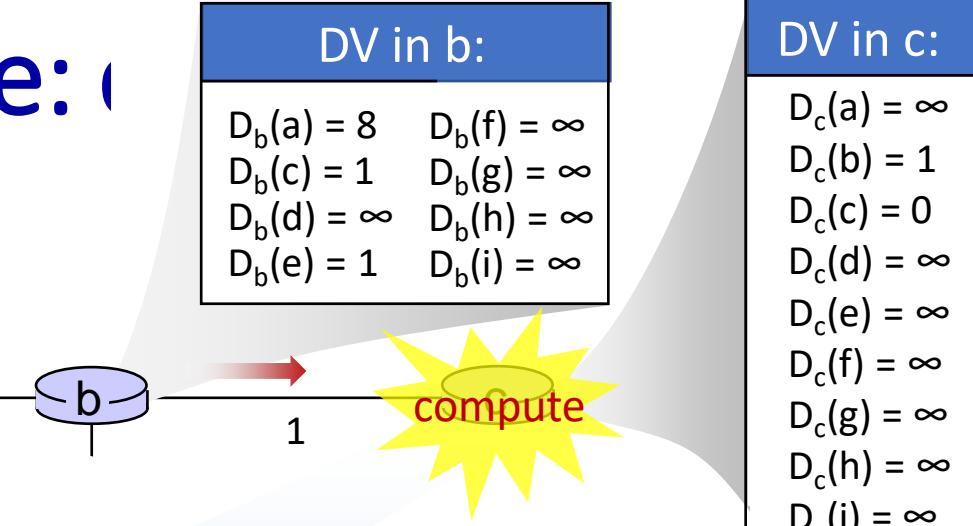
# Distance vector example: (t=1)



t=1

- c receives DVs from b computes:

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



DV in c:

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

\* Check out the online interactive exercises for more examples:  
[http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Distance vector example: t=1

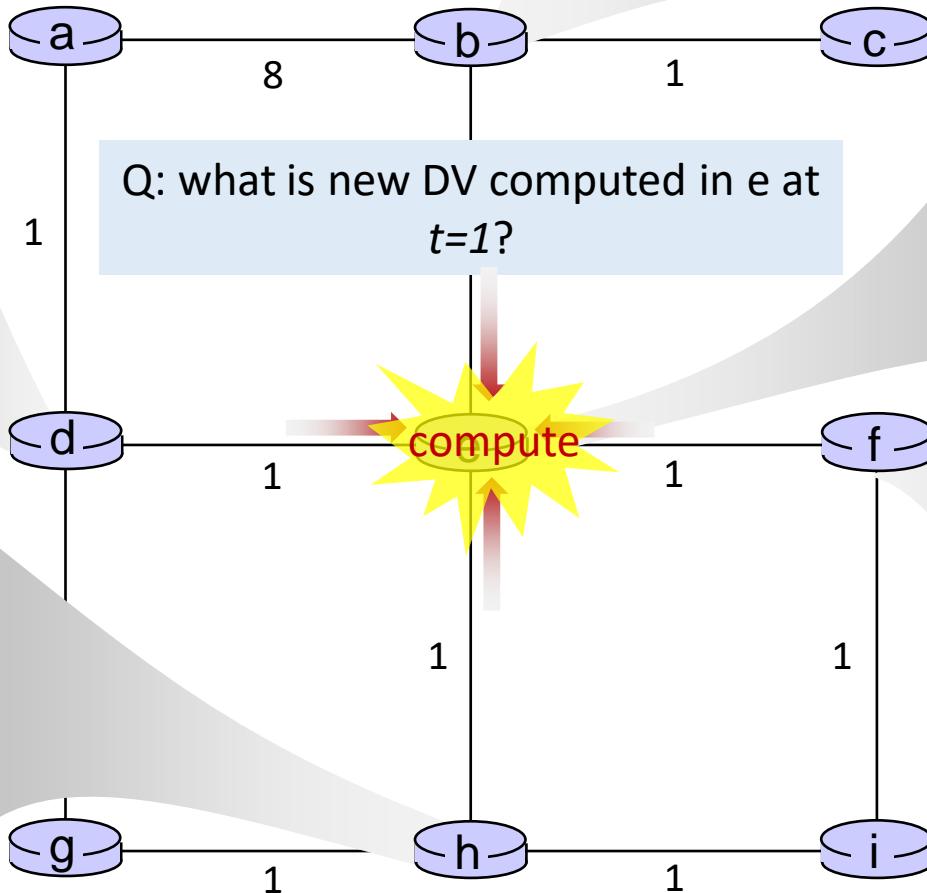


**t=1**

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

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

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



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

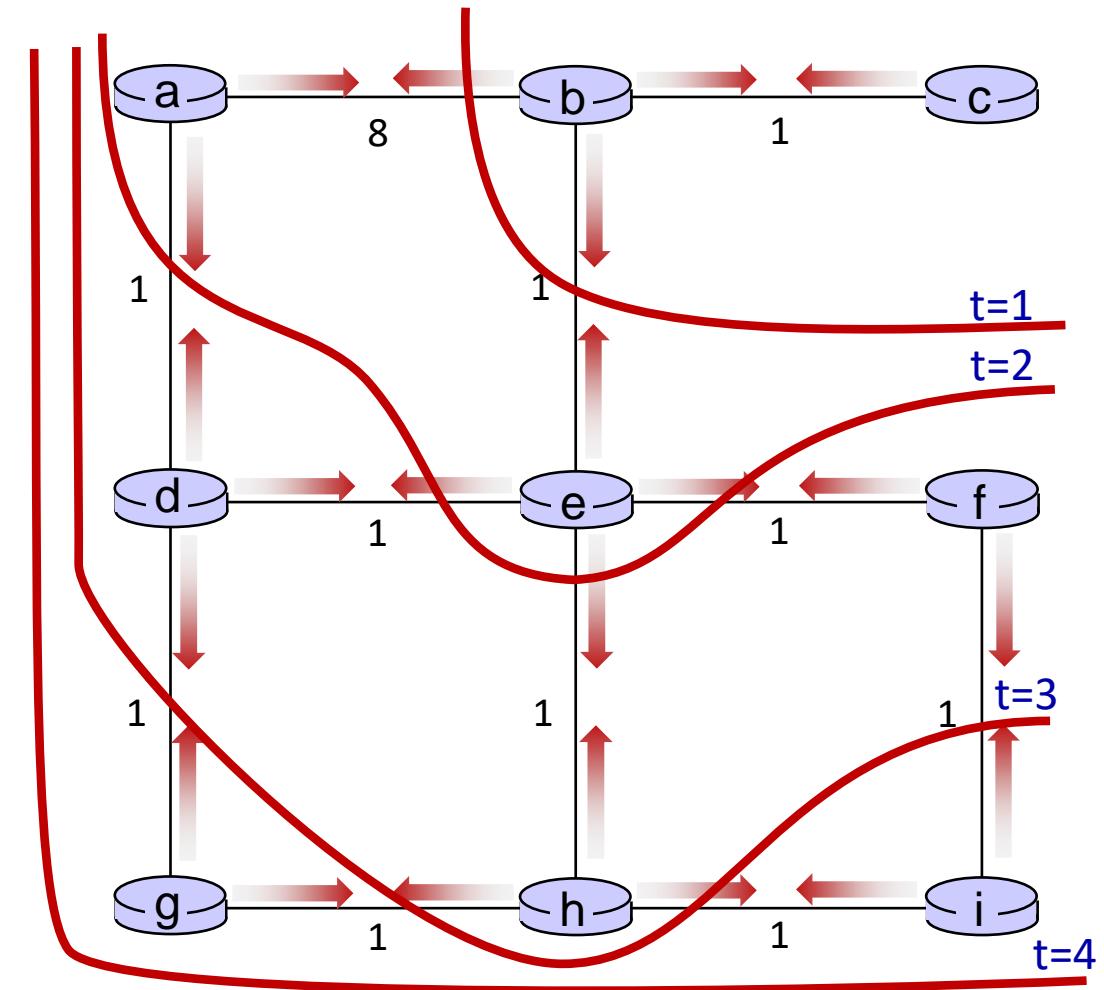
| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

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

# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

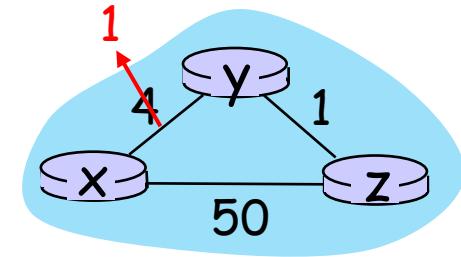
-  t=0 c's state at t=0 is at c only
-  t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  t=3 c's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



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

“good news travels fast”       $t_1$  :  $z$  receives update from  $y$ , updates its table, computes new least cost to  $x$ , sends its neighbors its DV.

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

# Distance vector: link cost changes

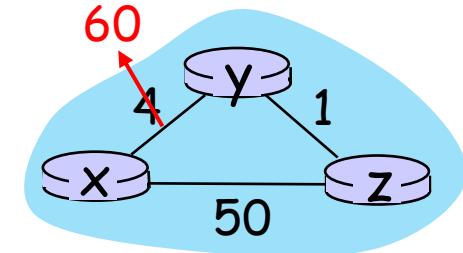
## link cost changes:

- node detects local link cost change
- “bad news travels slow” – count-to-infinity

problem: 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 table used by others: error propagate thru network

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- **intra-ISP routing: OSPF**
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# 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”)

*AS = Very big subsets (country)  
= associated to a number (yfb = 1 AS)*

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

*↳ IN*

- 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)

*↳ between the AS routing  
=====*

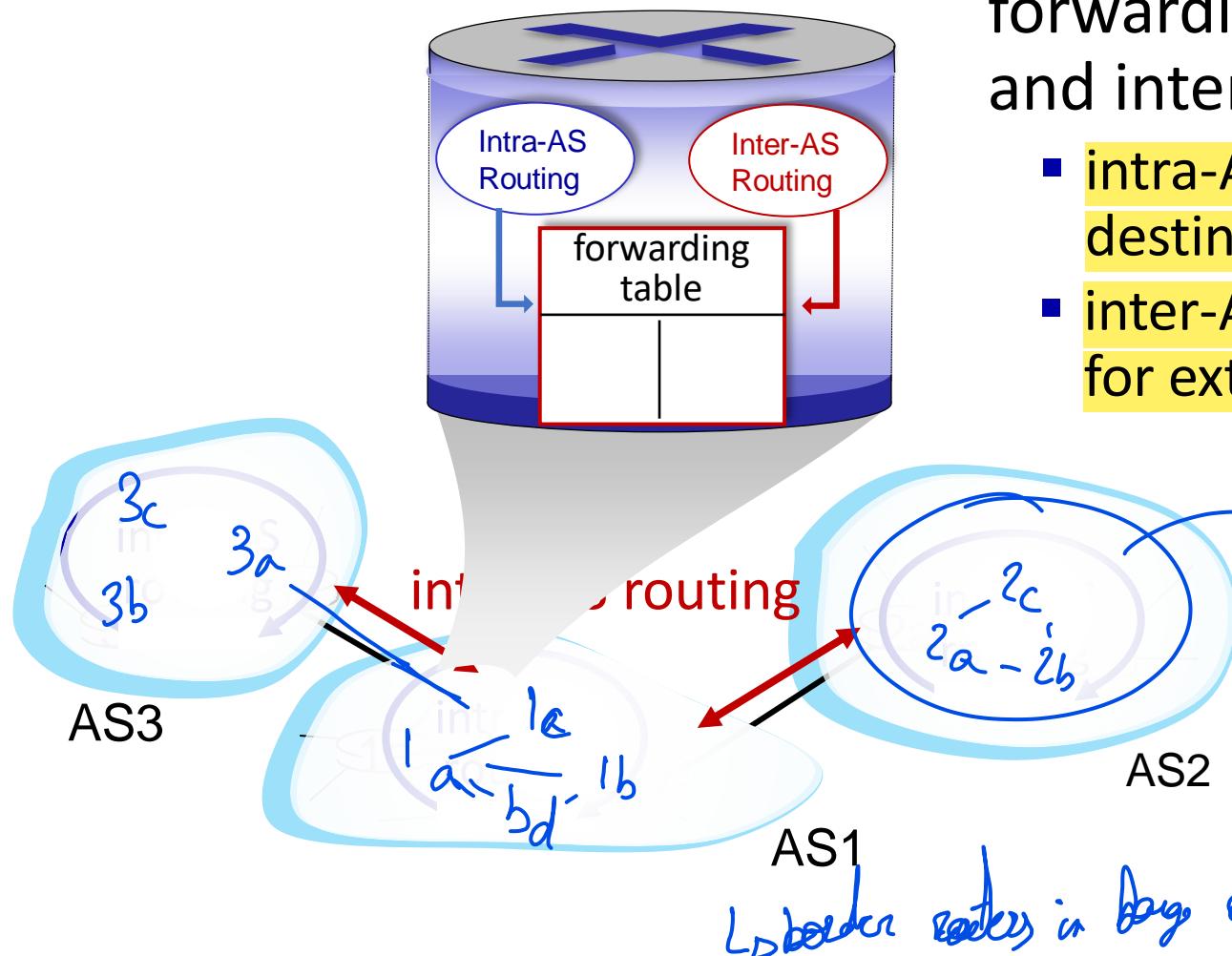
# Interconnected ASes

regroup all those routers will only appear as 1 in the routing table

IAS → All universities {  
    Ilie VUL  
    Ilie VUB  
    Ilie VLB } ... => SCALABLE

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



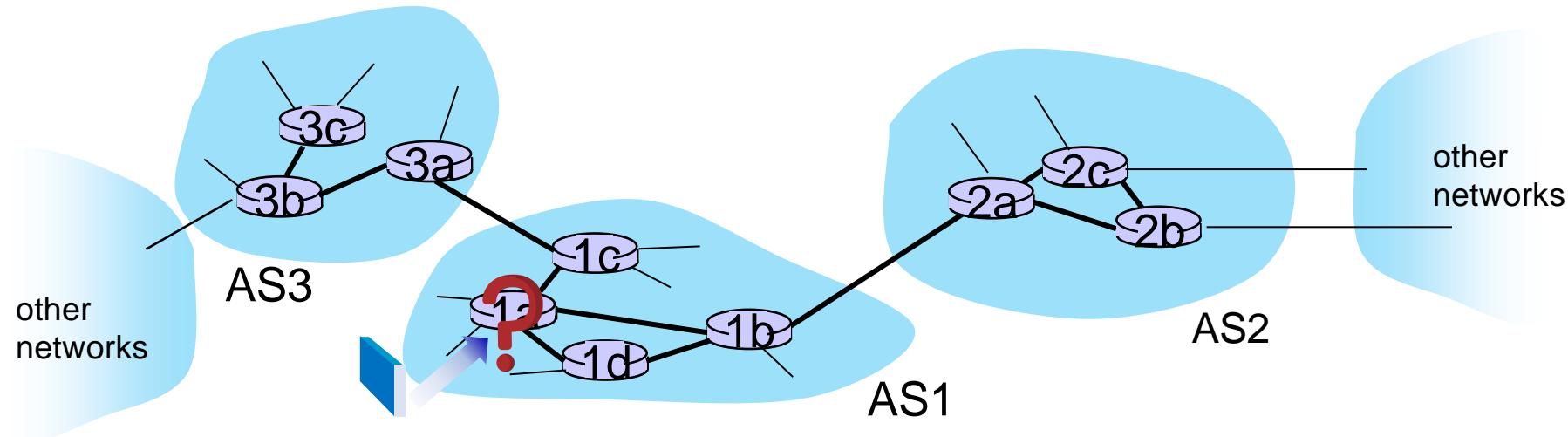
- Border routers : Inside INTRA
  - ↳ communicate with other AS

AS1 L<sub>D</sub> border borders in 3 ways of all of the domains

# 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?

**Touters en ASI**  
Avantage: Aller de 3 à 2, puis de 2 à 1. Ce qui il se passe entre n'est pas de pb (chaque ASI a son propre système de gestion (Dijkstra, ...))



# 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

# Inter-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

*Border gateway  
protocol*  
*→ used a lot*  
*↳ faster + more optimum for BGP*

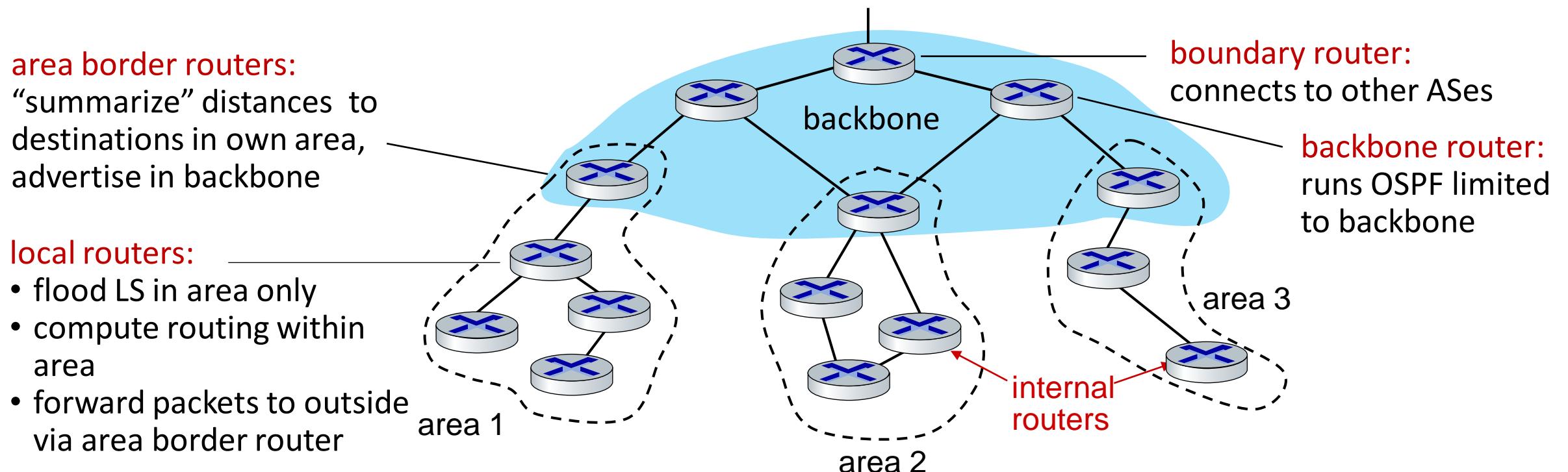
# OSPF (Open Shortest Path First) routing

- “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



# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- **routing among ISPs: BGP**
- SDN control plane
- Internet Control Message Protocol

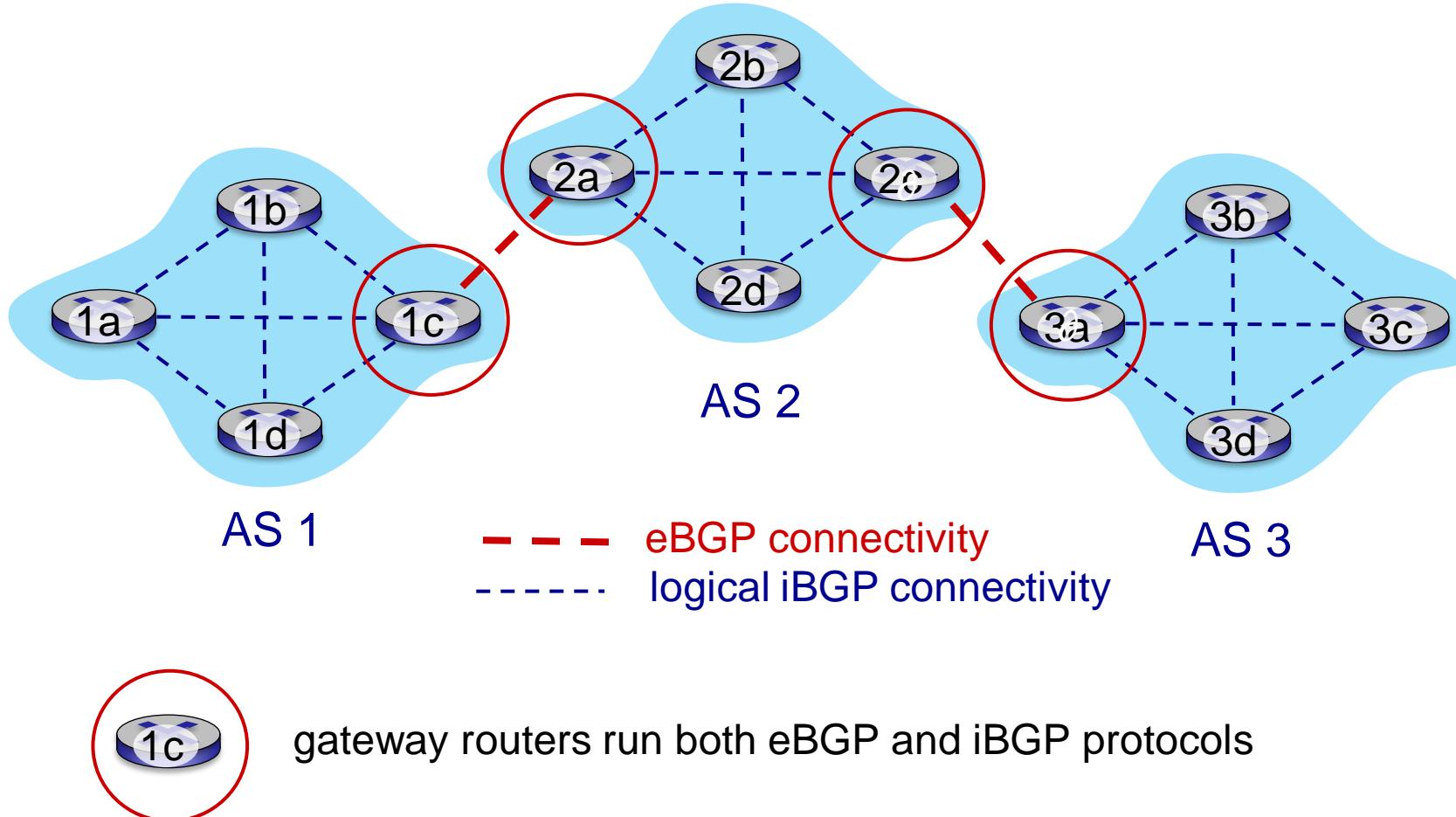


- network management, configuration
  - SNMP
  - NETCONF/YANG

# Internet inter-AS routing: BGP

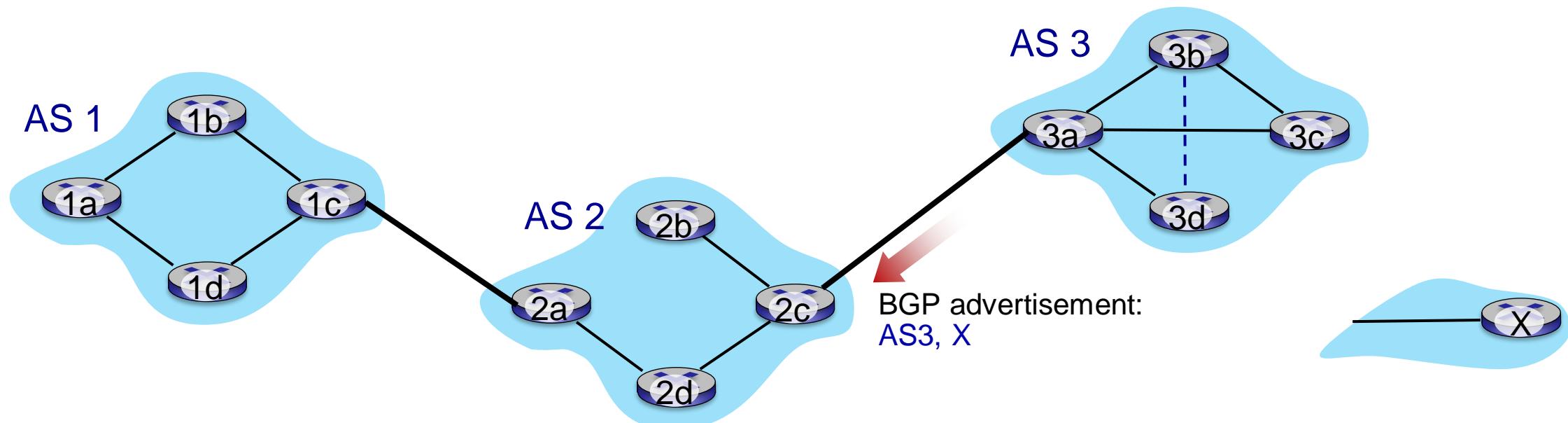
- BGP (Border Gateway Protocol): *the de facto inter-domain routing protocol*
  - “glue that holds the Internet together”
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *“I am here, here is who I can reach, and how”*
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: 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

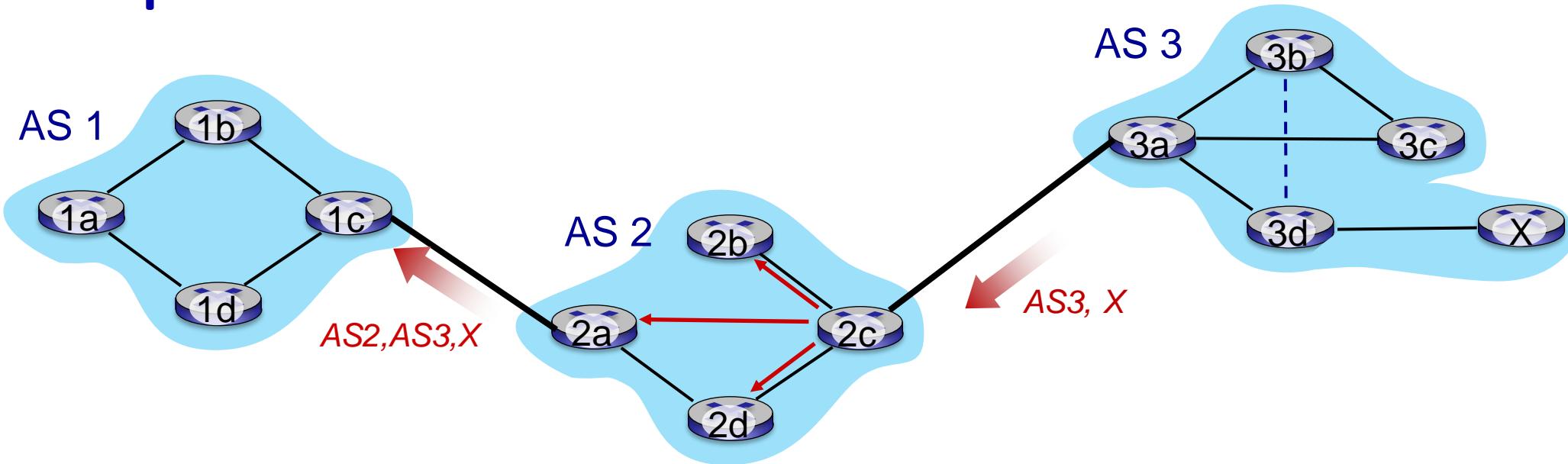
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway 3a advertises **path AS3,X** to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will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



# Path attributes and BGP routes

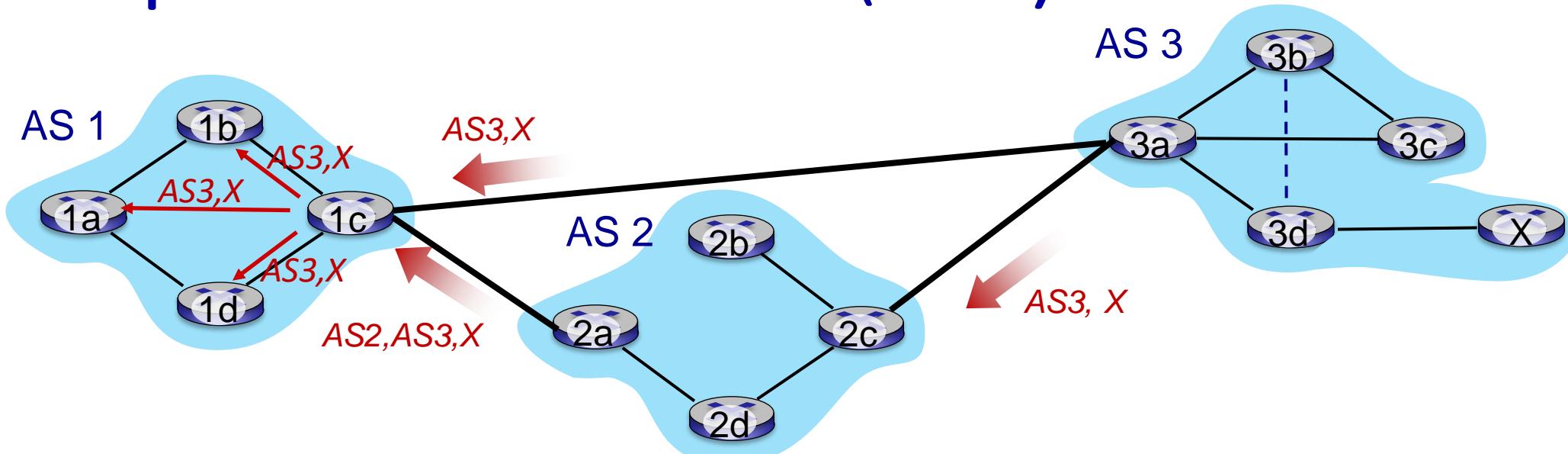
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- policy-based routing:
  - gateway receiving 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 other neighboring ASes

# 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)



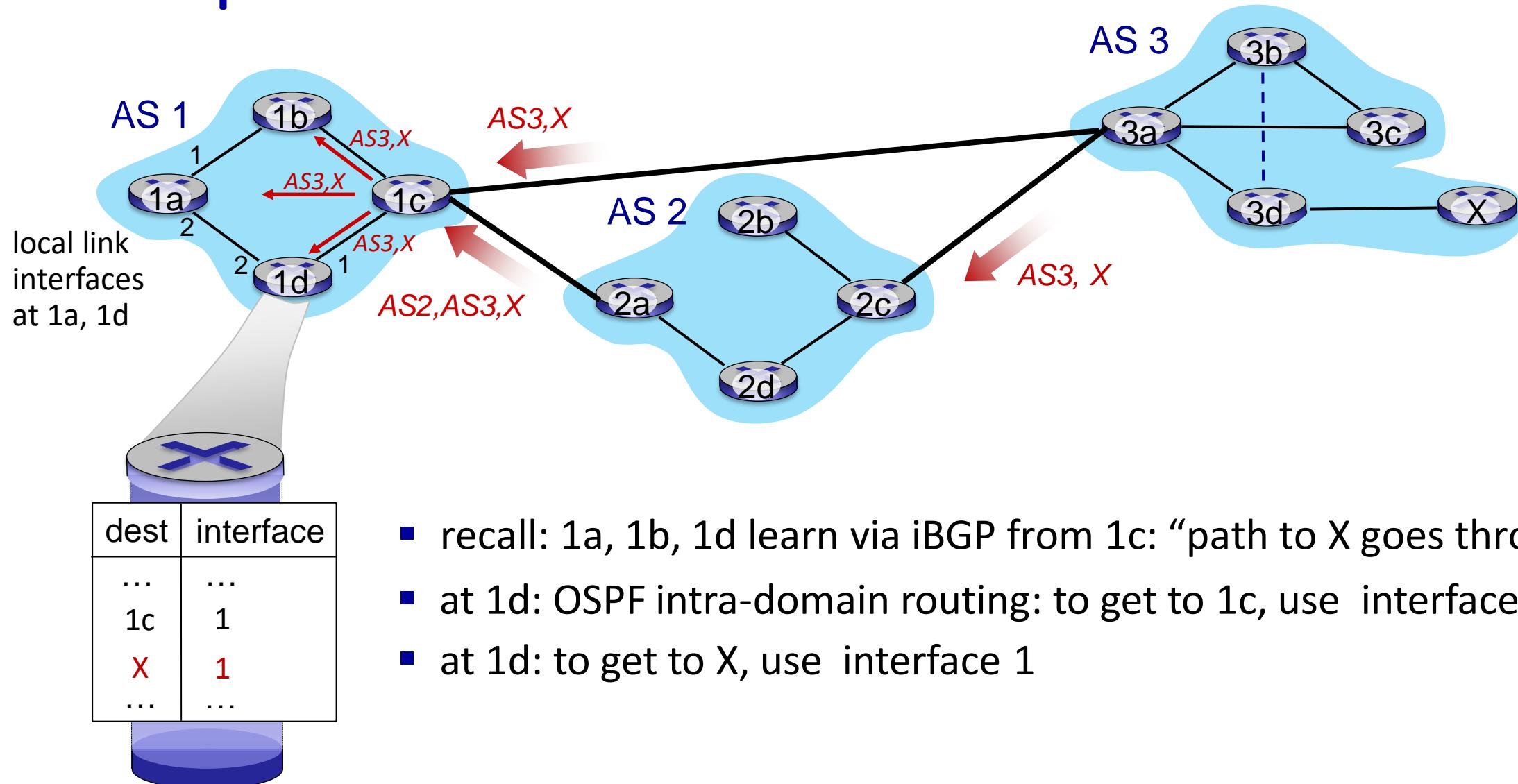
gateway router may learn about multiple paths to destination:

- 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

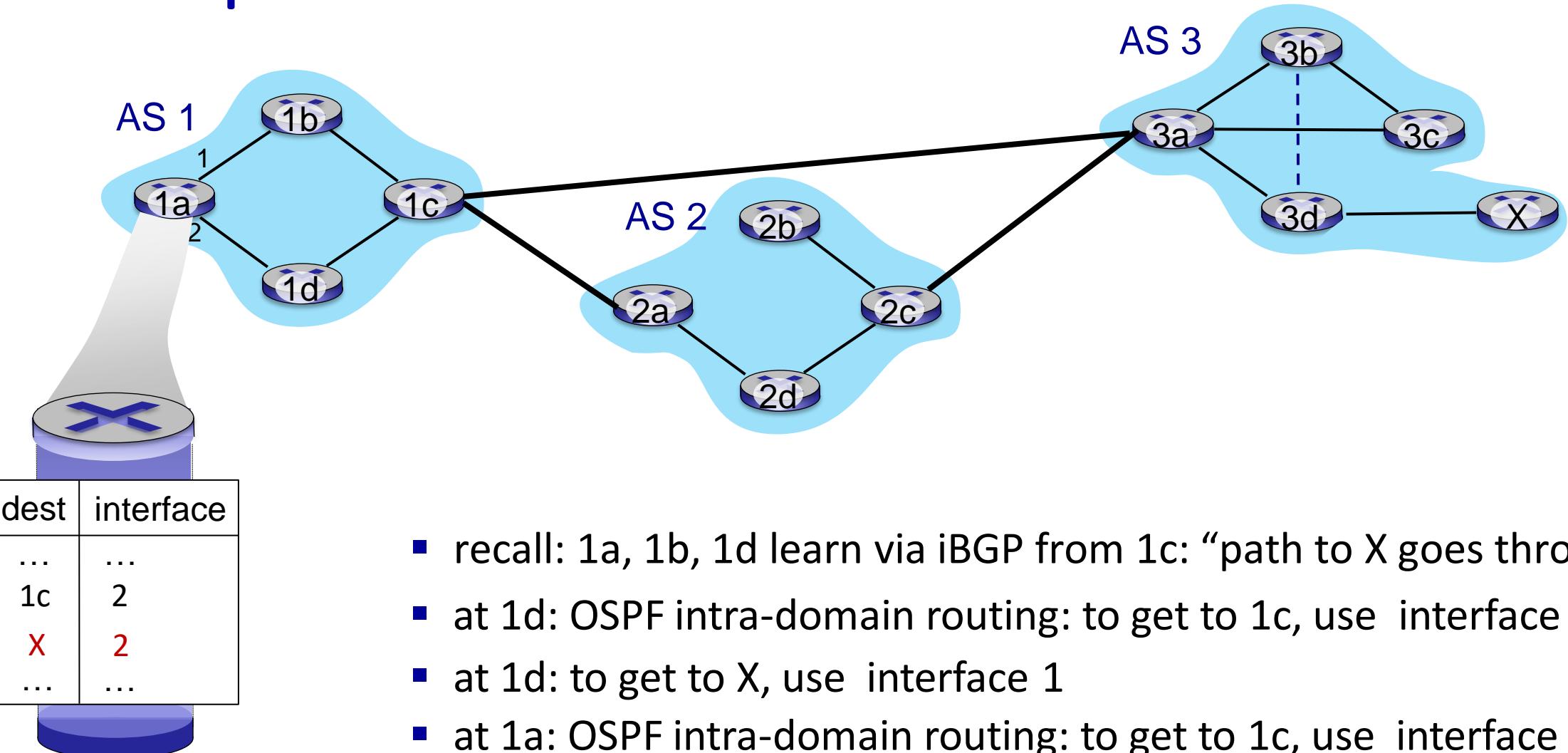
- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **UPDATE**: advertises new path (or withdraws old)
  - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request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

# BGP path advertisement



- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

# 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

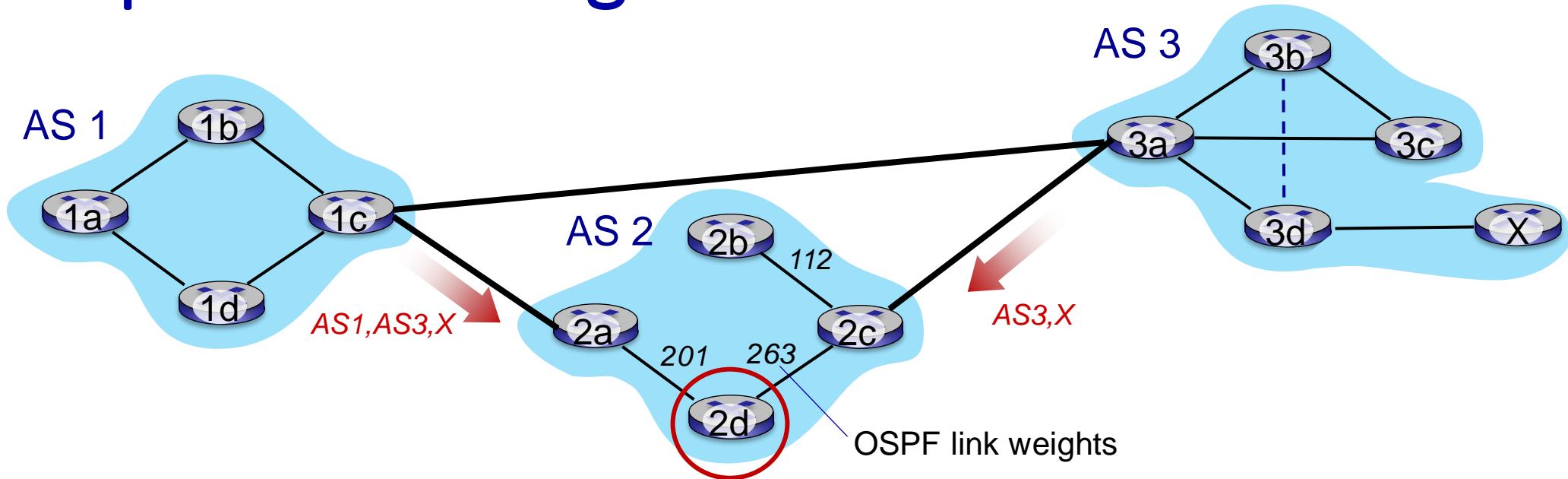
scale:

- hierarchical routing saves table size, reduced update traffic

performance:

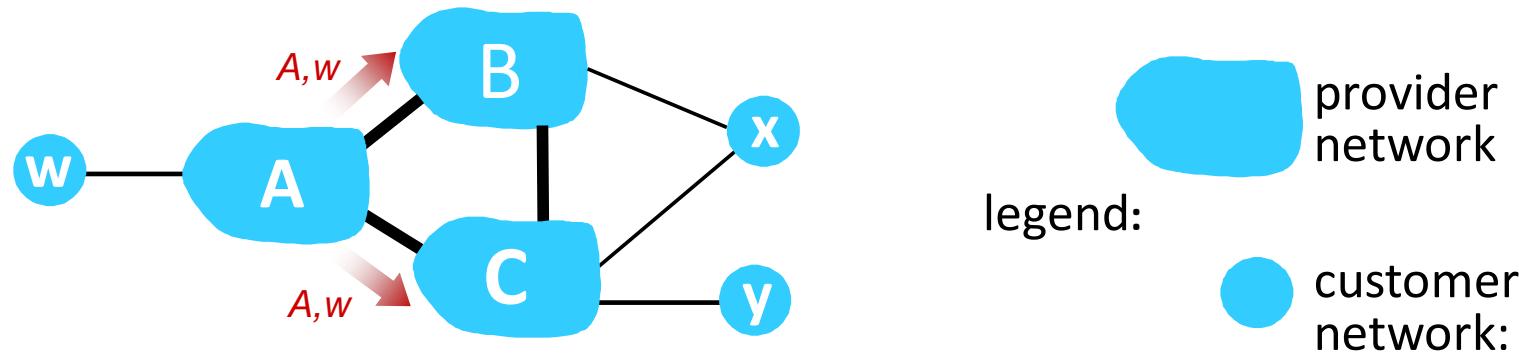
- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

# 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!

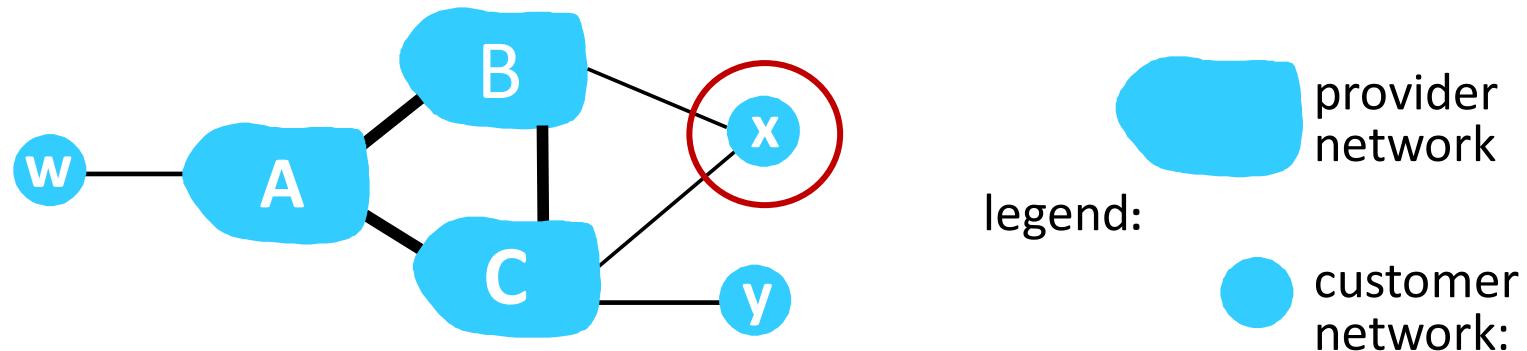
# BGP: achieving policy via advertisements



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

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

# BGP: achieving policy via advertisements (more)



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

- A,B,C are **provider networks**
- x,w,y are **customer** (of provider networks)
- x is **dual-homed**: attached to two networks
- **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

# BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- **SDN control plane**
- Internet Control Message Protocol



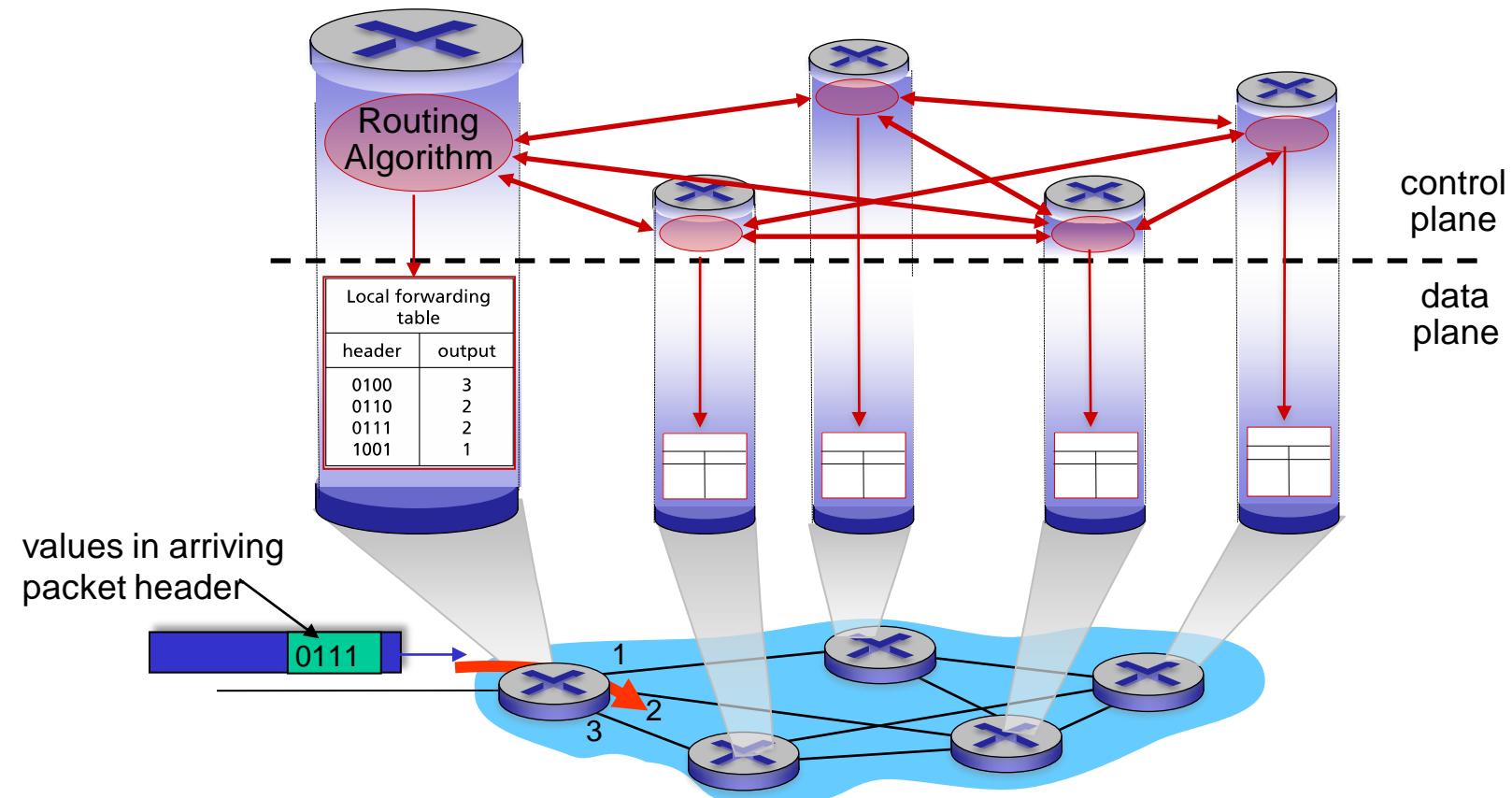
- network management, configuration
  - SNMP
  - NETCONF/YANG

# 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

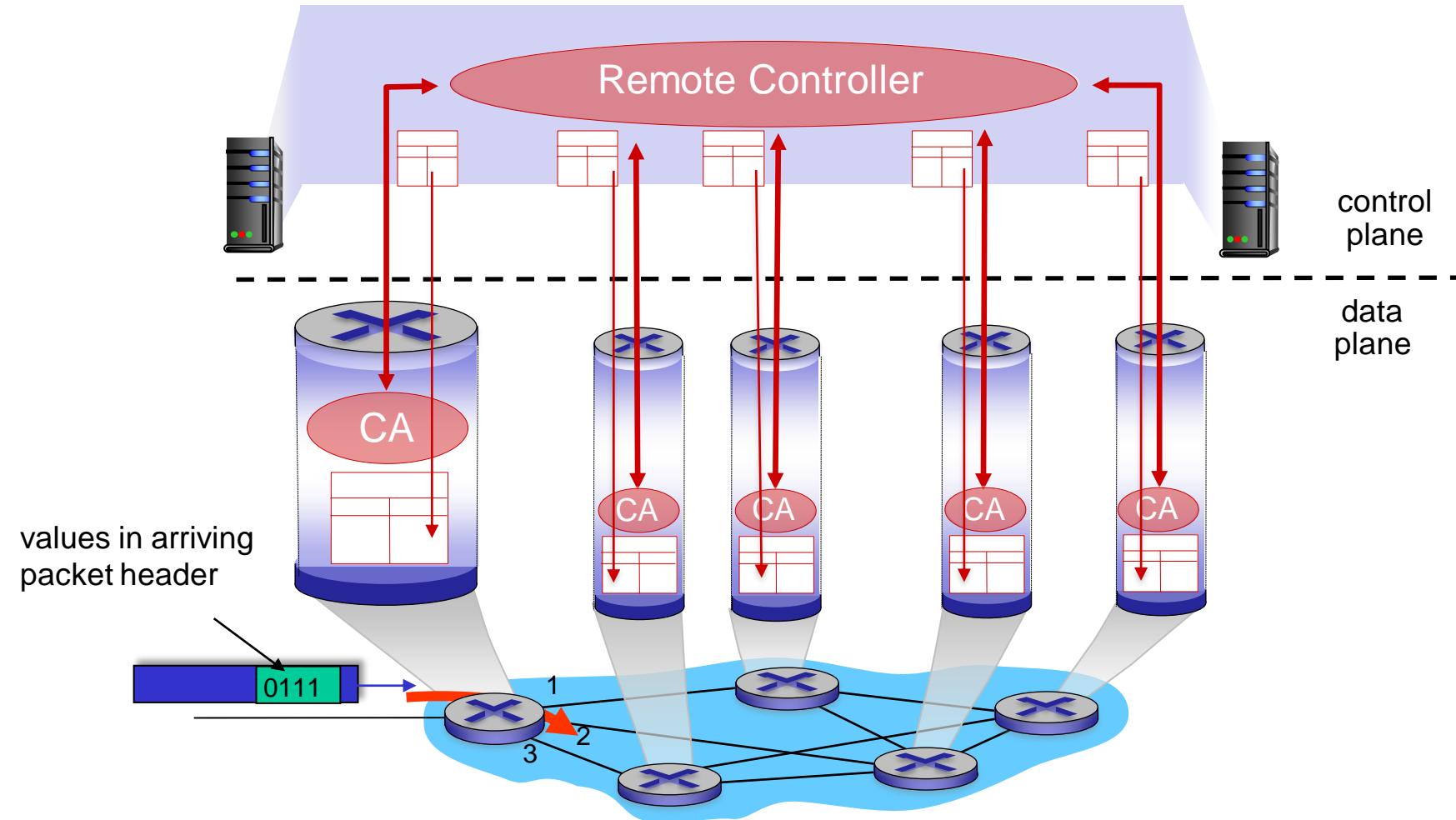
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane to computer forwarding tables



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Software defined networking (SDN)

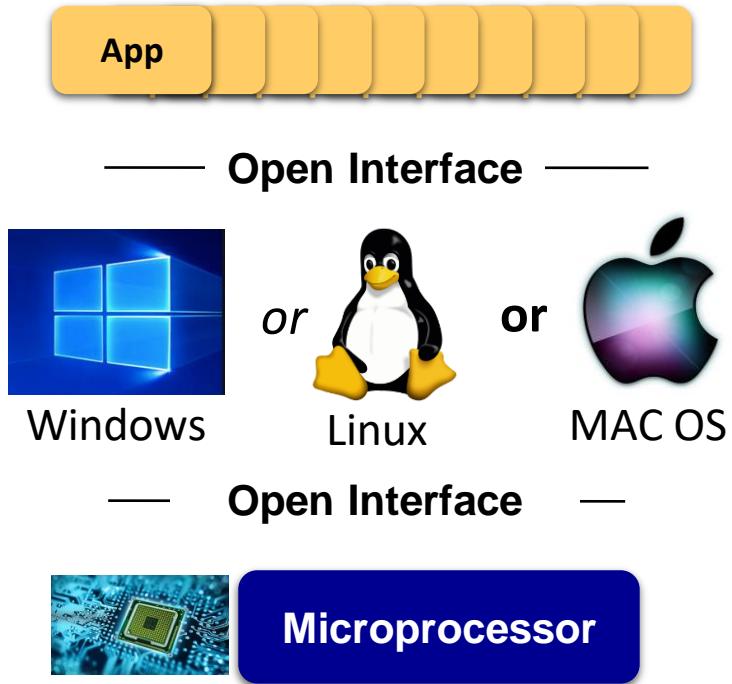
*Why* a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

# SDN analogy: mainframe to PC revolution

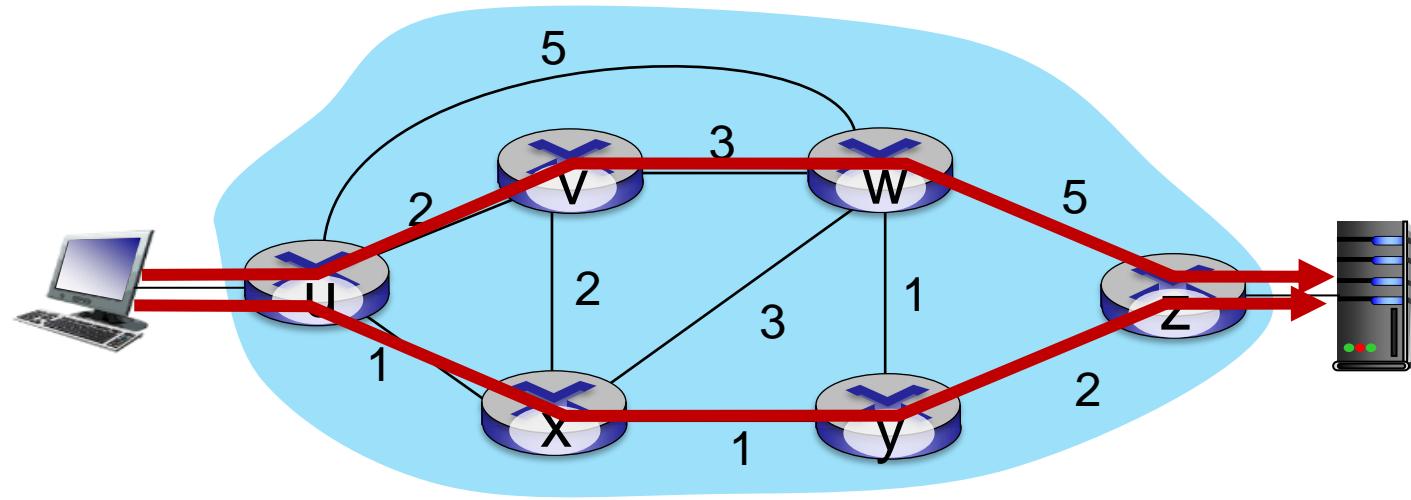


Vertically integrated  
Closed, proprietary  
Slow innovation  
Small industry



Horizontal  
Open interfaces  
Rapid innovation  
Huge industry

# Traffic engineering: difficult with traditional routing

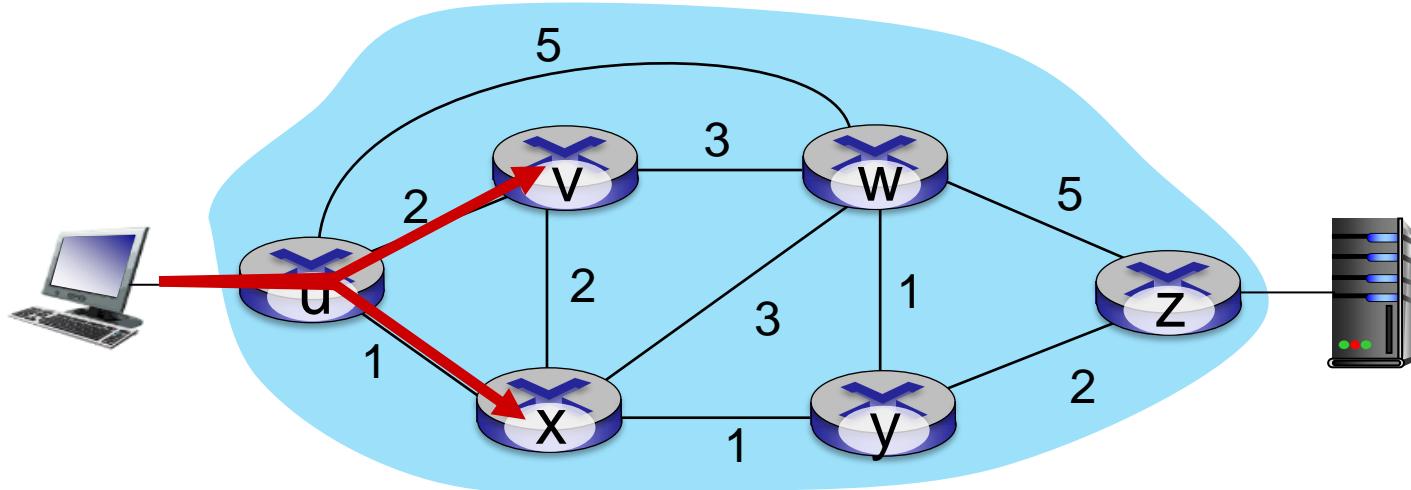


Q: what if network operator wants u-to-z traffic to flow along  $uvwz$ , rather than  $uxyz$ ?

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!*

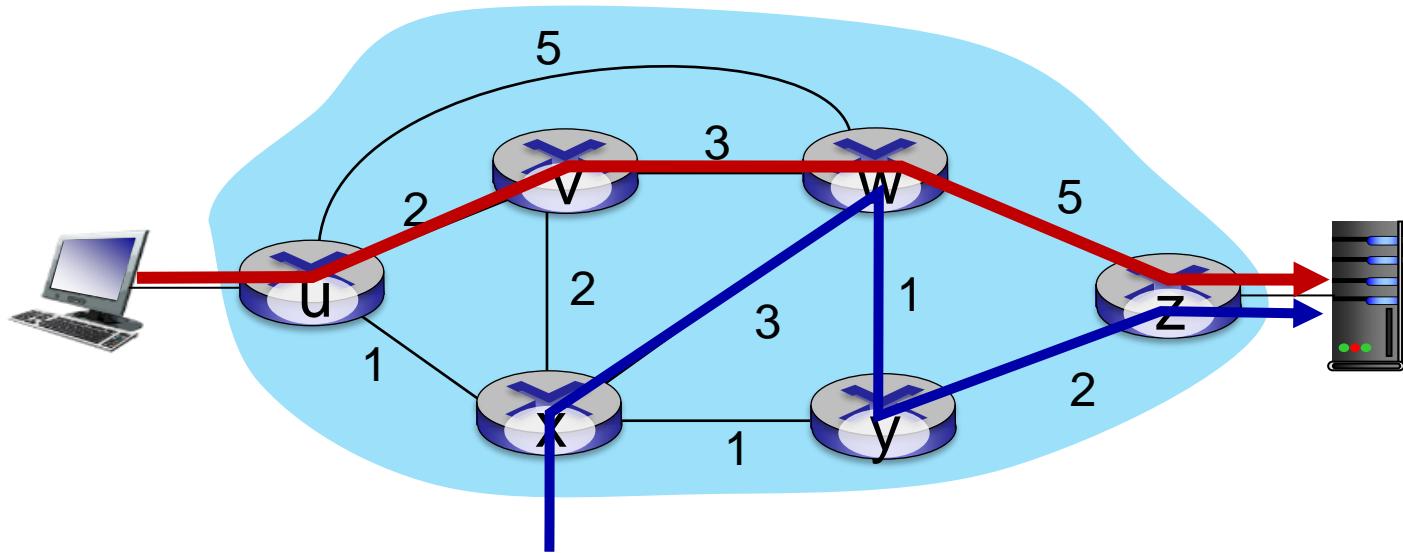
# Traffic engineering: difficult with traditional routing



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)

# 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)

We learned in Chapter 4 that generalized forwarding and SDN can be used to achieve *any* routing desired

# Software defined networking (SDN)

4. programmable  
control  
applications

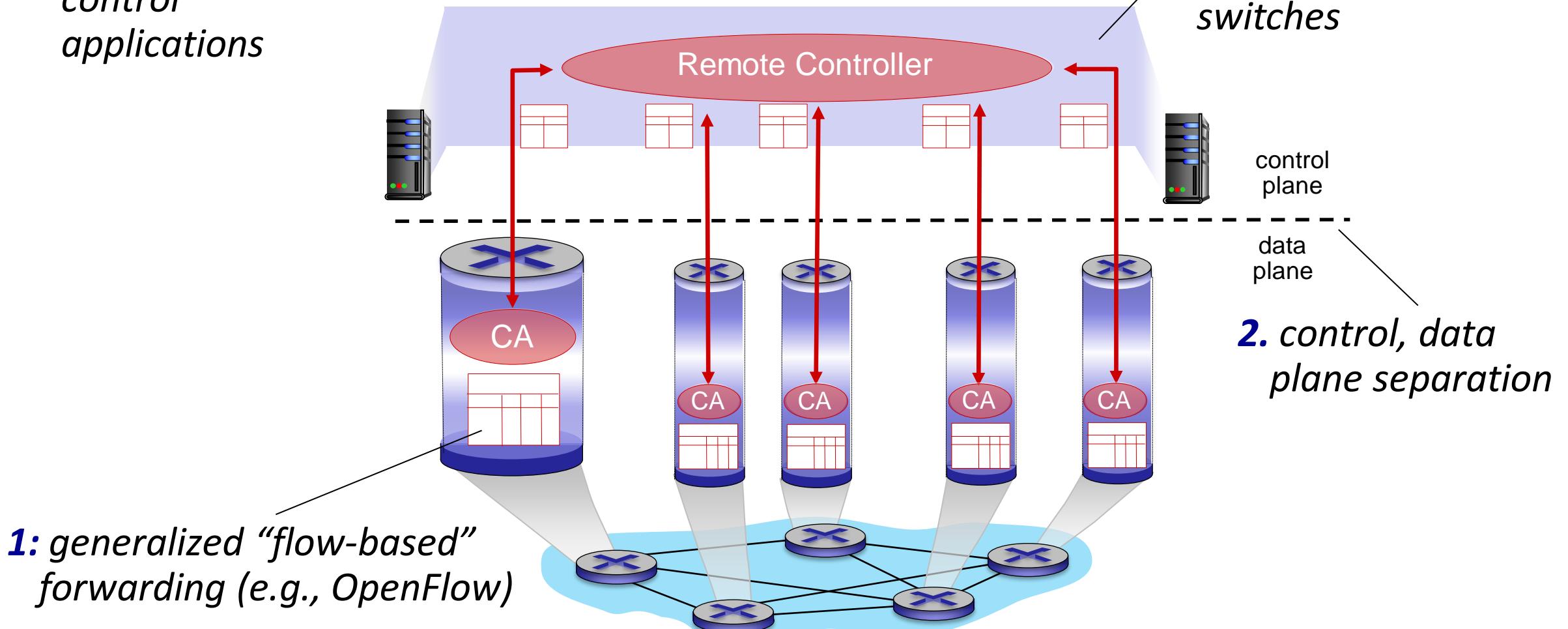
routing

access  
control

...

load  
balance

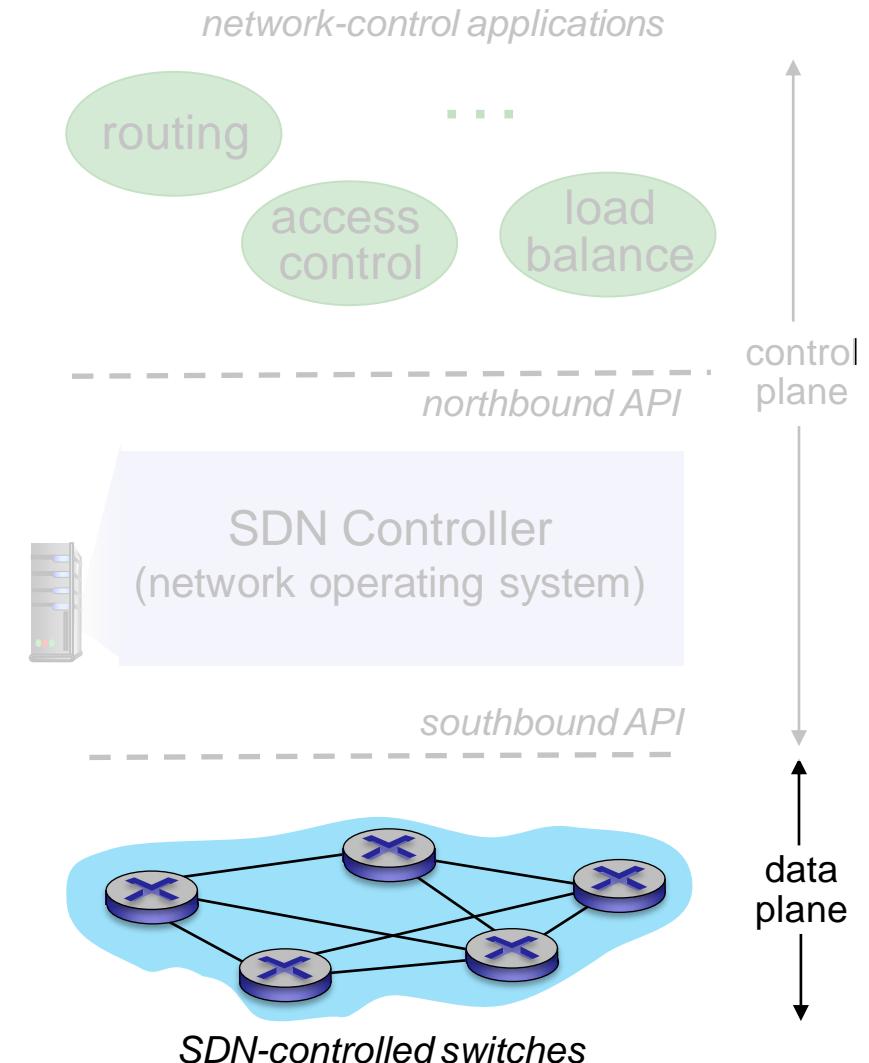
3. control plane functions  
external to data-plane  
switches



# Software defined networking (SDN)

## Data-plane switches:

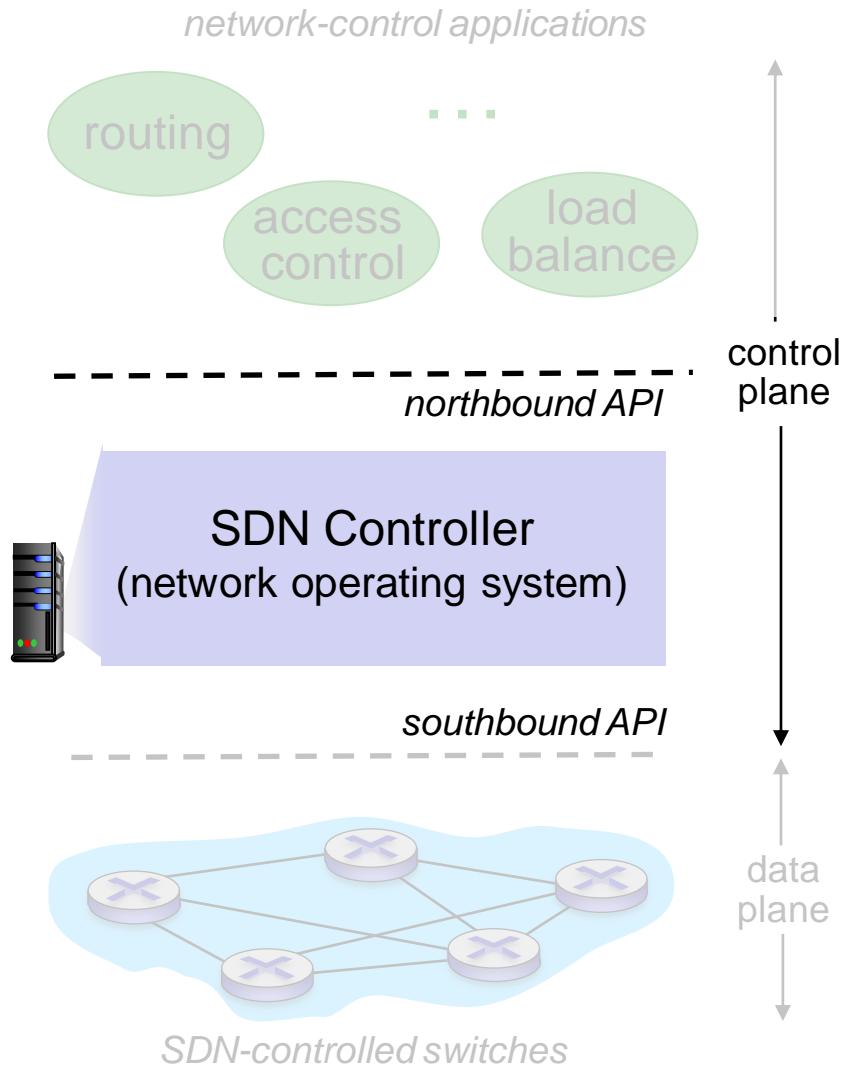
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating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



# Software defined networking (SDN)

## SDN controller (network OS):

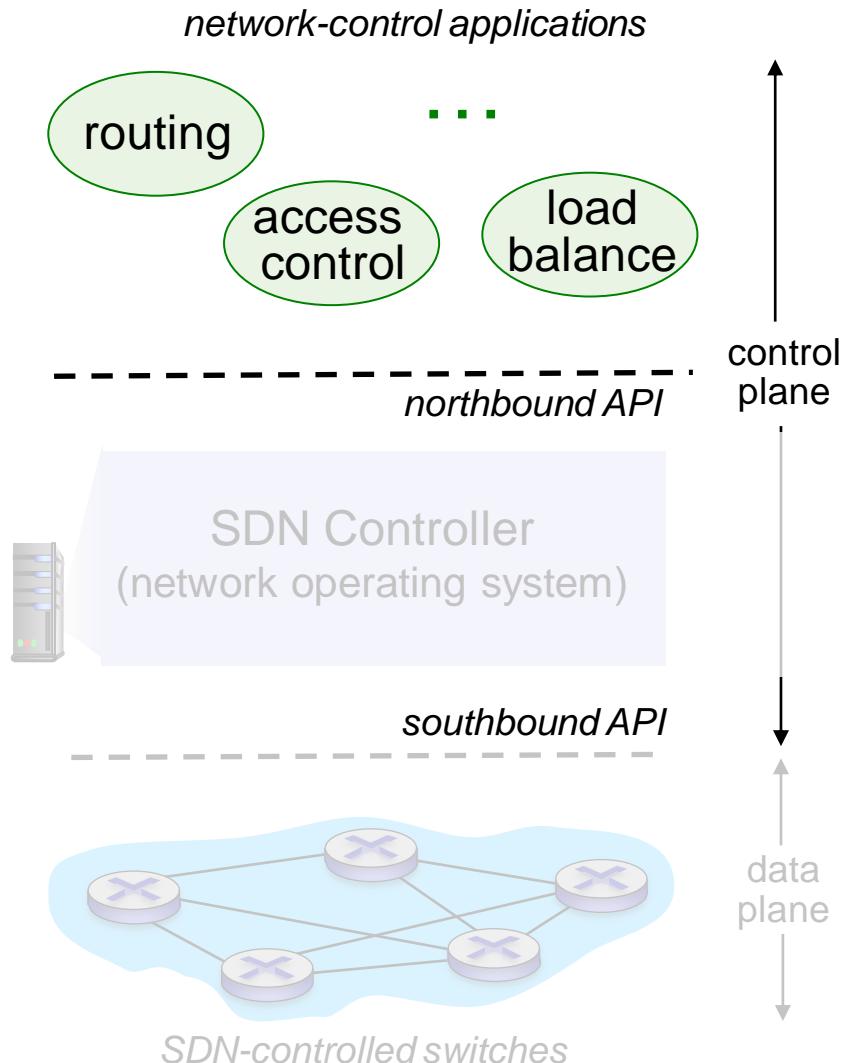
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# 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  
3<sup>rd</sup> party: distinct from routing  
vendor, or SDN controller

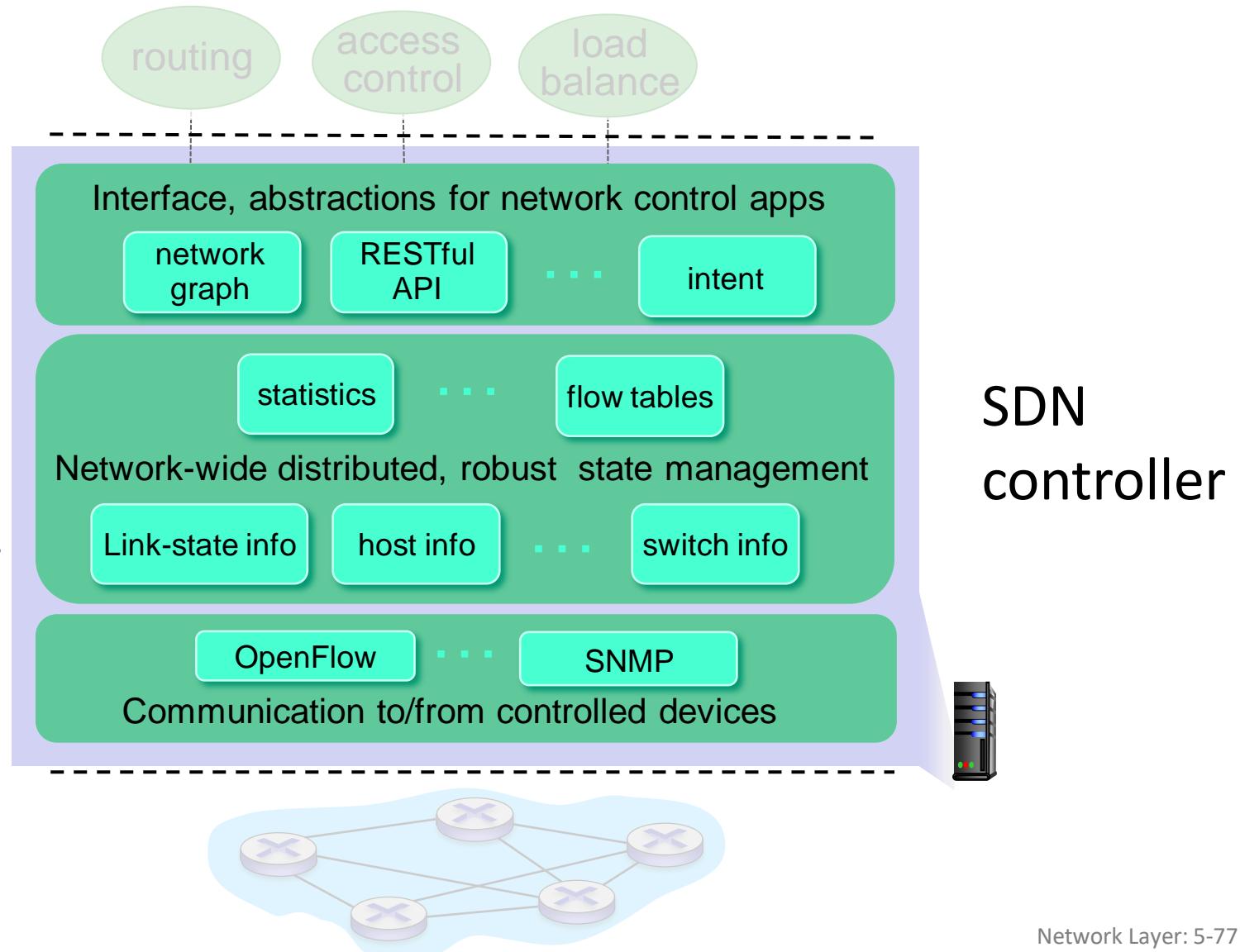


# Components of SDN controller

interface layer to network control apps: abstractions API

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

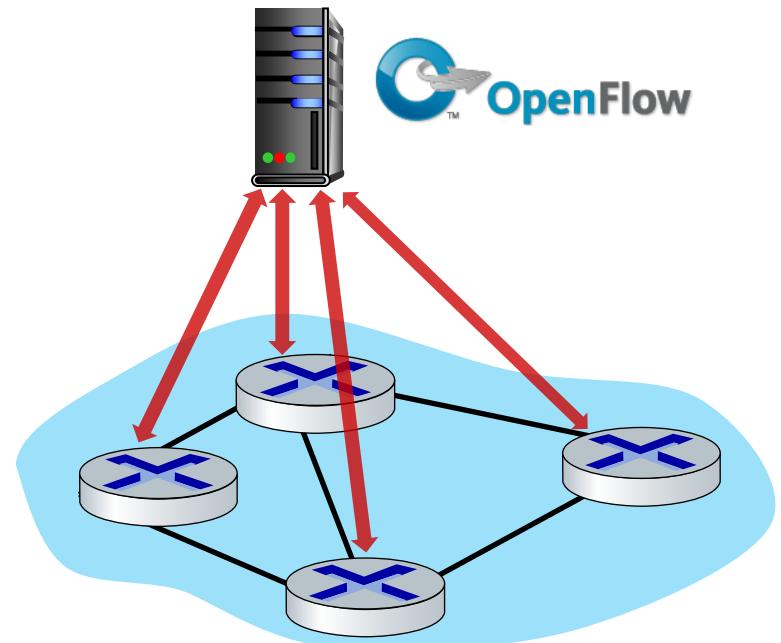
**communication**: communicate between SDN controller and controlled switches



# OpenFlow protocol

- operates between controller, 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

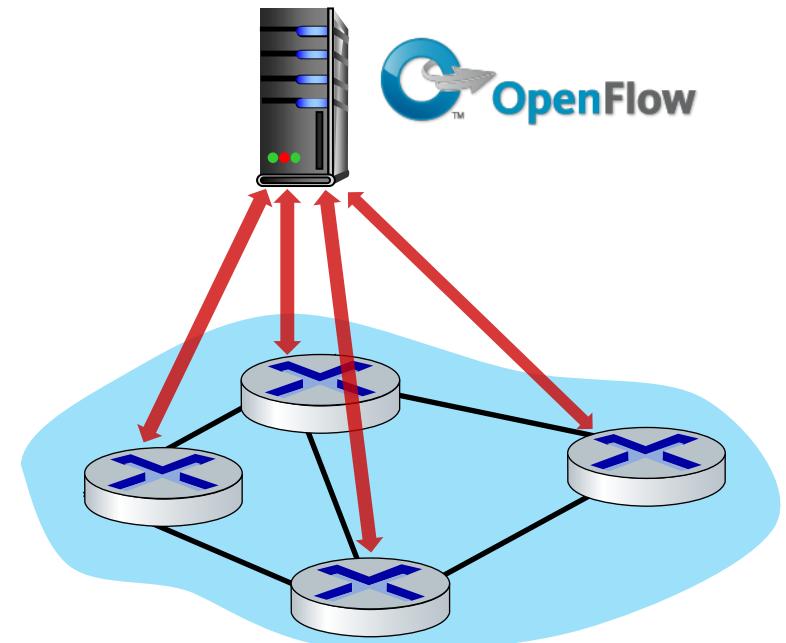


# OpenFlow: controller-to-switch messages

## 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

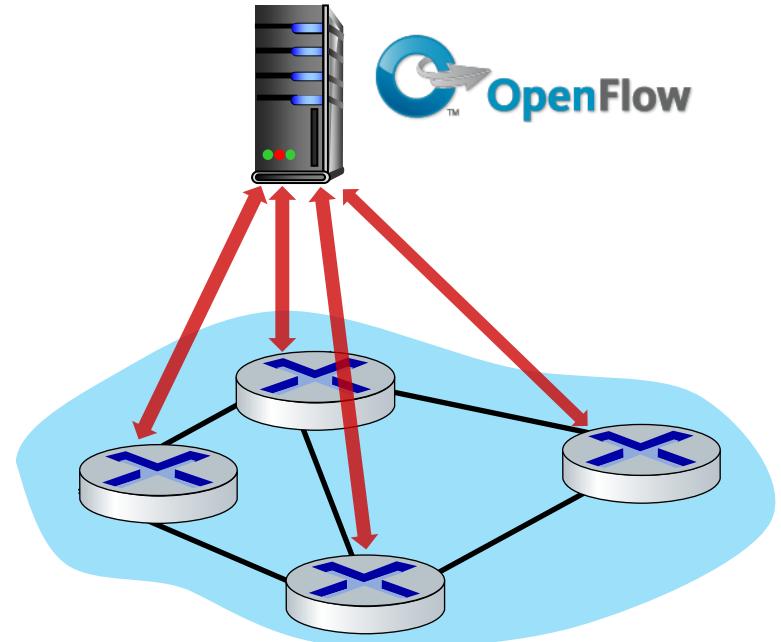


# OpenFlow: switch-to-controller messages

## Key switch-to-controller messages

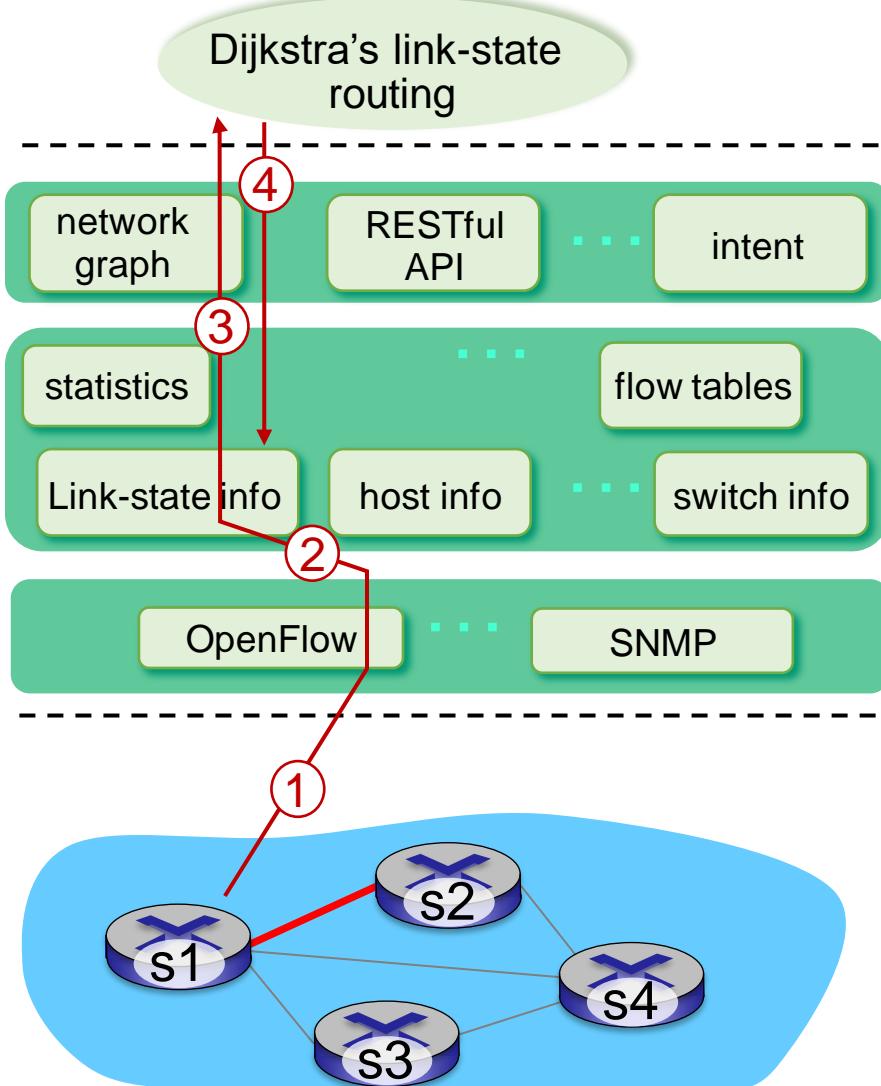
- *packet-in*: transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed*: flow table entry deleted at switch
- *port status*: inform controller of a change on a port.

## OpenFlow Controller



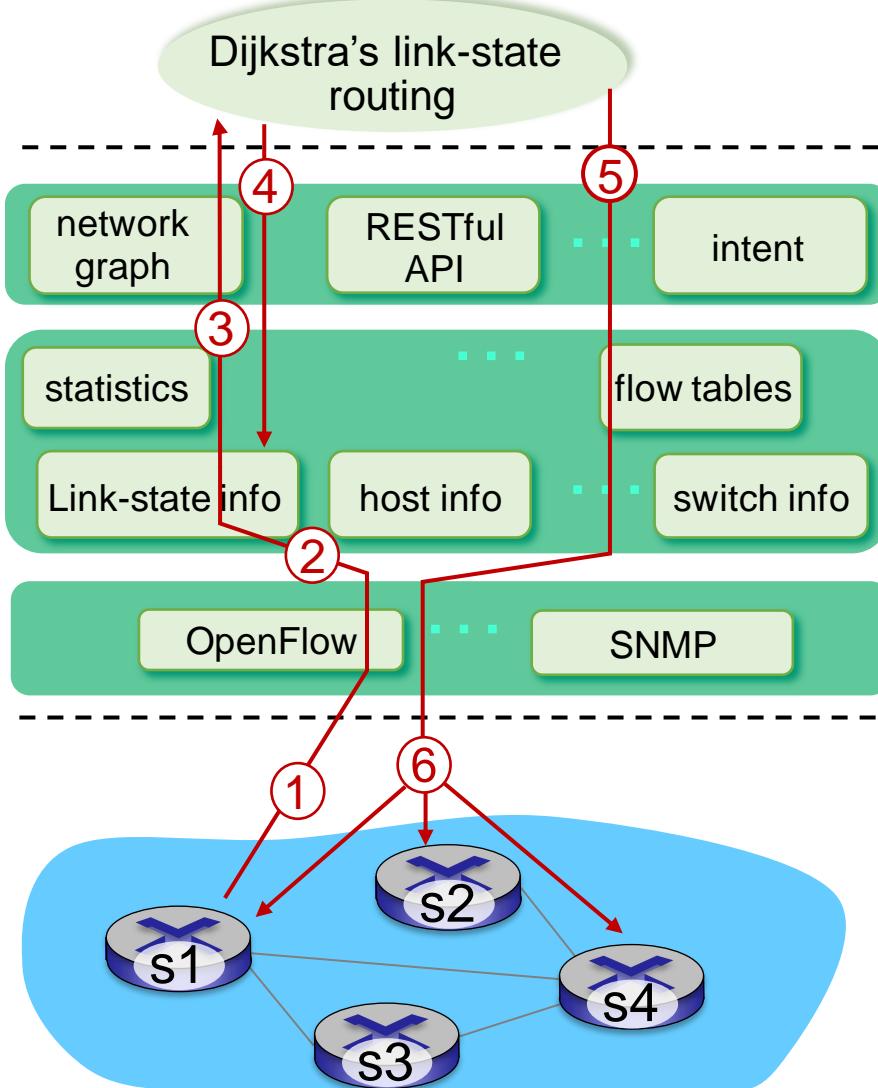
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN: control/data plane interaction example



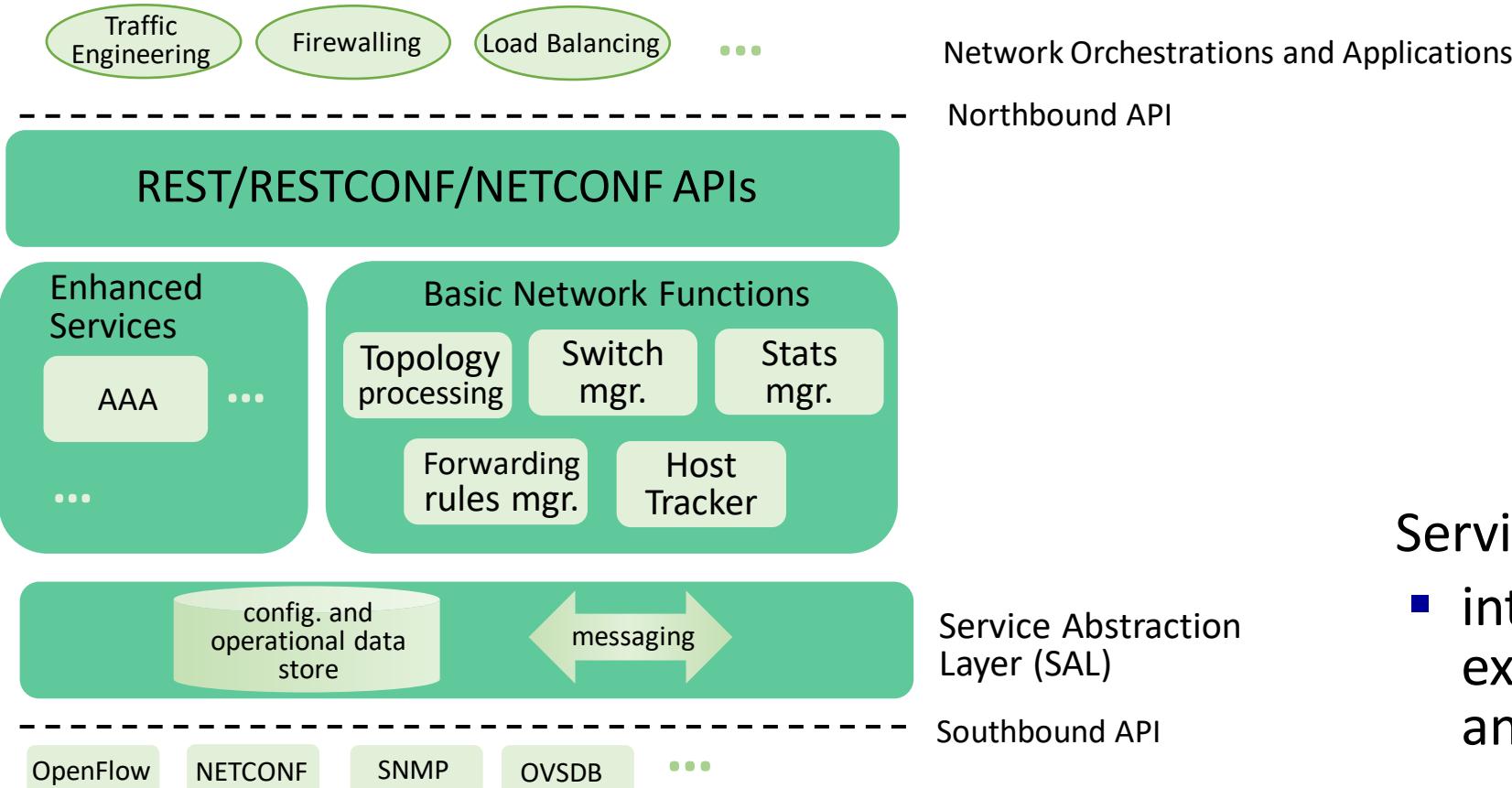
- ① S1, experiencing link failure uses OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's 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 tables in switches that need updating

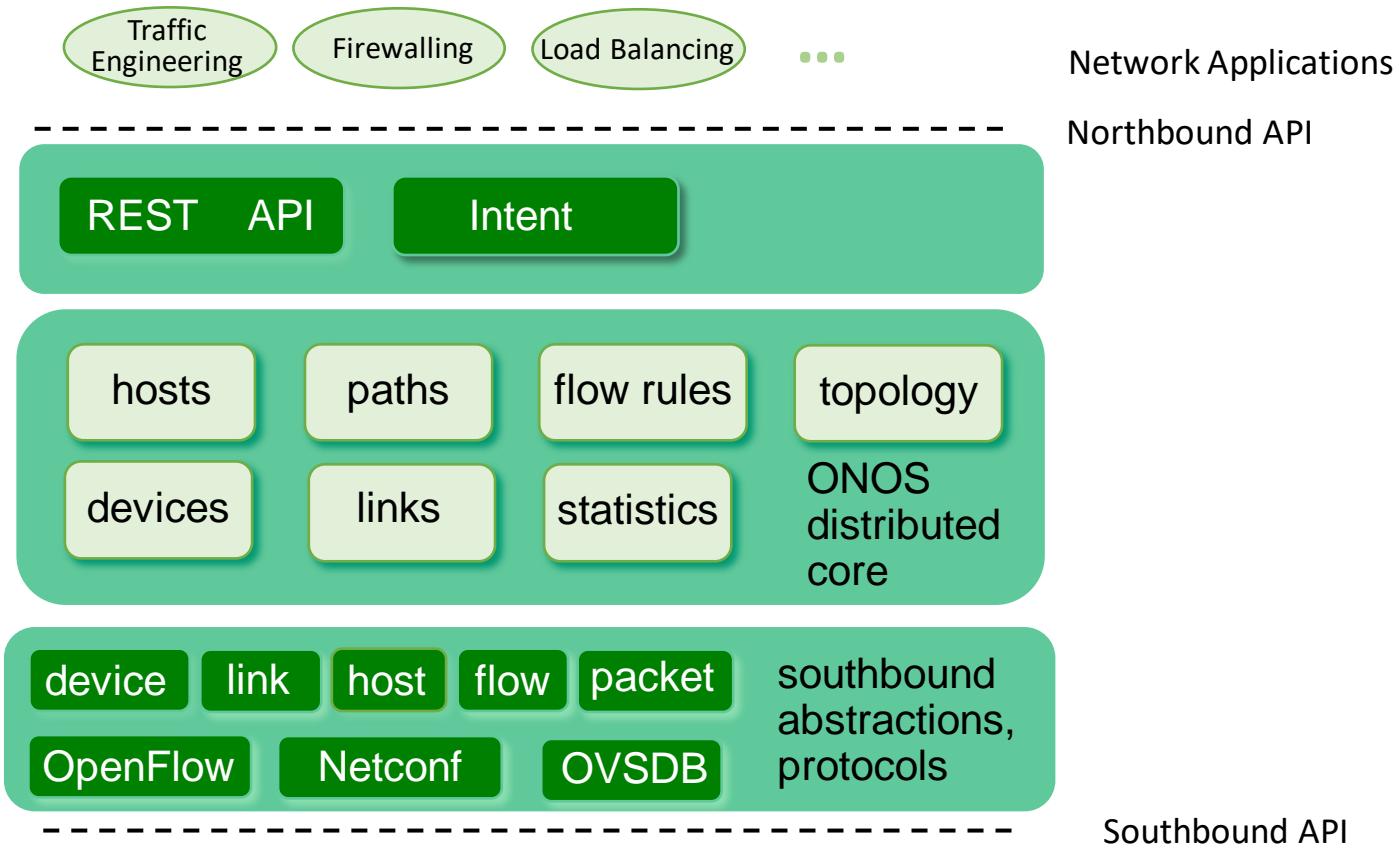
# OpenDaylight (ODL) controller



## 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, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling: beyond a single AS
- SDN critical in 5G cellular networks

# SDN and the future of traditional network protocols

- SDN-computed versus router-computer forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
  - controller sets sender rates based on router-reported (to controller) congestion levels



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



# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- **Internet Control Message Protocol**



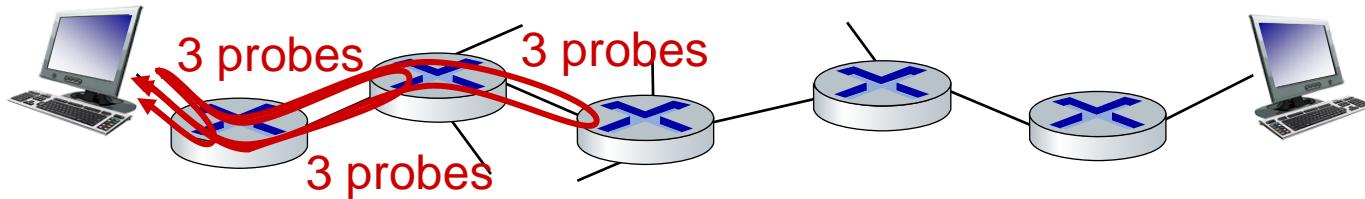
- network management, configuration
  - SNMP
  - NETCONF/YANG

# ICMP: internet control message protocol

- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP messages carried in IP datagrams
- *ICMP message*: type, code plus first 8 bytes of IP datagram causing error

| Type | Code | description                                   |
|------|------|-----------------------------------------------|
| 0    | 0    | echo reply (ping)                             |
| 3    | 0    | dest. network unreachable                     |
| 3    | 1    | dest host unreachable                         |
| 3    | 2    | dest protocol unreachable                     |
| 3    | 3    | dest port unreachable                         |
| 3    | 6    | dest network unknown                          |
| 3    | 7    | dest host unknown                             |
| 4    | 0    | source quench (congestion control - not used) |
| 8    | 0    | echo request (ping)                           |
| 9    | 0    | route advertisement                           |
| 10   | 0    | router discovery                              |
| 11   | 0    | TTL expired                                   |
| 12   | 0    | bad IP header                                 |

# Traceroute and ICMP



- source sends sets of UDP segments to destination
  - 1<sup>st</sup> set has TTL =1, 2<sup>nd</sup> set has TTL=2, etc.
- datagram in  $n$ th set arrives to  $n$ th router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

## stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP “port unreachable” message (type 3, code 3)
- source stops

# Network layer: Summary

we've learned a lot!

- approaches to network control plane
  - per-router control (traditional)
  - logically centralized control (software defined networking)
- traditional routing algorithms
  - implementation in Internet: OSPF , BGP
- SDN controllers
  - implementation in practice: ODL, ONOS
- Internet Control Message Protocol
- network management

*next stop: link layer!*