Chapter 5
Network Layer: Control Plane

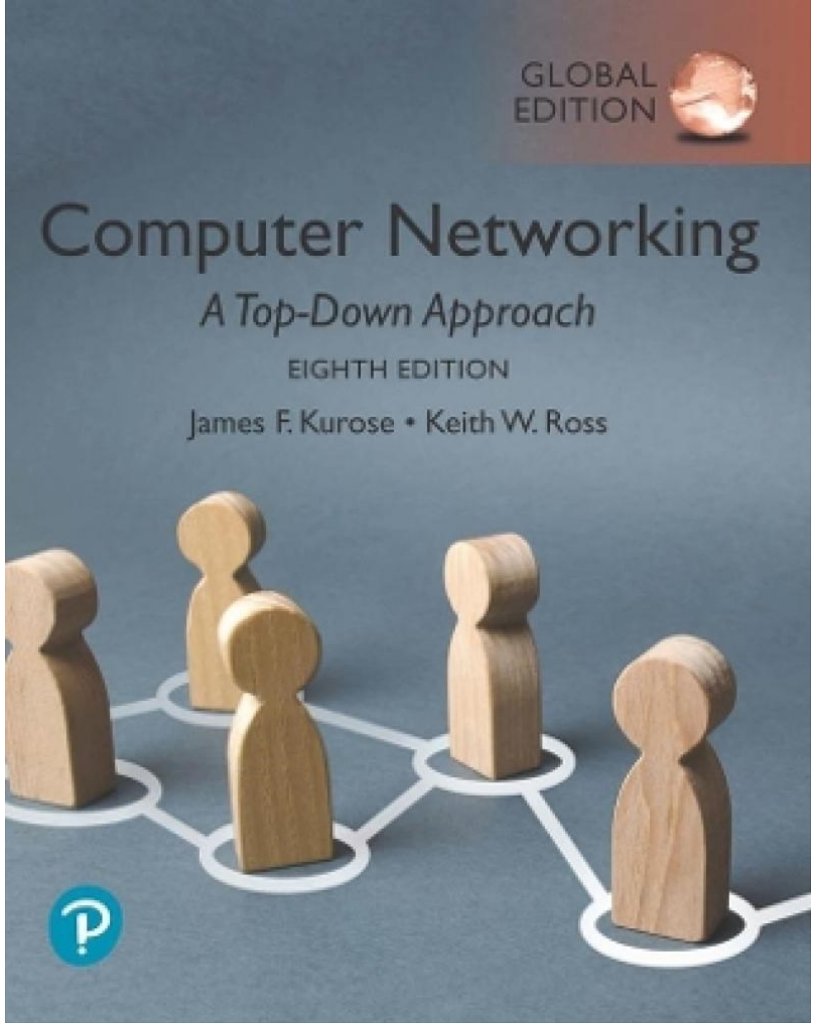
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks, and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Network layer control plane: our goals

- understand principles behind network control plane:
  - traditional routing algorithms
  - SDN controllers

- instantiation, implementation in the Internet:
  - OSPF, BGP
  - OpenFlow, ODL and ONOS controllers
  - Internet Control Message Protocol: ICMP

5 Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

- 5.2.1 Link state
- 5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Network-layer functions

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

data plane

- Routing: determine route taken by packets from source to destination

control plane

Two approaches to routing:

- Distributed routing (traditional)
- Logically centralized routing (software defined networking)

5 Network layer: “control plane” roadmap

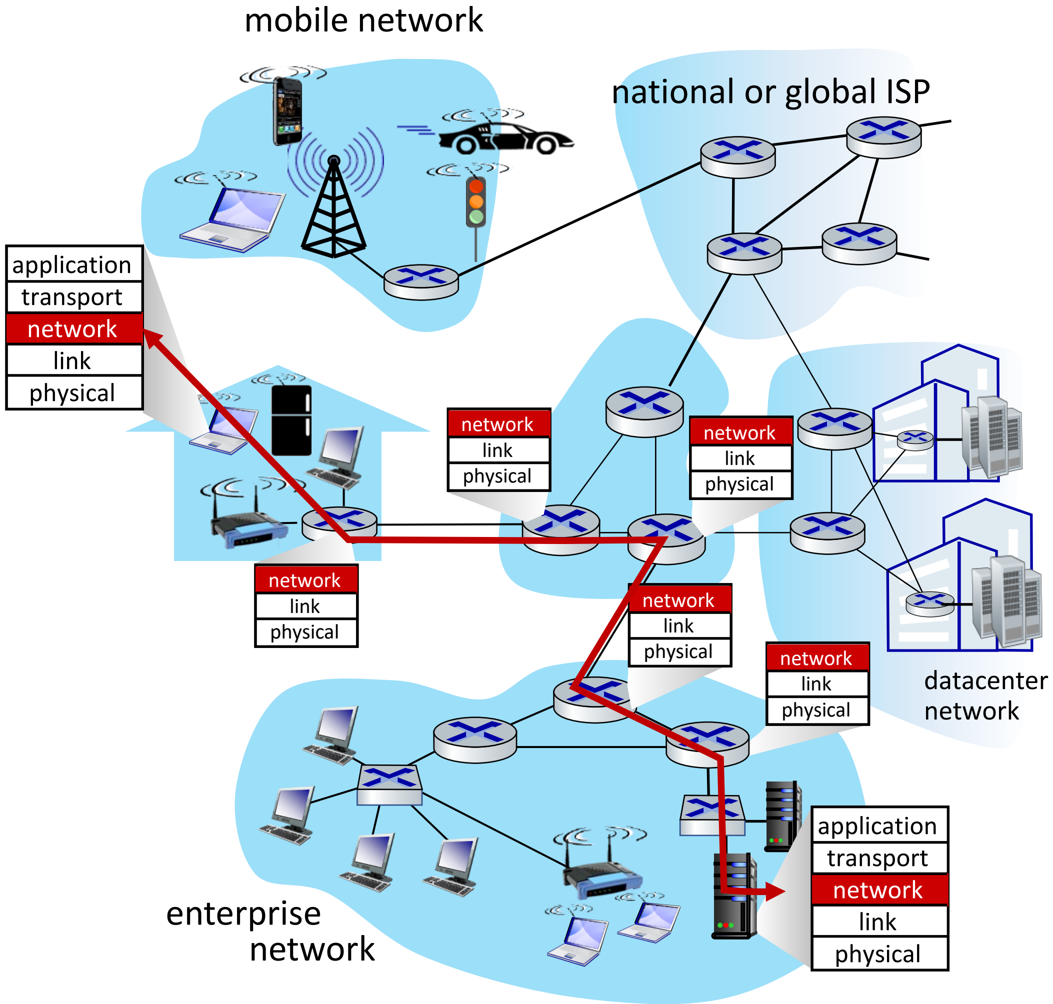
- 5.1 Introduction
- 5.2 Routing algorithms
  - 5.2.1 Link state
  - 5.2.2 Distance vector
- 5.3 Intra-ISP routing: OSPF
- 5.4 routing among ISPs: BGP
- 5.5 SDN control plane
- 5.6 Internet Control Message Protocol

Routing algorithms

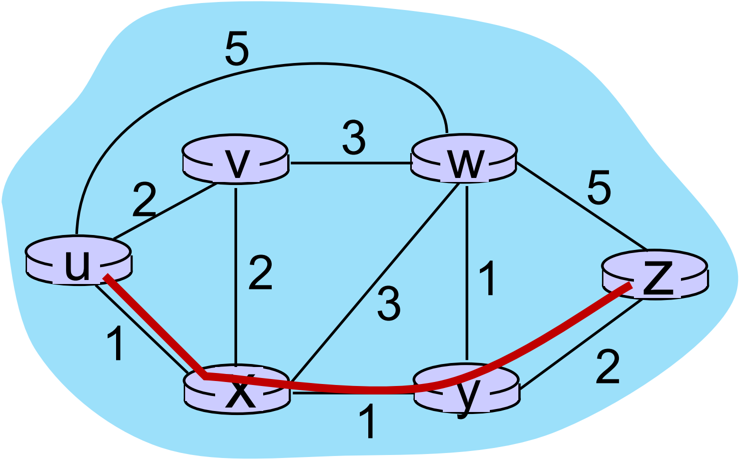
compute routing tables

Routing goal: determine "good" paths 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"



Graph abstraction: link costs



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

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

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

graph: *G = (N,E)*

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

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

Routing algorithm classification

“Global” algorithms: complete network topology

all routers have *complete* topology, link cost info

- “link state” algorithms

“Distributed” algorithms: partial network topology

iterative process of computation, exchange of info with neighbors

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

global or decentralized information?

5 Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

- 5.2.1 Link state
- 5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Dijkstra's link-state routing algorithm

- centralized: global network topology, link costs known to *all* nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- Each node/router runs this algorithm
  - computes least cost paths from one node ("source") to all other nodes
  - gives *routing table* for that router
- iterative: after *k* iterations, know least cost path to *k* destinations

notation

- $c_{x,y}$: *direct* distance (cost) from node *x* to *y*; initially ∞ if not direct neighbors
- *d(v):* distance (cost) from source to a node *v*
- *p(v):* path vector - points to the adjacent node with the shortest distance to node v
- *N:* set of nodes

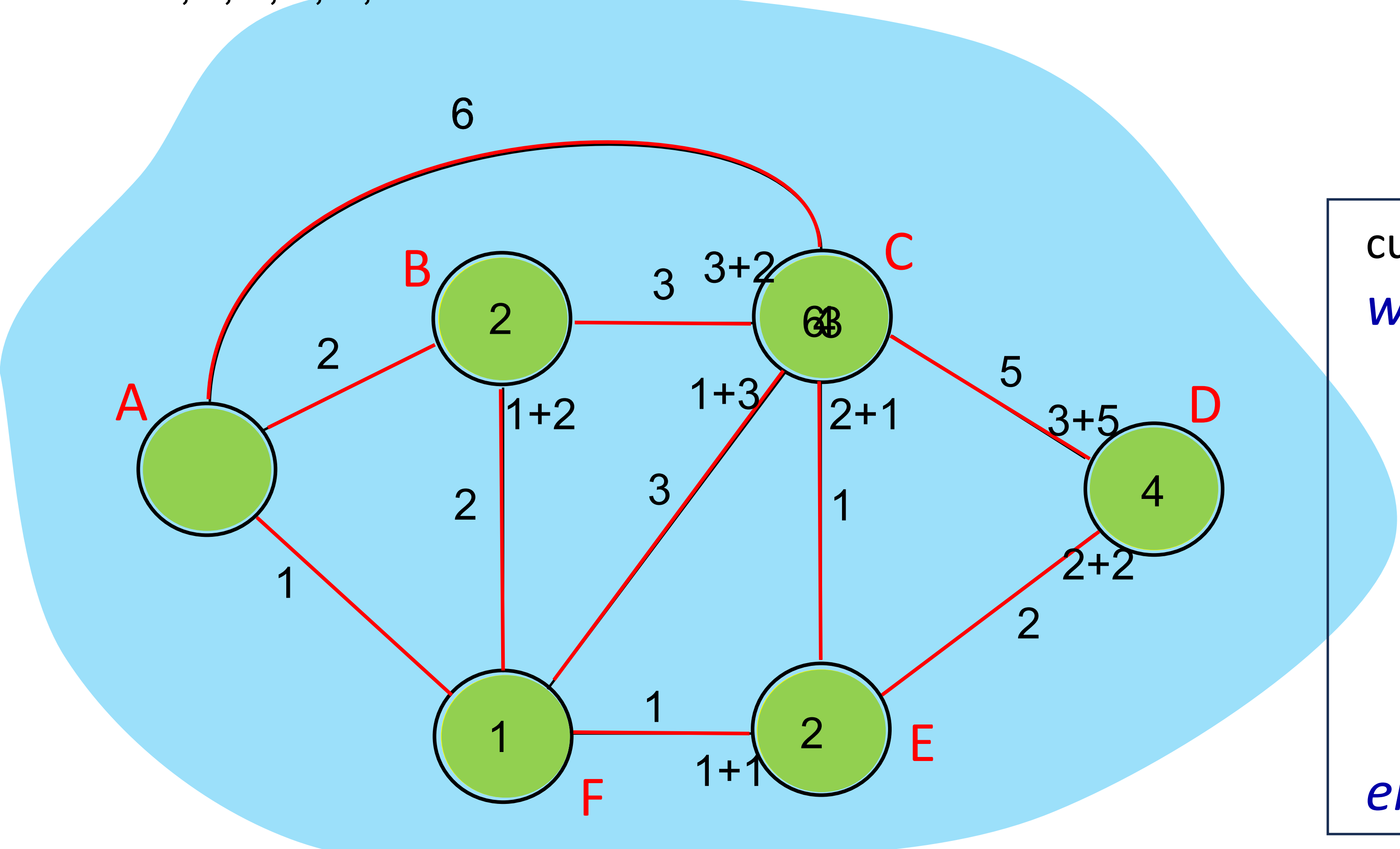
Dijkstra’s link-state routing algorithm

```
Initialization:
N = {u}                          /* compute least cost path from u to all other nodes */
for all nodes v
    if v adjacent to u           /* u initially knows distance/costs only to direct neighbors   */
        then d[v] = $c_{u,v}$
    else d[v] = ∞

current node u = source node; add u to N
while N is not complete
    for each neighbor v of u,  not in N:
        // shortest distance to v is unchanged or goes via u to v:
        d[v] = min ( d[v],  d[u] + $c_{u,v}$ )
        p[v] = p[u]
    next
    new current node u = unvisited node in N with shortest distance to current node u
    add u to N
```

Step	N	d(B),p(B)	d(C),p(C)	d(D),p(D)	d(E),p(E)	d(F),p(F)
0	A	2, A	6, A			1, A
1	A,F	2, A	4, F		2, F	
2	A,F,B		4, F			
3	A,F,B,E		3, E	4, E		
4	A,F,B,E,C			4, E		
5	A,F,B,E,C,D					

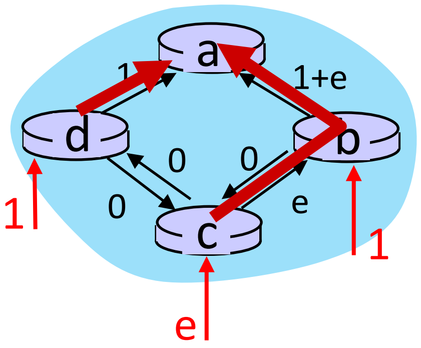
n	d(n)	p(n)
A	-	-
B	2	A
C	3	E
D	4	E
E	2	F
F	1	A



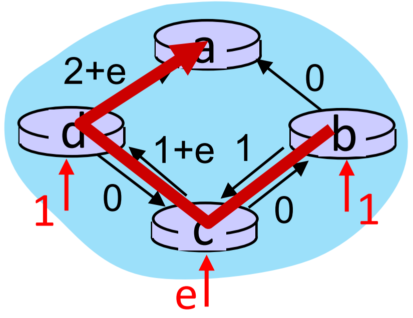
```
current node $u$ = source node, add $u$ to N
while N is not complete
    for each neighbor $v$ to $u$,  not in N
        $d[v]$ = min ( $d[v]$,  $d[u] + c_{u,v}$ )
        $p[v] = p[u]$
    next
    let $u$ = unvisited node in N with min.
                distance to current node $u$
    add $u$ to N
end while
```

Dijkstra’s algorithm: oscillations possible

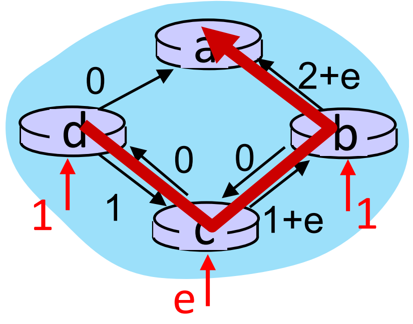
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



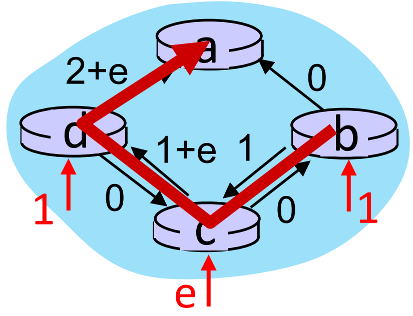
initially



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

5.1 Introduction

5.2 Routing algorithms

- 5.2.1 Link state
- 5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Distance vector algorithm

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

Bellman-Ford equation

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

Then:

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

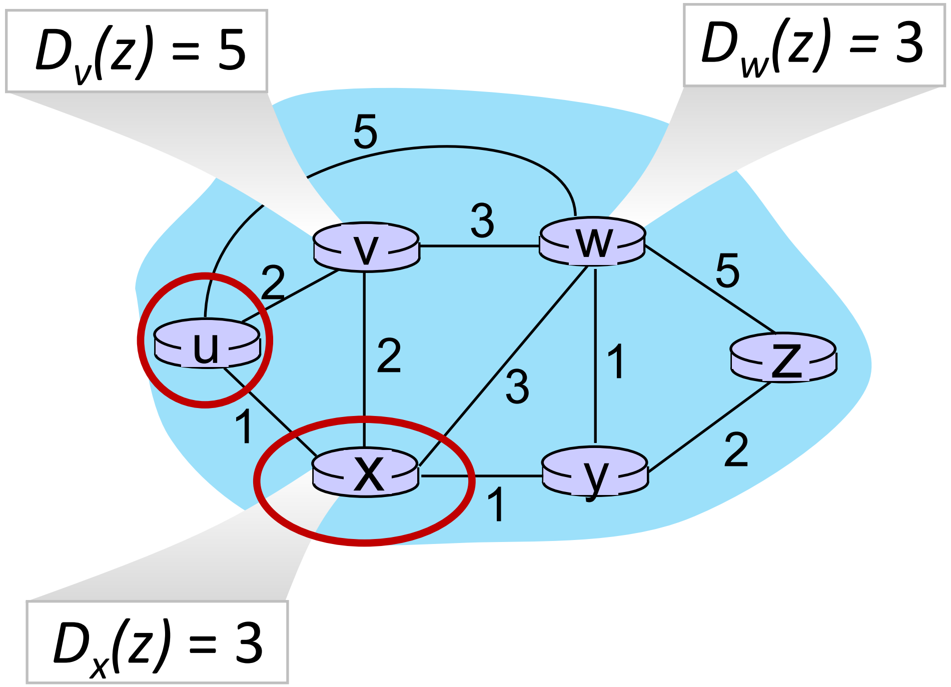
v's estimated least-cost-path cost to y

min taken over all neighbors v of x

direct cost of link from x to v

Bellman-Ford Example

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



$D_v(z) = 5$

$D_w(z) = 3$

$D_x(z) = 3$

Bellman-Ford equation says:

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

node x is next hop on least-cost path to destination z

Distance vector algorithm

key idea:

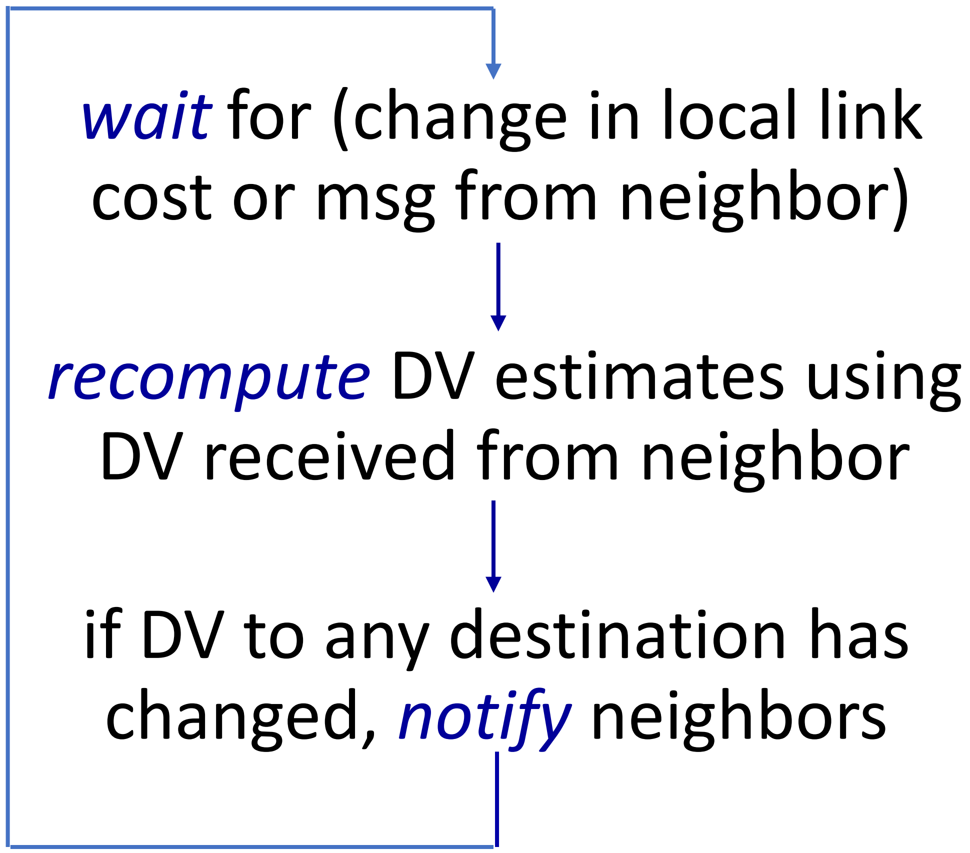
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

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

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

Distance vector algorithm:

each node:



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

recompute DV estimates using DV received from neighbor

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

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

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

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

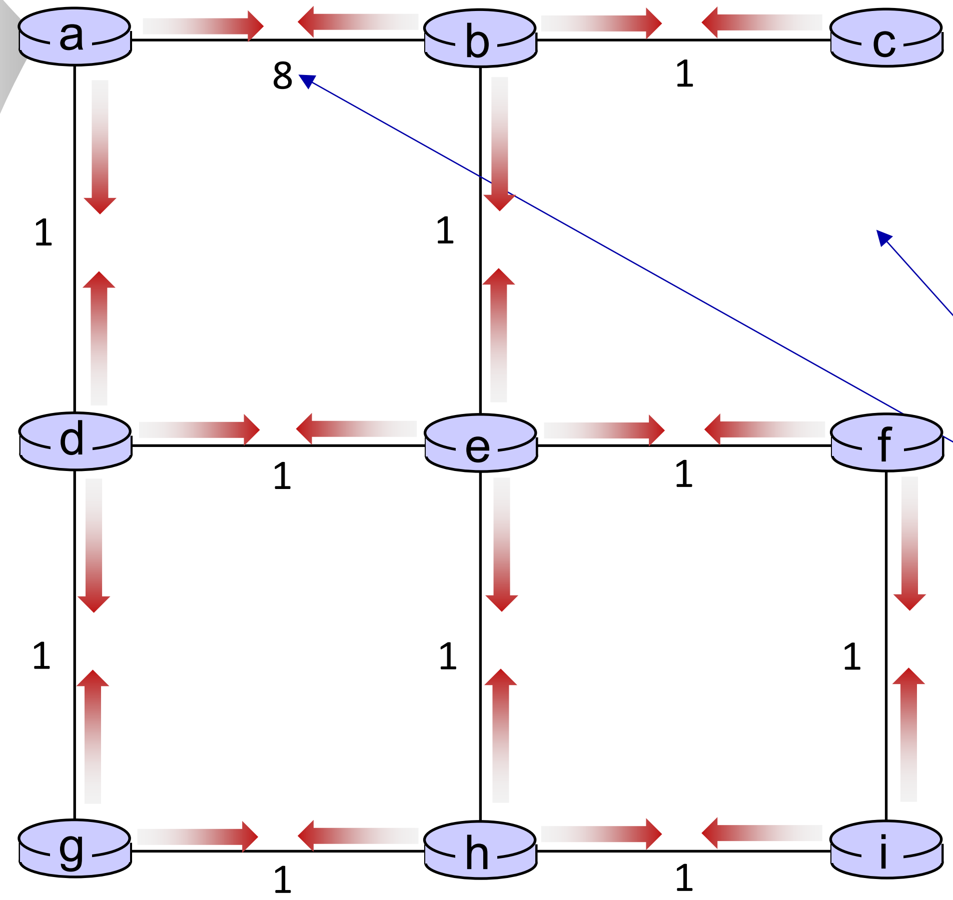
Distance vector: example

t=0

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

DV in a:

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



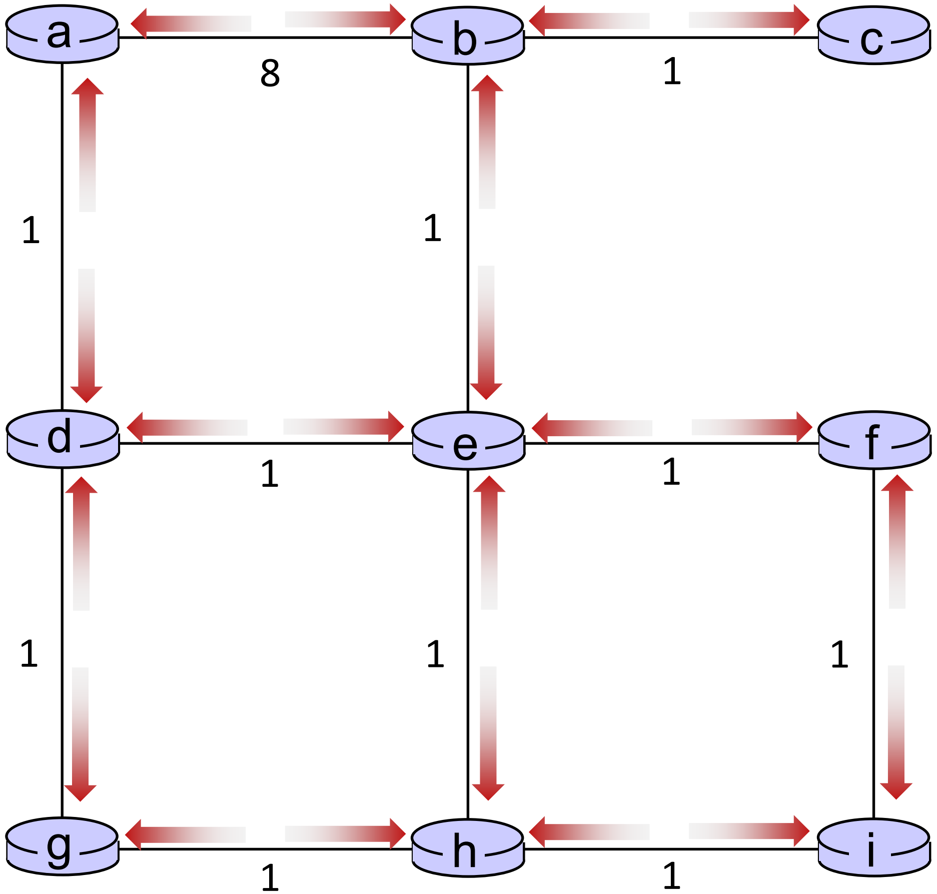
A few asymmetries:
- missing link
- larger cost

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

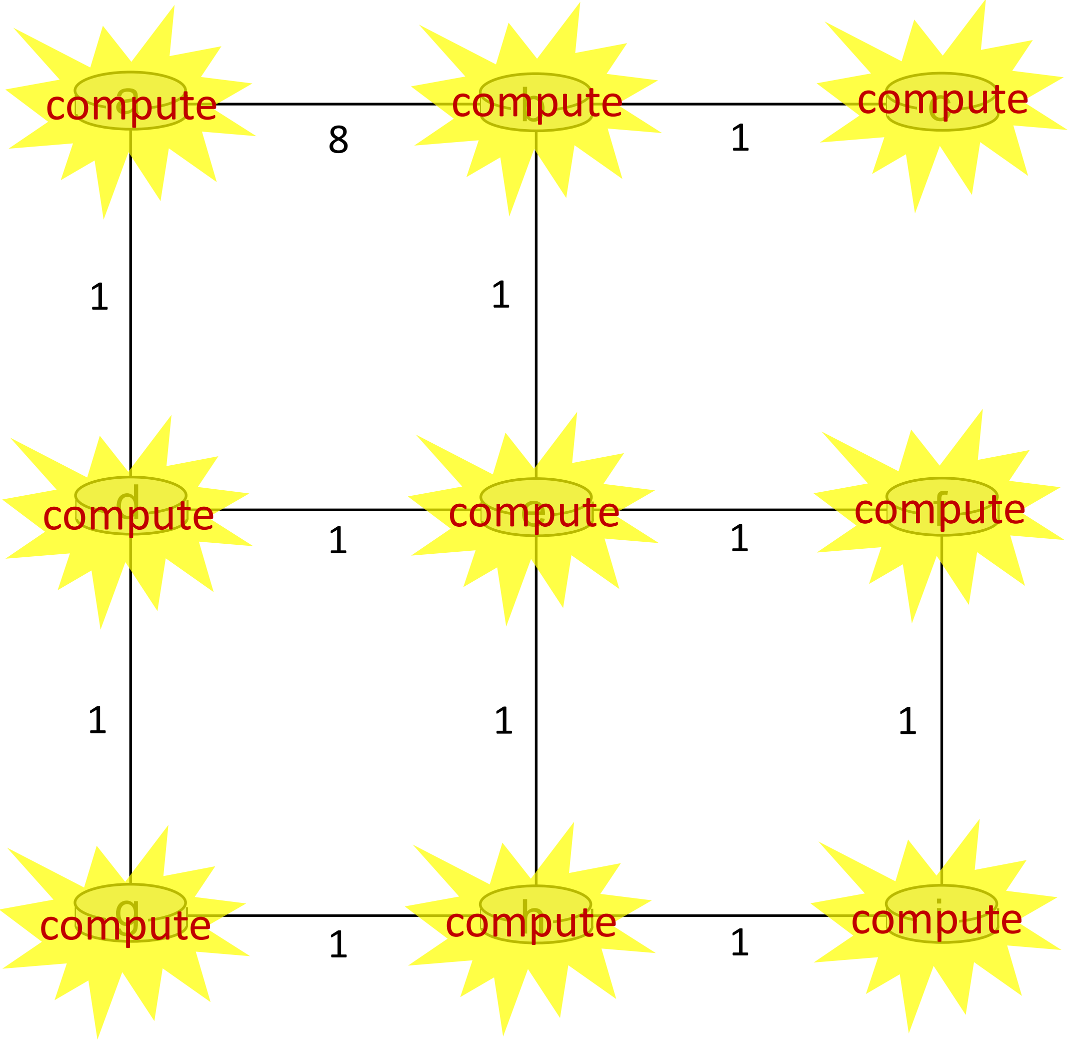


Distance vector example: iteration

t=1

All nodes:

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

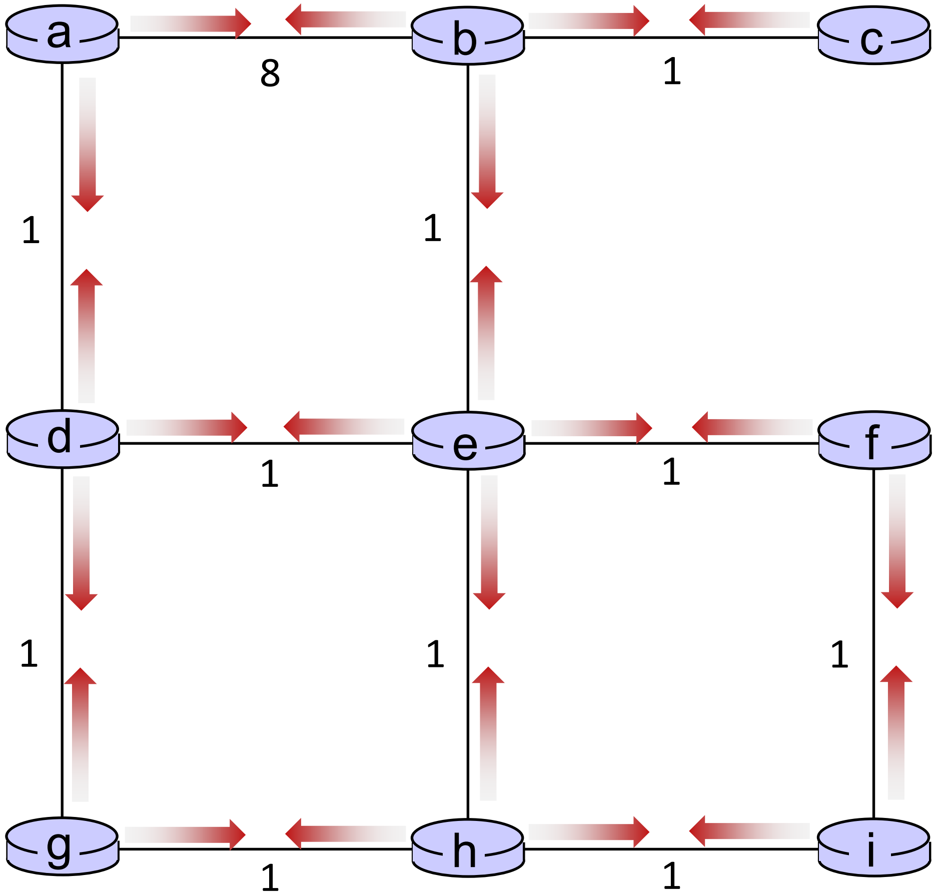


Distance vector example: iteration

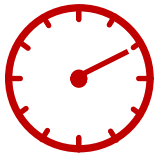
t=1

All nodes:

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



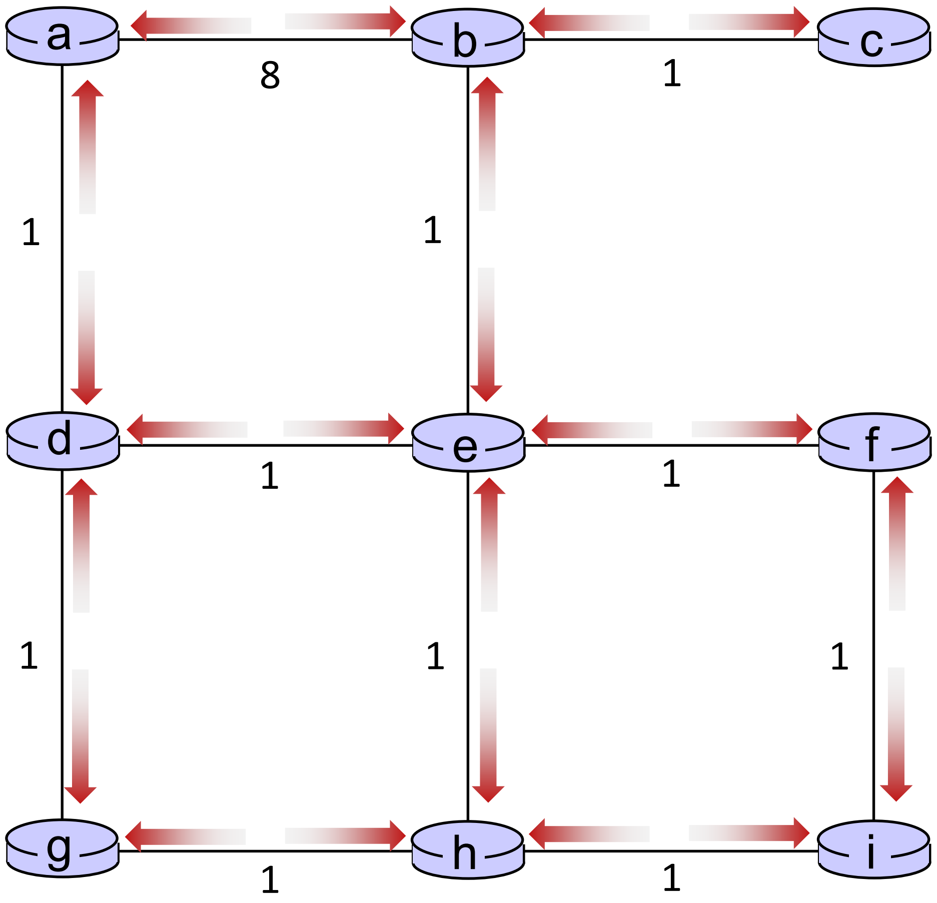
Distance vector example: iteration



t=2

All nodes:

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

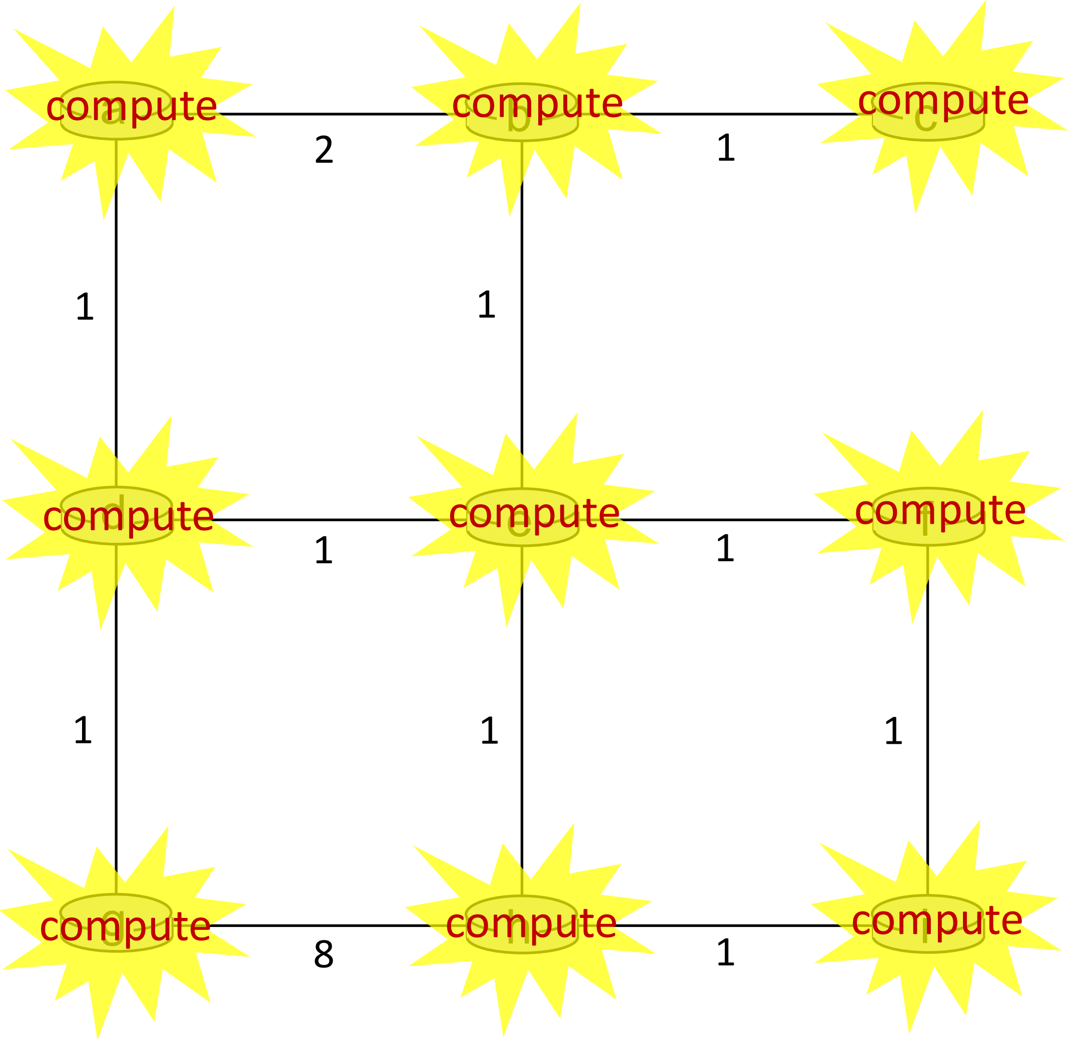


Distance vector example: iteration

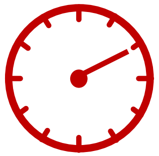
t=2

All nodes:

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



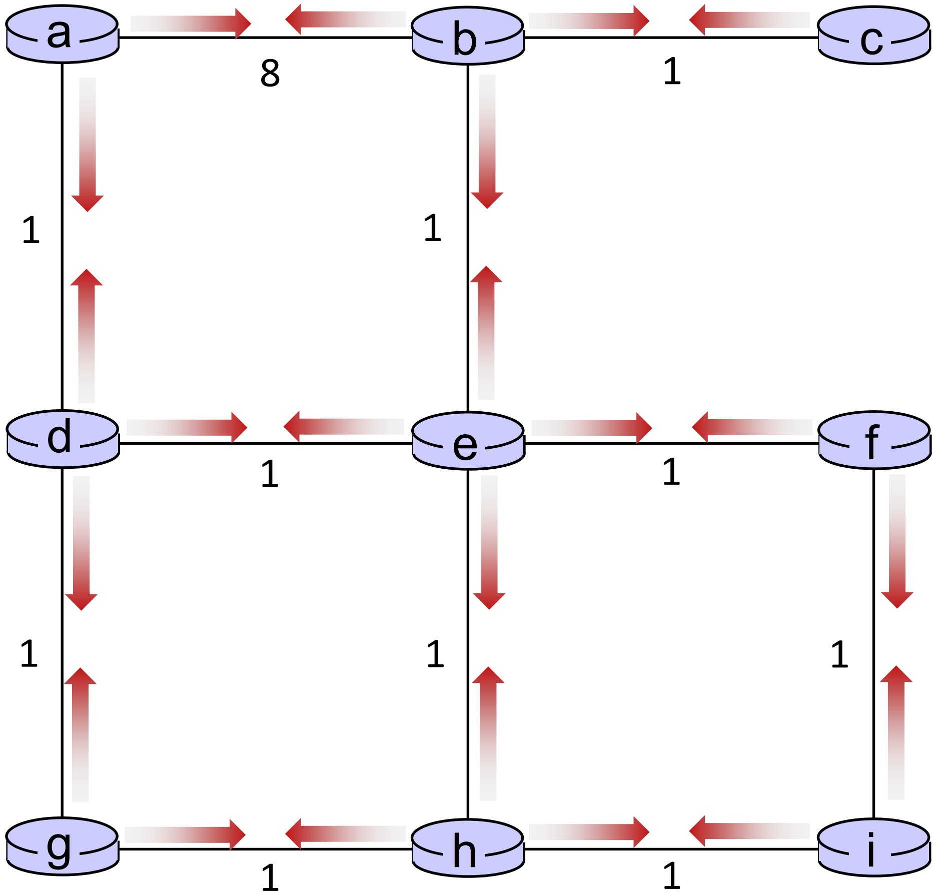
Distance vector example: iteration



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

- b receives DVs from a, c, e

DV in a:

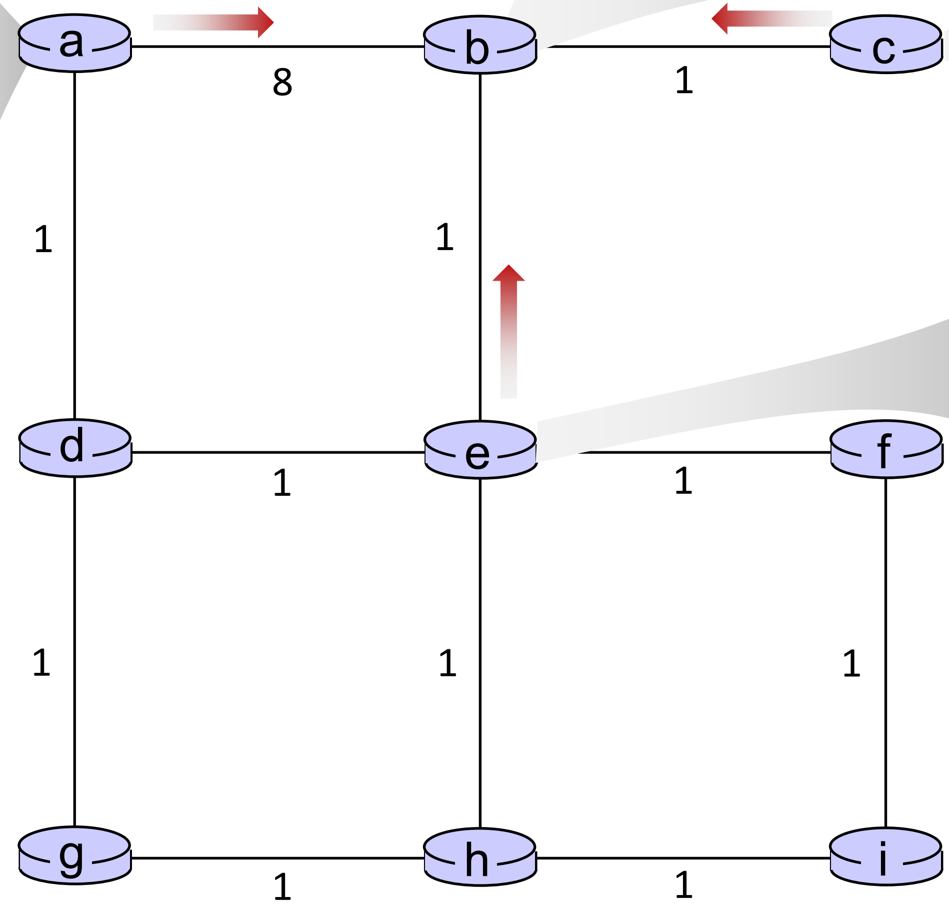
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

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:

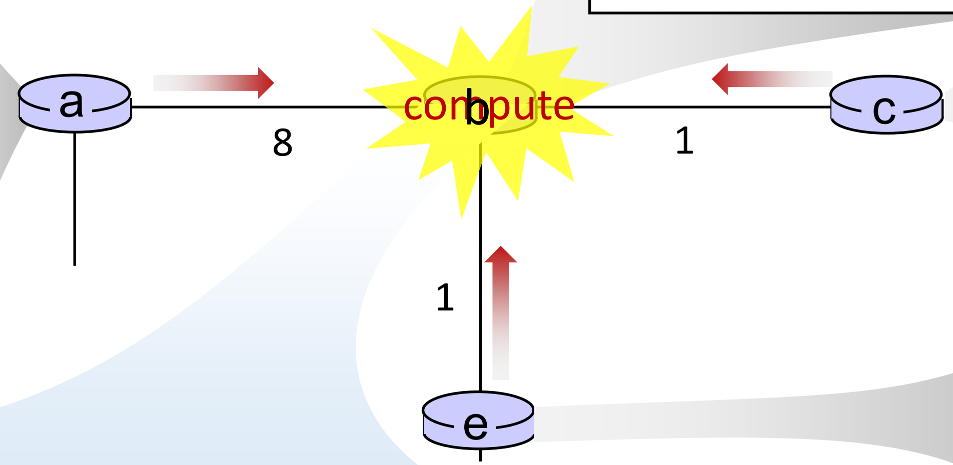
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

Distance vector example:

t=1

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

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

$D_b(a) = \min\{c_{b,a}+D_a(a),\ c_{b,c}+D_c(a),\ c_{b,e}+D_e(a)\} = \min\{8,\infty,\infty\} = 8$

$D_b(c) = \min\{c_{b,a}+D_a(c),\ c_{b,c}+D_c(c),\ c_{b,e}+D_e(c)\} = \min\{\infty,1,\infty\} = 1$

$D_b(d) = \min\{c_{b,a}+D_a(d),\ c_{b,c}+D_c(d),\ c_{b,e}+D_e(d)\} = \min\{8+1,\infty,1+1\} = 2$

$D_b(e) = \min\{c_{b,a}+D_a(e),\ c_{b,c}+D_c(e),\ c_{b,e}+D_e(e)\} = \min\{\infty,\infty,1\} = 1$

$D_b(f) = \min\{c_{b,a}+D_a(f),\ c_{b,c}+D_c(f),\ c_{b,e}+D_e(f)\} = \min\{\infty,\infty,1+1\} = 2$

$D_b(g) = \min\{c_{b,a}+D_a(g),\ c_{b,c}+D_c(g),\ c_{b,e}+D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty$

$D_b(h) = \min\{c_{b,a}+D_a(h),\ c_{b,c}+D_c(h),\ c_{b,e}+D_e(h)\} = \min\{\infty, \infty, 1+1\} = 2$

$D_b(i) = \min\{c_{b,a}+D_a(i),\ c_{b,c}+D_c(i),\ c_{b,e}+D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty$

DV in b:	
$D_b(a) = 8$	$D_b(f) =2$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = 2$	$D_b(h) = 2$
$D_b(e) = 1$	$D_b(i) = \infty$

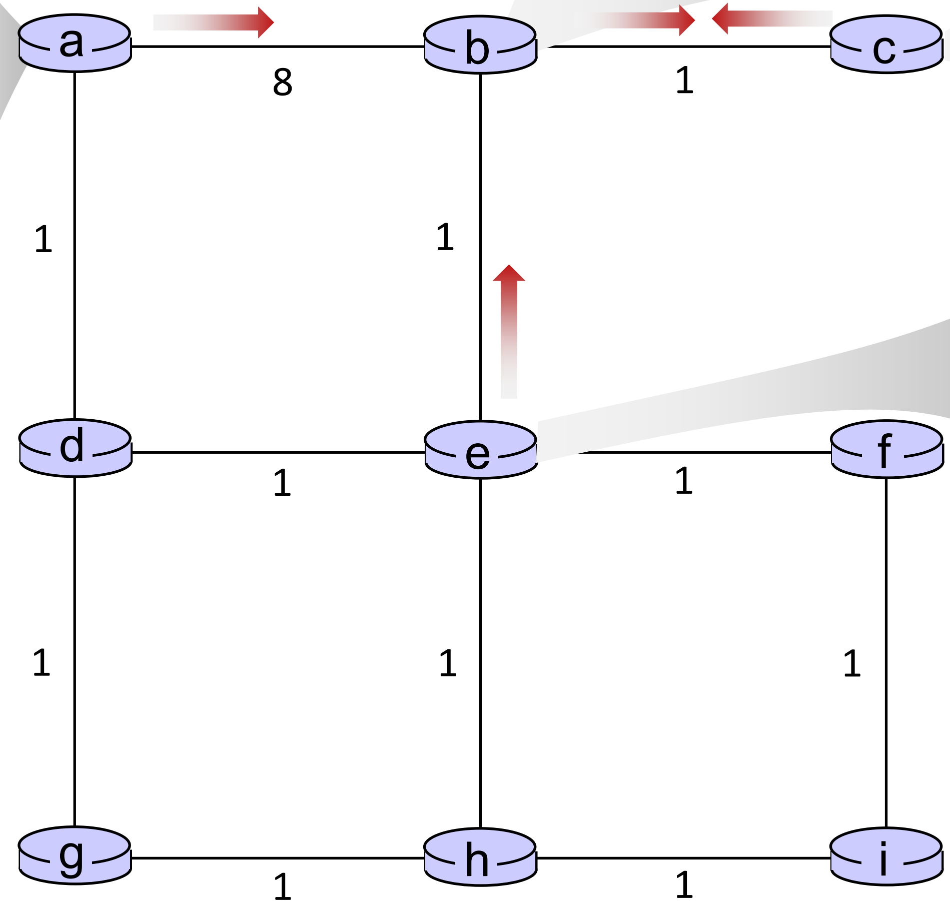
Distance vector example:

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:

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

DV in c:

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

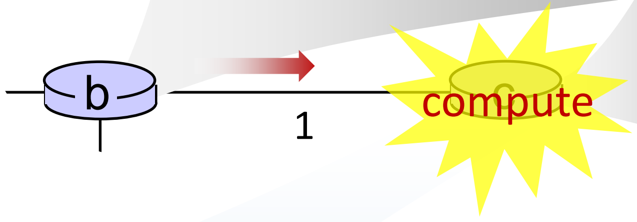
DV in e:

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

Distance vector example:

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



t=1

- c receives DVs from b computes:

$D_c(a) = \min\{c_{c,b}+D_b(a)\} = 1 + 8 = 9$

$D_c(b) = \min\{c_{c,b}+D_b(b)\} = 1 + 0 = 1$

$D_c(d) = \min\{c_{c,b}+D_b(d)\} = 1+ \infty = \infty$

$D_c(e) = \min\{c_{c,b}+D_b(e)\} = 1 + 1 = 2$

$D_c(f) = \min\{c_{c,b}+D_b(f)\} = 1+ \infty = \infty$

$D_c(g) = \min\{c_{c,b}+D_b(g)\} = 1+ \infty = \infty$

$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$

$D_c(i) = \min\{c_{c,b}+D_b(i)\} = 1+ \infty = \infty$

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

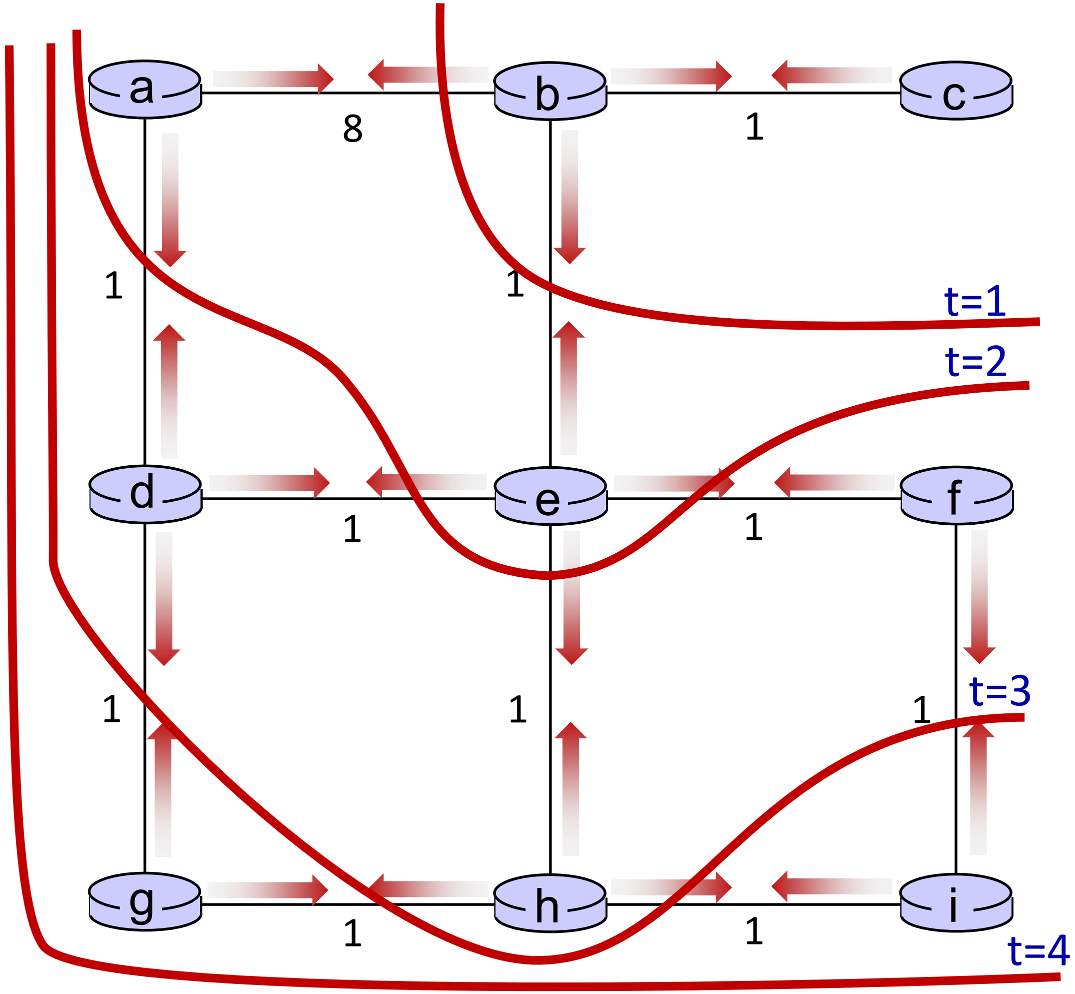
t=0 c’s state at t=0 is at c only

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

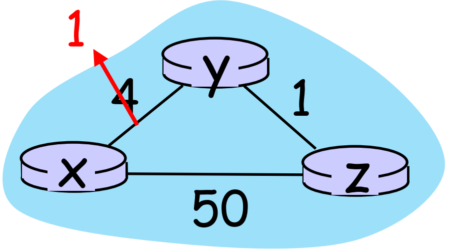
t=2 c’s state at t=0 may now influence distance vector computations up to **2** hops away i.e., at b and now at a, e as well

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

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



Distance vector: link cost changes



link cost changes:

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

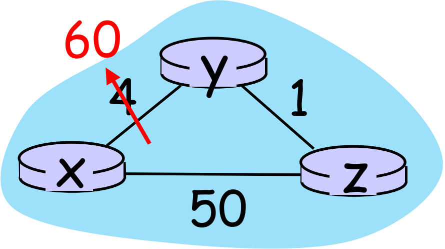
“good news travels fast”

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

t_1 : *z* receives update from *y*, updates its 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:
    - *y* sees direct link to *x* has new cost 60, but z has said it has a path at cost of 5. So *y* computes "my new cost to x will be 6, via z); notifies *z* of new cost of 6 to *x*.
    - *z* learns that path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y), notifies *y* of new cost of 7 to *x*.
    - *y* learns that path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y), notifies *z* of new cost of 8 to *x*.
    - *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y), notifies *y* of new cost of 9 to *x*.

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

Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

5.3 Intra-ISP routing: OSPF

5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Making routing scalable

our routing study thus far - idealized

- all routers identical
- network "flat"

… not true in practice

scale: billions of destinations:

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

administrative autonomy:

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

Internet approach to scalable routing

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

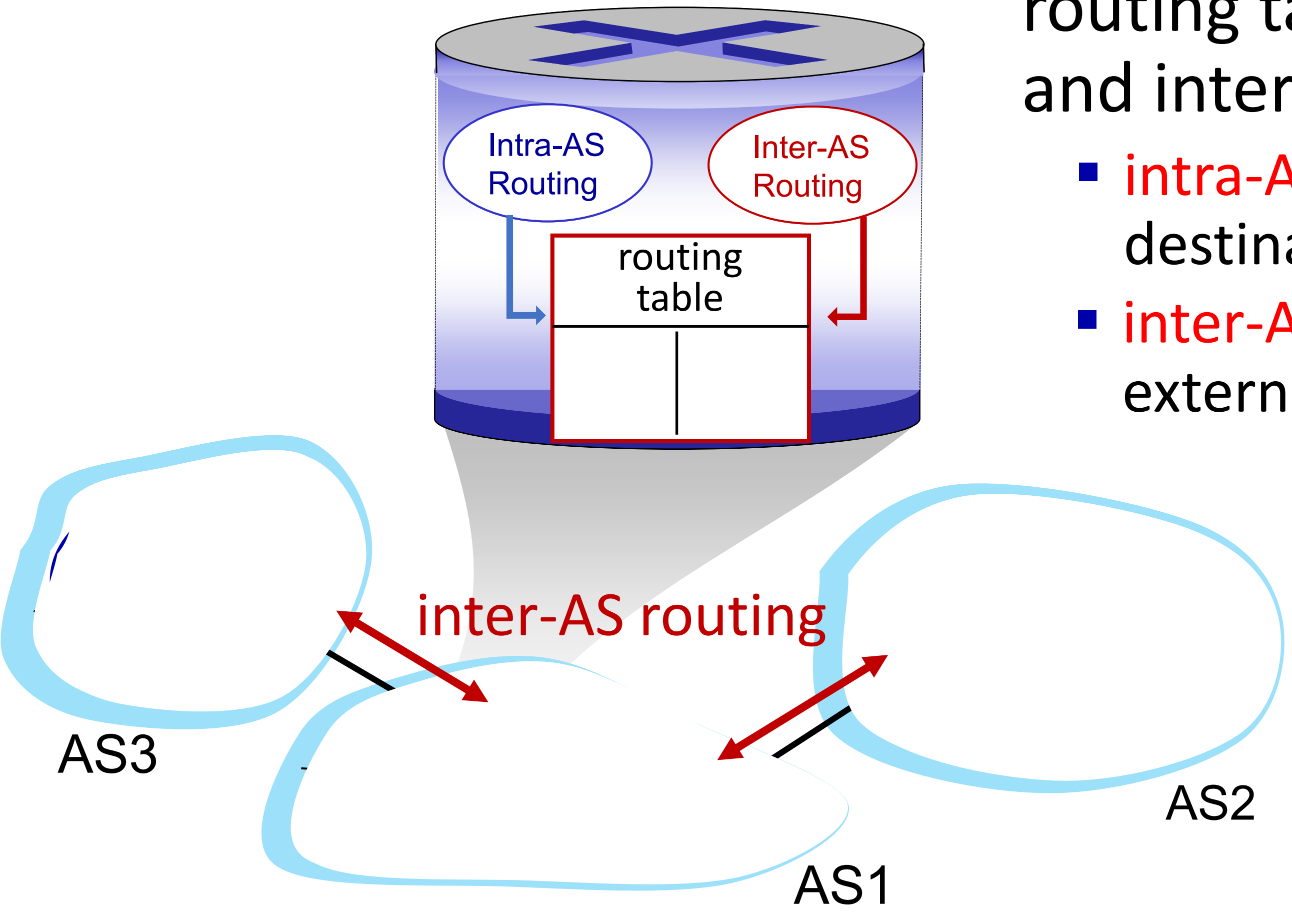
intra-AS (aka "intra-domain"): routing among *within same AS ("network")*

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

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

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

Interconnected ASes



routing tables configured by intra- and inter-AS routing algorithms

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

Intra-AS routing: Routing within an AS

Most common intra-AS routing protocols:

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

OSPF (Open Shortest Path First) routing

- centralized routing algorithm
  - each router has full topology, uses Dijkstra’s algorithm to compute routing table
  - 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
- *security:* all OSPF messages authenticated (to prevent malicious intrusion)

Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

5.3 Intra-ISP routing: OSPF

5.4 Routing among ISPs: BGP

5.5 SDN control plane

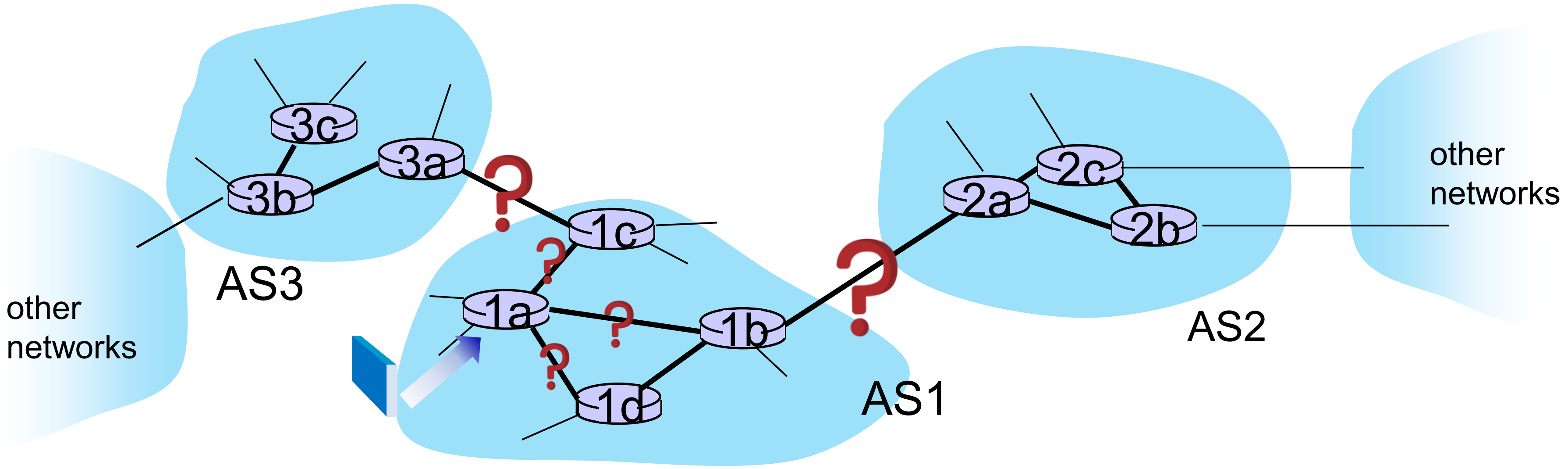
5.6 Internet Control Message Protocol

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:

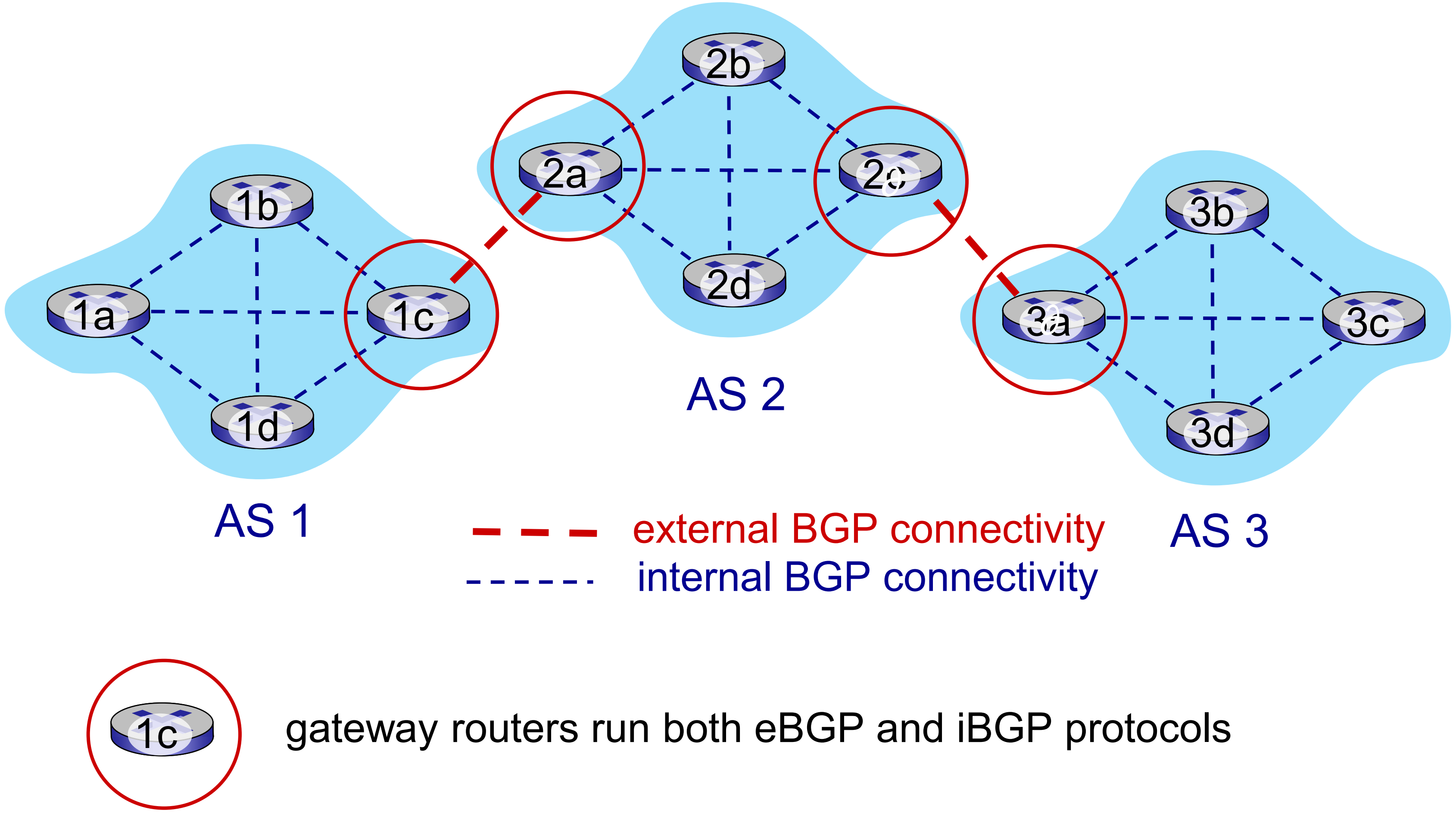
1. learn which destinations are reachable through AS2 and AS3
2. propagate this reachability info to all routers in AS1



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:
  - external BGP: obtain subnet reachability information from neighboring ASes
  - internal BGP: propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and *policy*

eBGP, iBGP connections



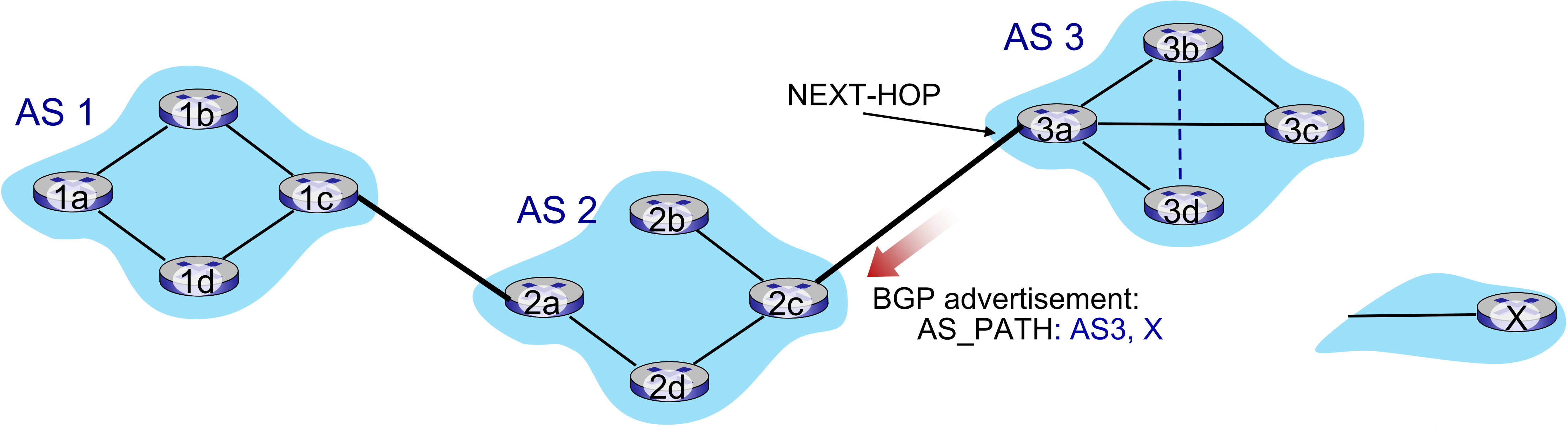
gateway routers run both eBGP and iBGP protocols

Path attributes and BGP routes

- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of AS's through which prefix advertisement has passed
    - NEXT-HOP: indicates IP address of the router interface that begins the AS-PATH
- 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

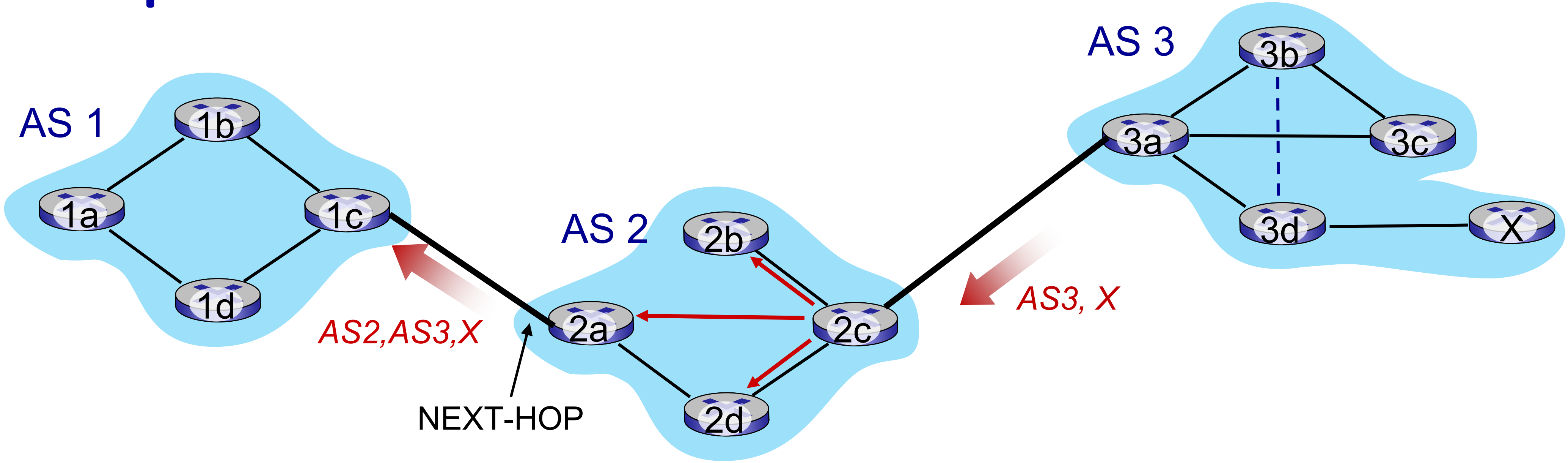
BGP basics

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



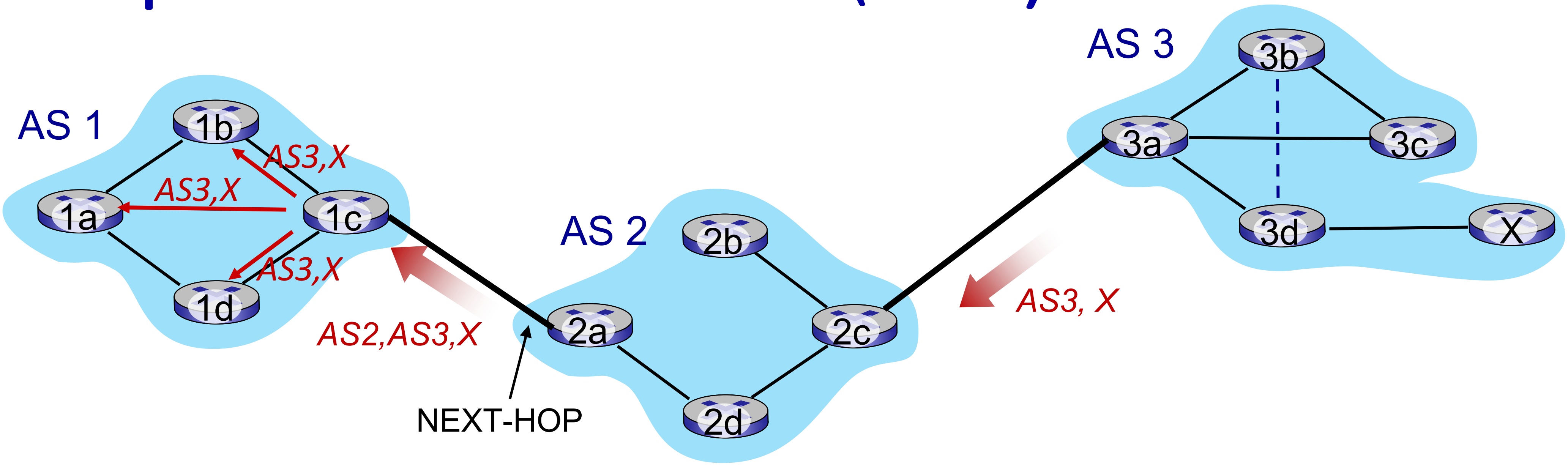
BGP advertisement:
AS_PATH: AS3, X

BGP path advertisement



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

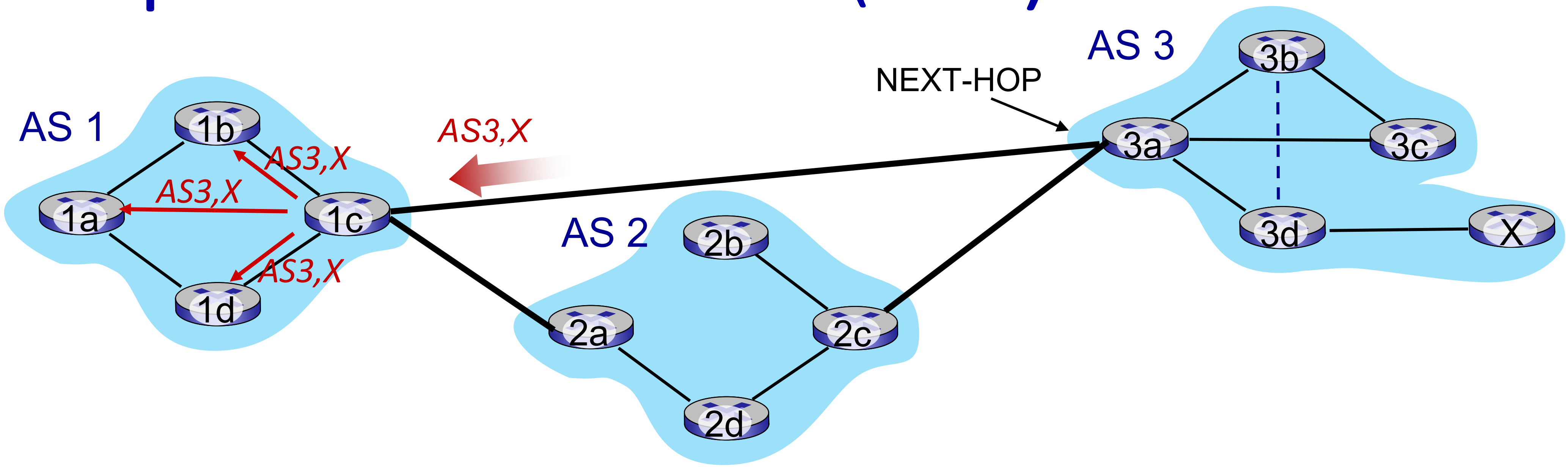
BGP path advertisement (more)



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 internal BGP

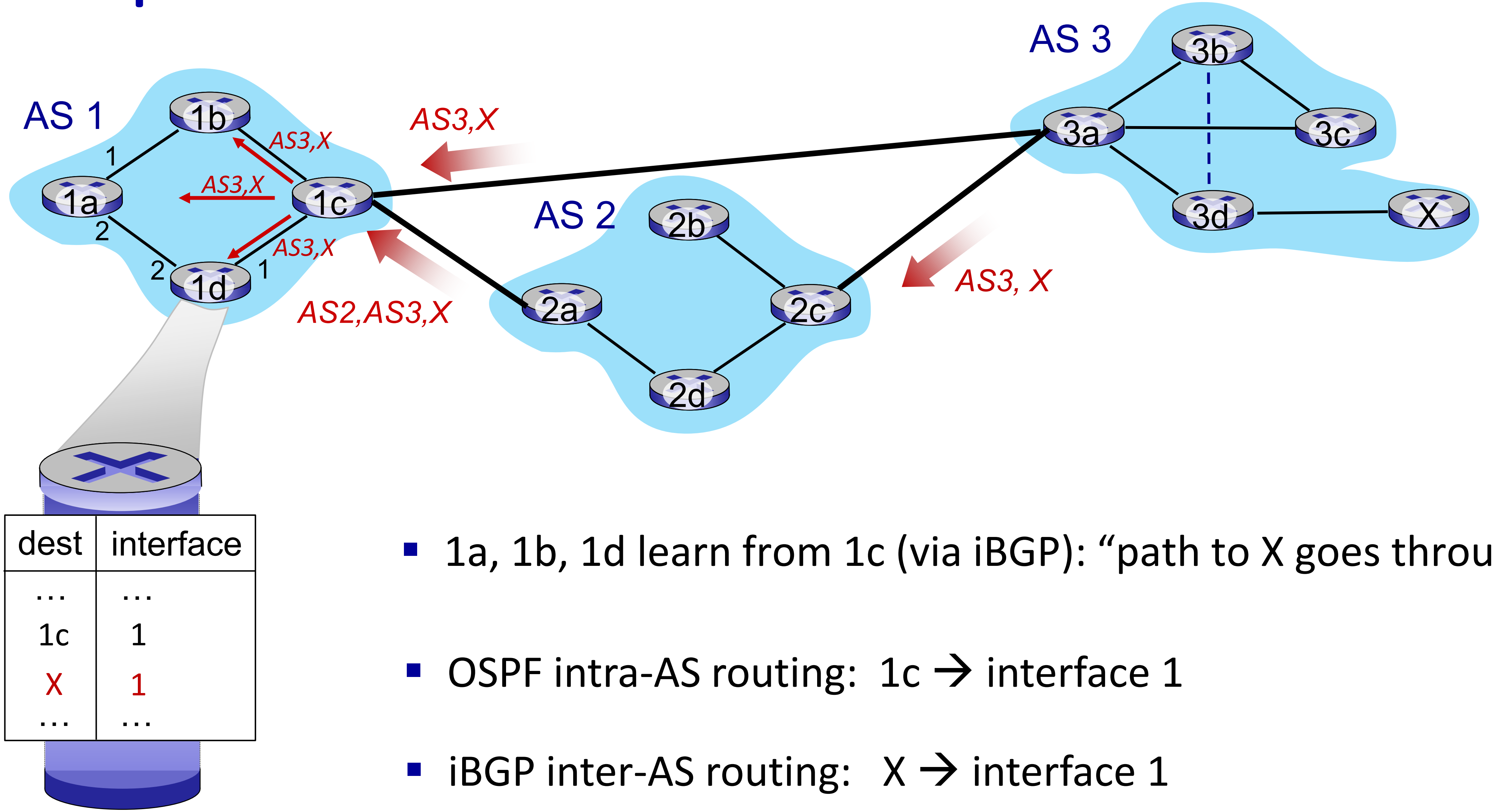
BGP path advertisement (more)



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 internal BGP

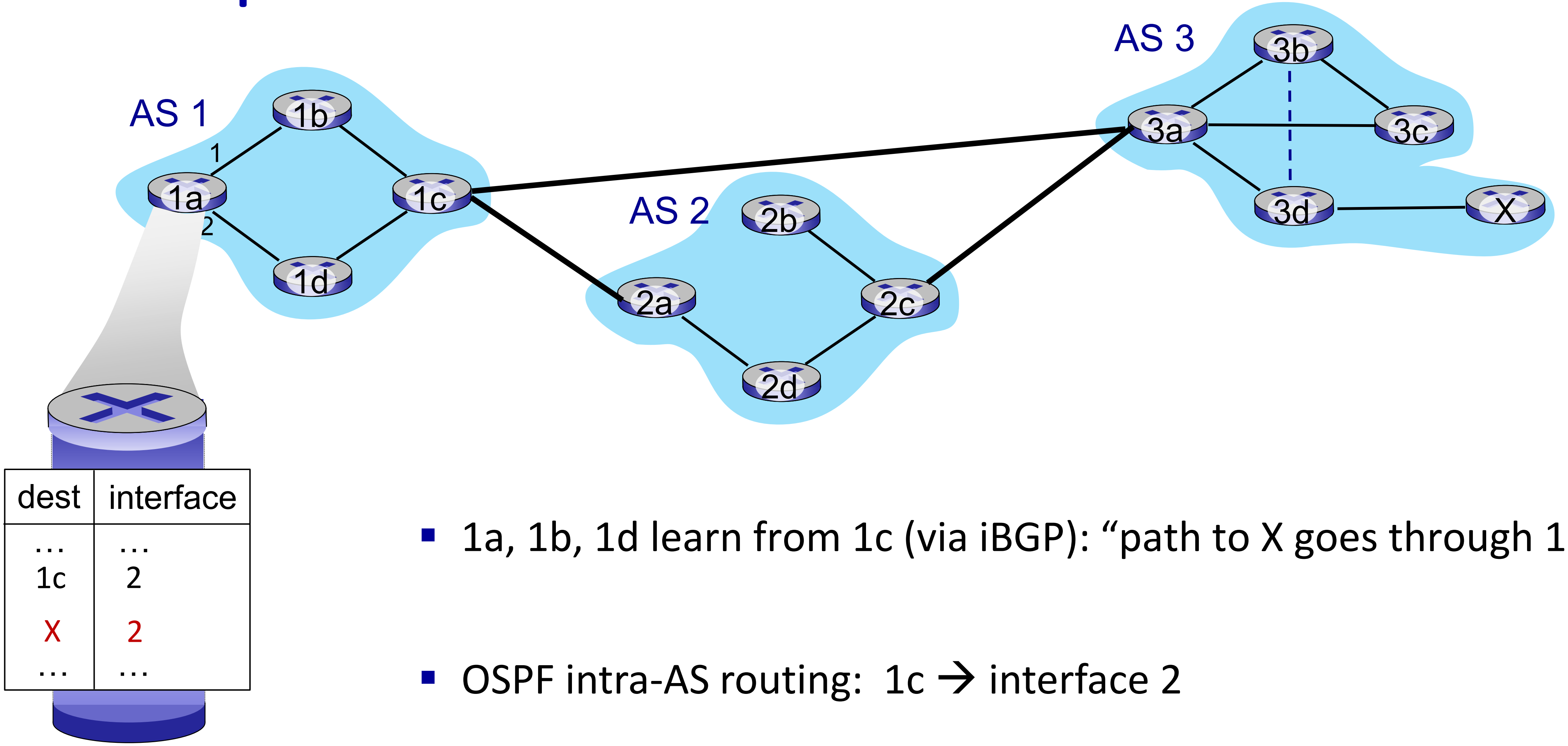
BGP path advertisement



dest	interface
…	…
1c	1
X	1
…	…

- 1a, 1b, 1d learn from 1c (via iBGP): “path to X goes through 1c”
- OSPF intra-AS routing: 1c → interface 1
- iBGP inter-AS routing: X → interface 1

BGP path advertisement



dest	interface
…	…
1c	2
X	2
…	…

- 1a, 1b, 1d learn from 1c (via iBGP): “path to X goes through 1c”
- OSPF intra-AS routing: 1c → interface 2
- iBGP inter-AS routing: X → interface 2

Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

5.3 Intra-ISP routing: OSPF

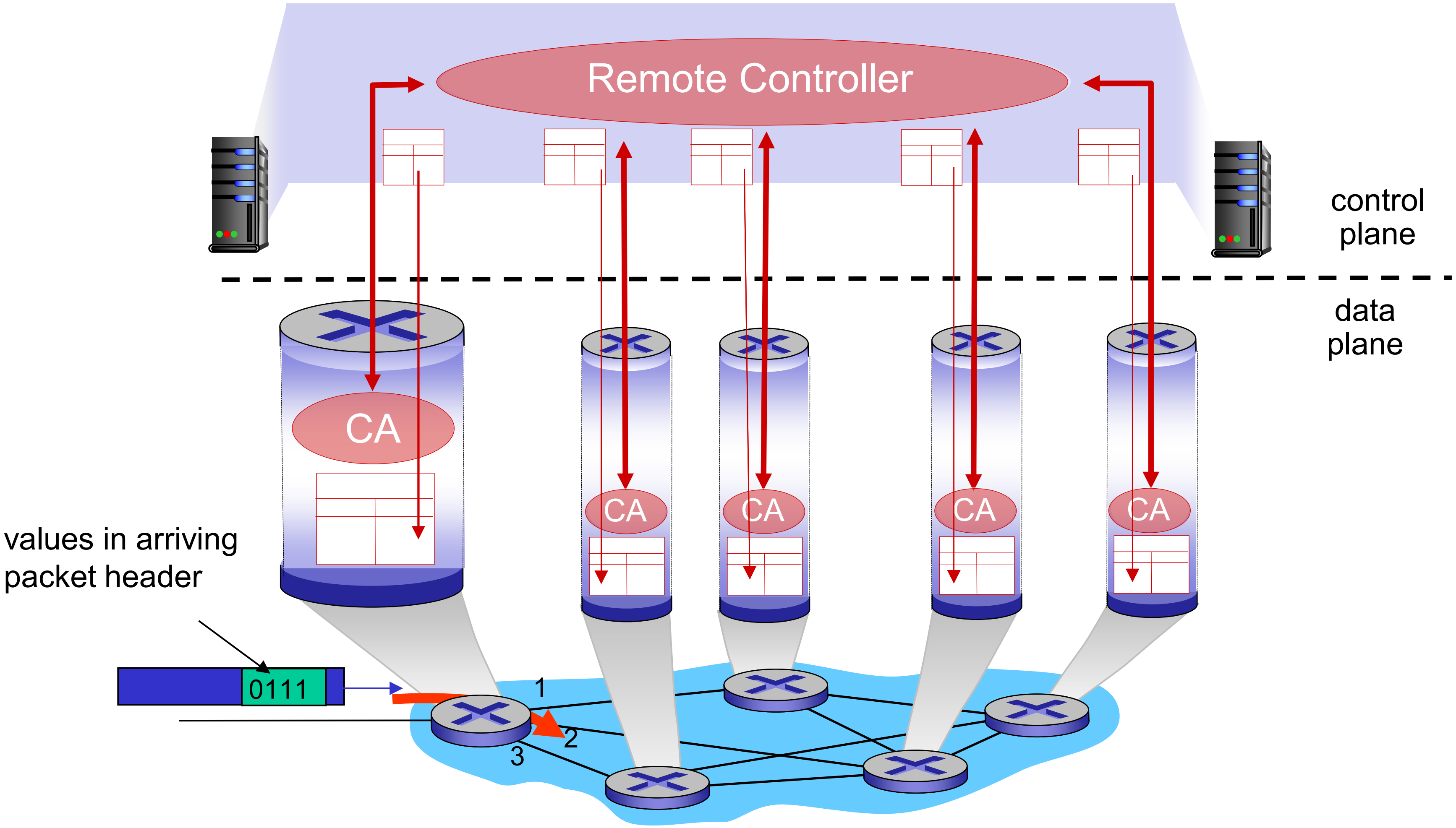
5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



Software defined networking (SDN)

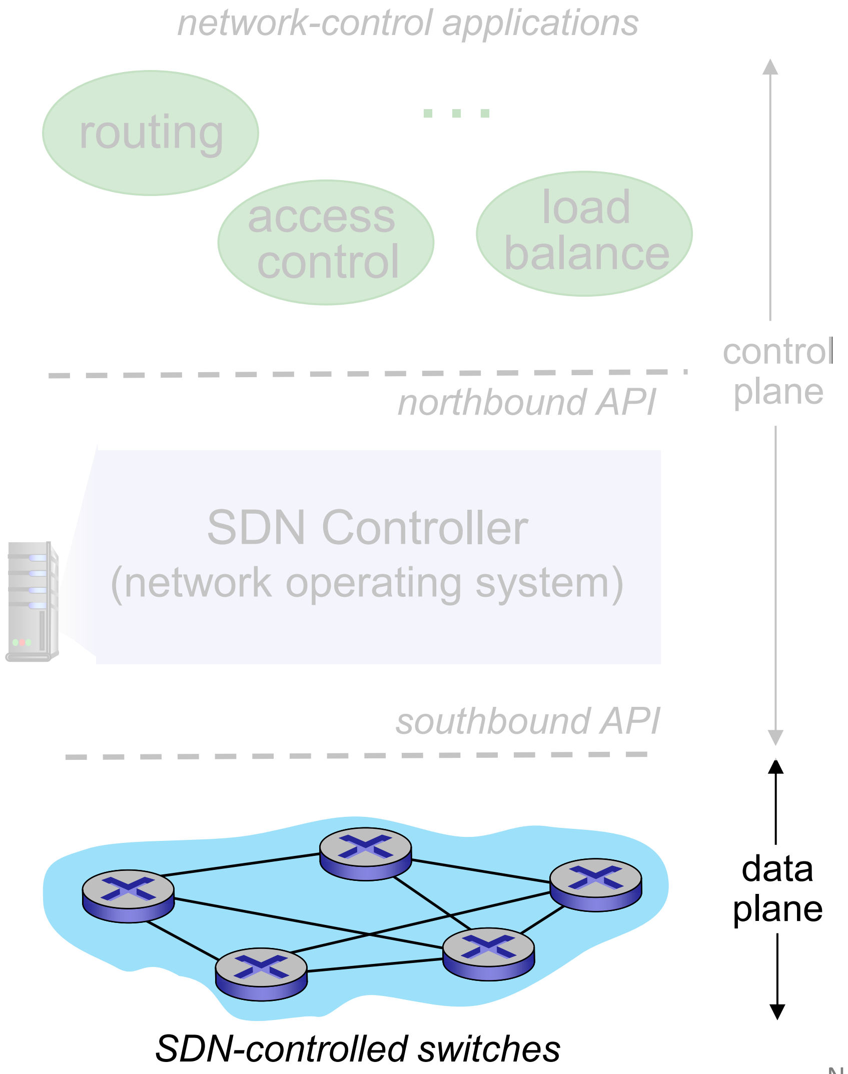
Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- "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

Software defined networking (SDN)

Data-plane switches:

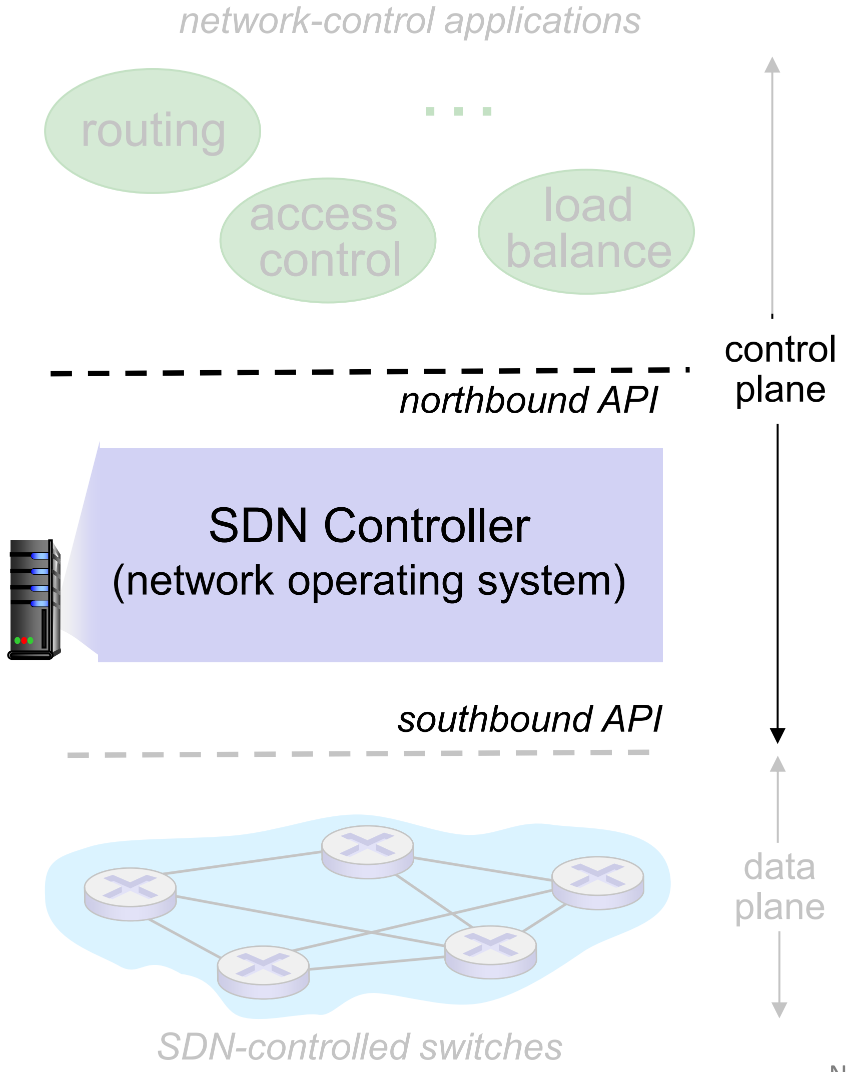
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating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



Software defined networking (SDN)

SDN controller (network OS):

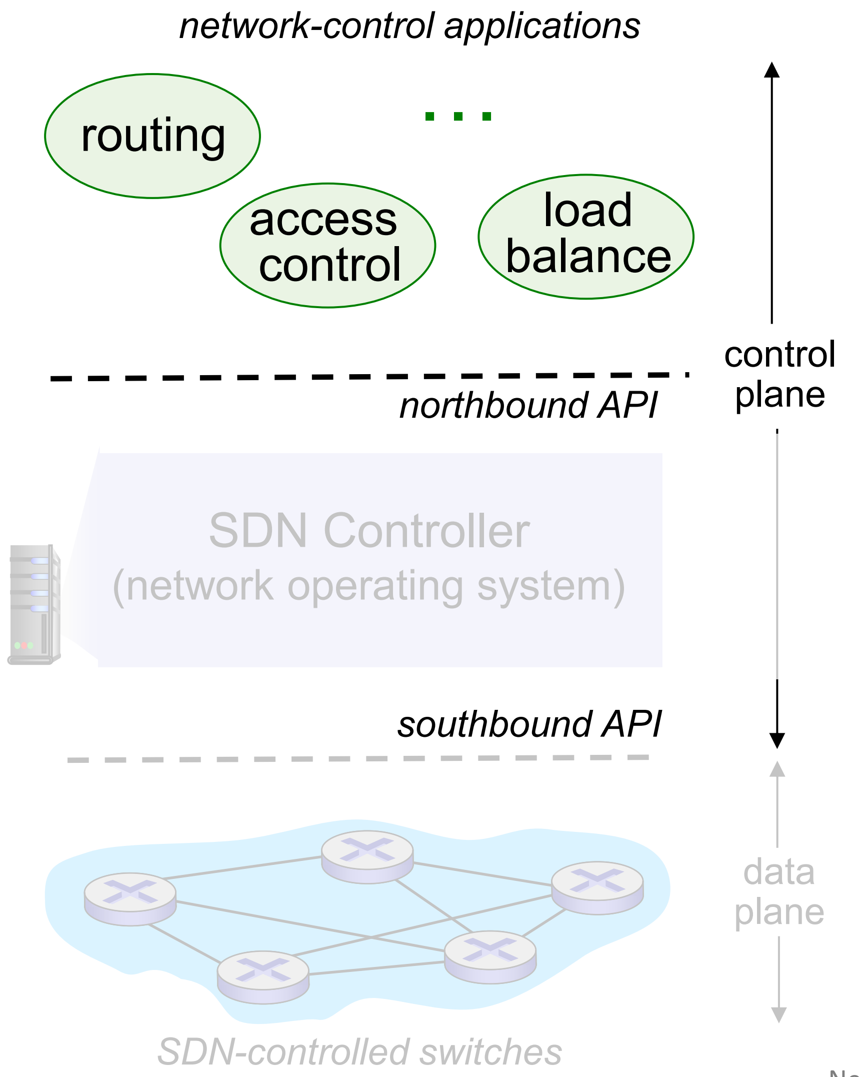
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



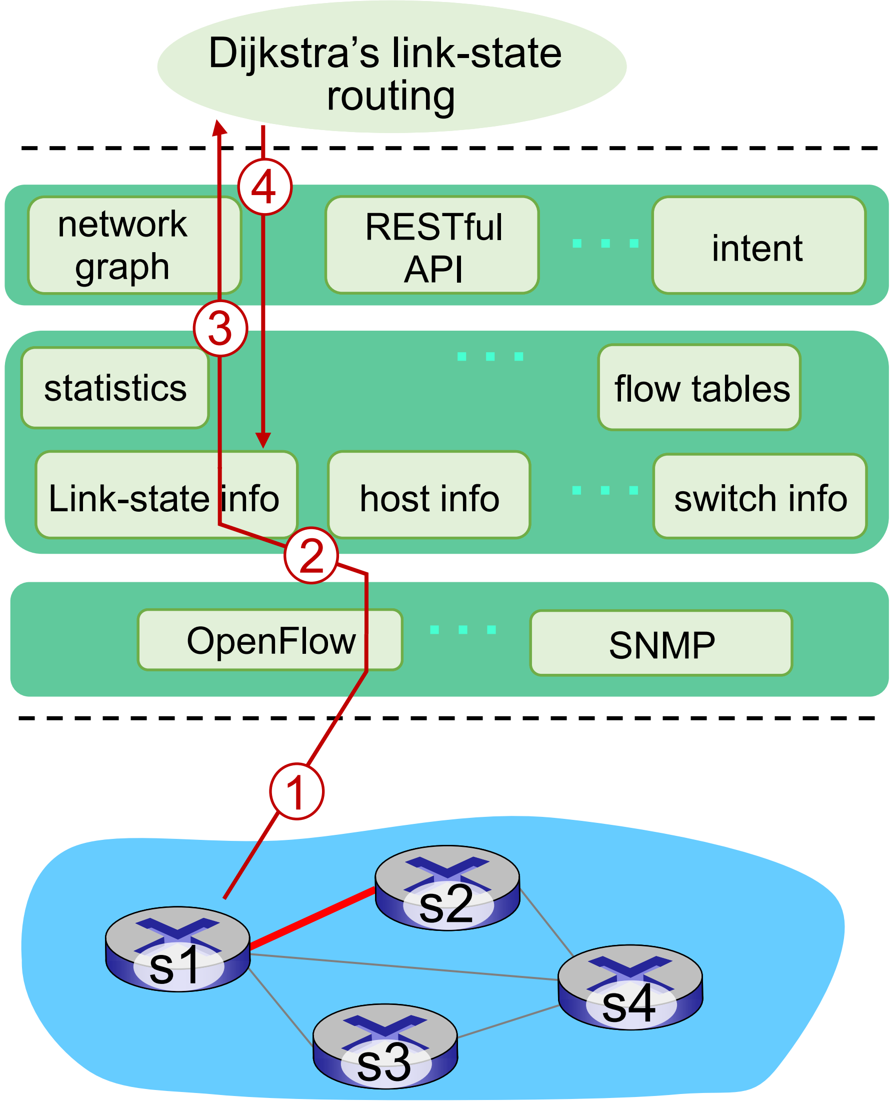
Software defined networking (SDN)

network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



SDN: control/data plane interaction example



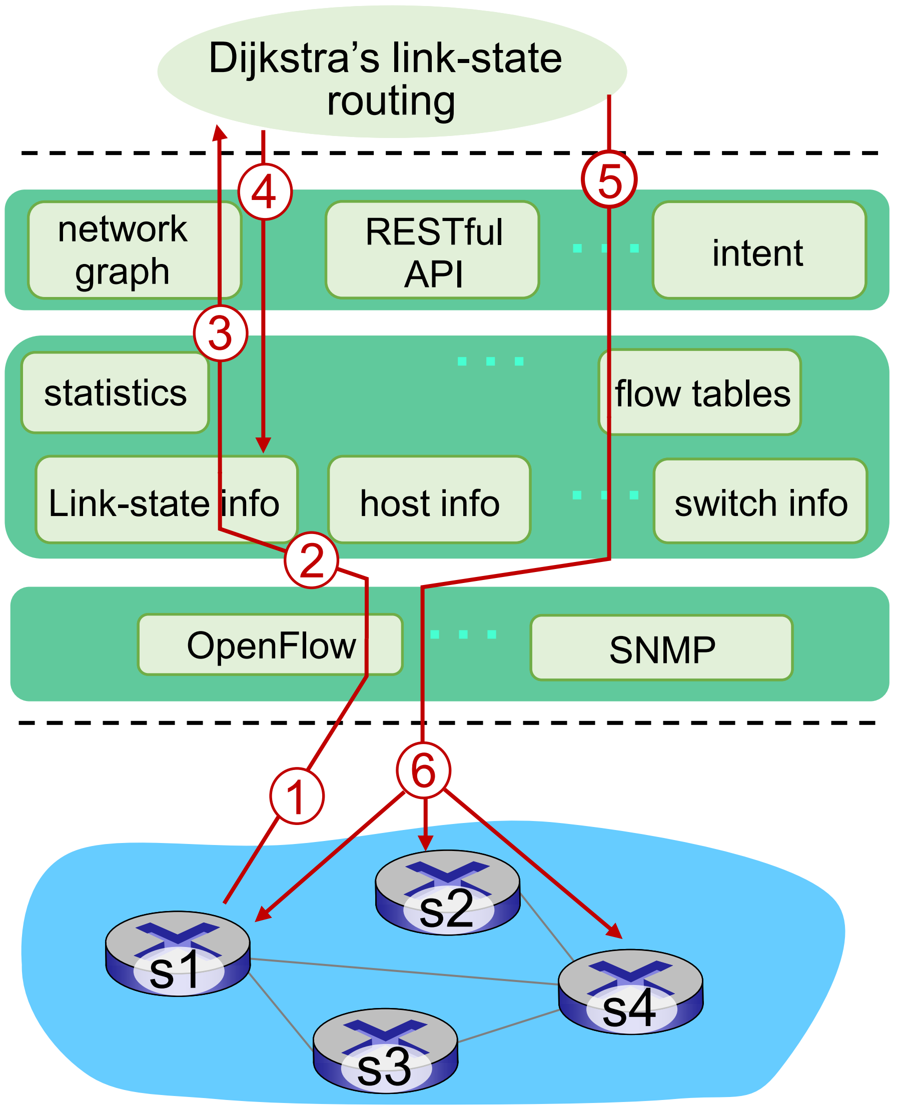
① S1, experiencing link failure uses OpenFlow port status message to notify controller

② SDN controller receives OpenFlow message, updates link status info

③ Dijkstra's routing algorithm application has previously registered to be called when ever 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

NoviSwitch™ 2150 High Performance OpenFlow Switch

NoviSwitch 2150 is an OpenFlow switch offering genuine wire-speed performance using the OpenFlow 1.3 to 1.5 standards and has been specifically designed for use in high bandwidth / flow-intensive network deployments. Includes the *NoviWare™ 400.4* OpenFlow Switch Software for use with the Mellanox high performance NP-5 network processor.

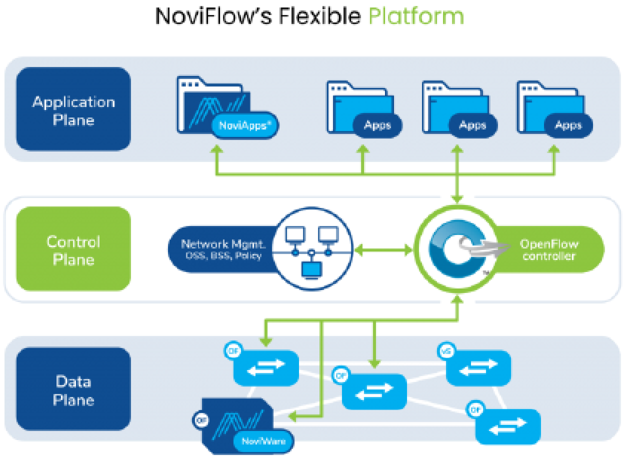
Today's major network operators demand flexible, scalable networking solutions that deliver wire-speed performance. **NoviFlow Inc.™** is changing the traditional approach to networking by making switching smarter. The company delivers upon the promise of OpenFlow and Software Defined Networking (SDN) by combining the benefits of virtualization and programmability with network processors that can handle complex flows to make it possible for carriers, cloud providers and hyperscale data centers to keep up with today's exponentially growing networking demand.

NoviSwitch 2150 was specifically designed for deployment in Access Networks and data centers looking to leverage the benefits of SDN to improve the cost/performance, security, scalability and flexibility of networks. It is a forwarding plane platform delivering maximum OpenFlow capability in a compact form factor. The system is provided in a stand-al0one, self-contained, 1U rack-mountable enclosure box that can be configured to support a wide variety of networking applications to deliver unmatched performance levels.

Key Features:

Features the NoviWare 400.4 OpenFlow switching software, supporting <u>all</u> required and optional OpenFlow 1.3 and 1.4 match fields, instructions, actions and counters, as well as key OpenFlow 1.5 features, including group chaining.

- 256 Gbps and 190 Mpps of switching capacity powered by an Mellanox NP-5 NPU
- 50 data plane ports:
  - 2 QSFP+ transceiver cages for 40GE connectivity
  - 48 SFP transceiver cages for 10/100/1000BASE-T/100BASE-FX/1000BASE-X connectivity
- Up to 1 Million wildcard match flow (with optional external TCAM), otherwise 16,000 flow entries in up to 60 tables
- Up to 40,000 flow-mods/second
- Up to 1 Million meters and 10,000 entries in Groups table



Network layer: “control plane” roadmap

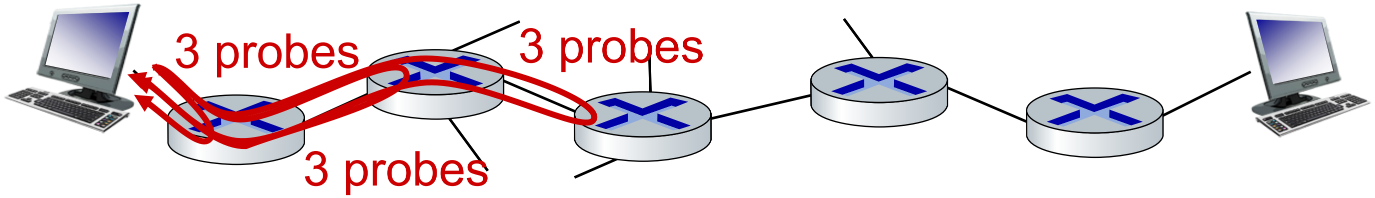
- 5.1 Introduction
- 5.2 Routing algorithms
- 5.3 Intra-ISP routing: OSPF
- 5.4 Routing among ISPs: BGP
- 5.5 SDN control plane
- 5.6 Internet Control Message Protocol

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
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth 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

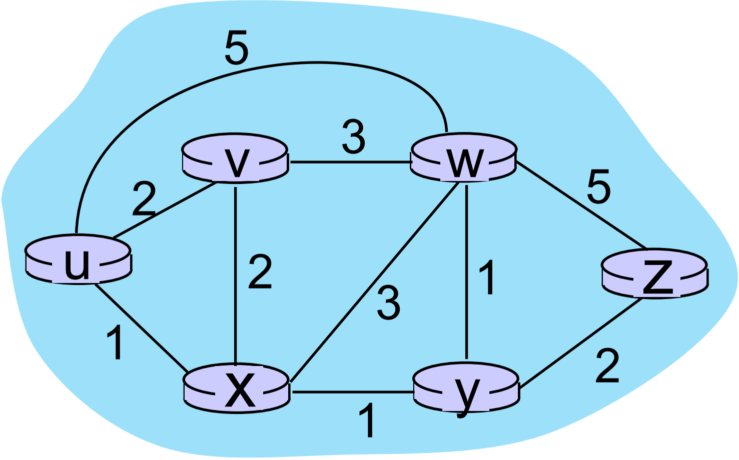
Network layer, control plane: Done!

- Introduction
- Routing algorithms
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

Original slides on Dijkstra example

Dijkstra’s algorithm: an example

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

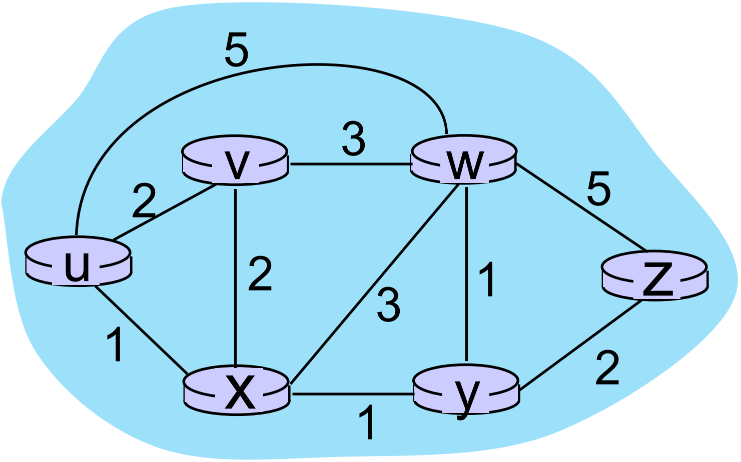


Initialization (step 0):

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

Dijkstra’s algorithm: an example

		t	u	v	w	y
Step	N	d(t), p(t)	d(u), p(u)	d(v), p(v)	d(w), p(w)	d(y), p(y)
0						
1						
2						
3						
4						
5						

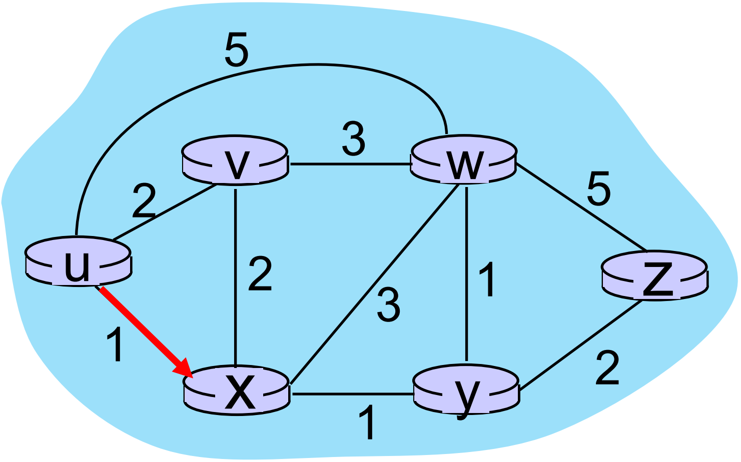


Initialization (step 0):

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

Dijkstra’s algorithm: an example

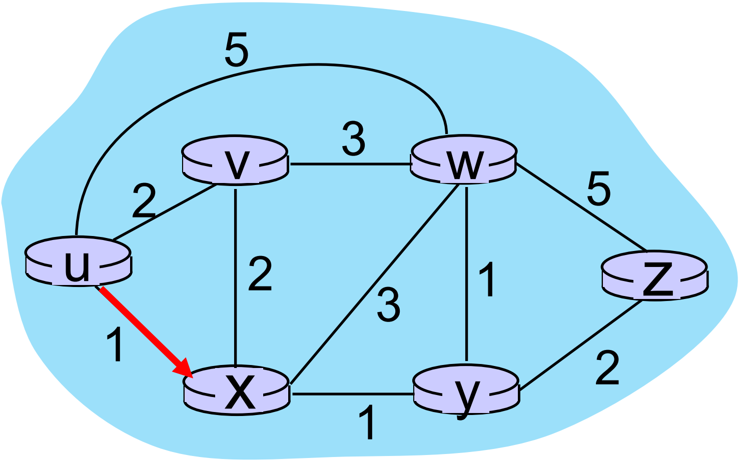
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux					
2						
3						
4						
5						



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

Dijkstra’s algorithm: an example

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



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

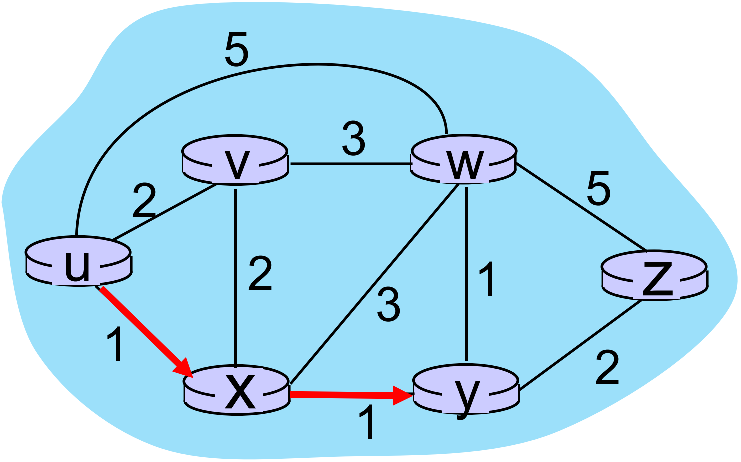
$D(v) = \min (D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$

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

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

Dijkstra’s algorithm: an example

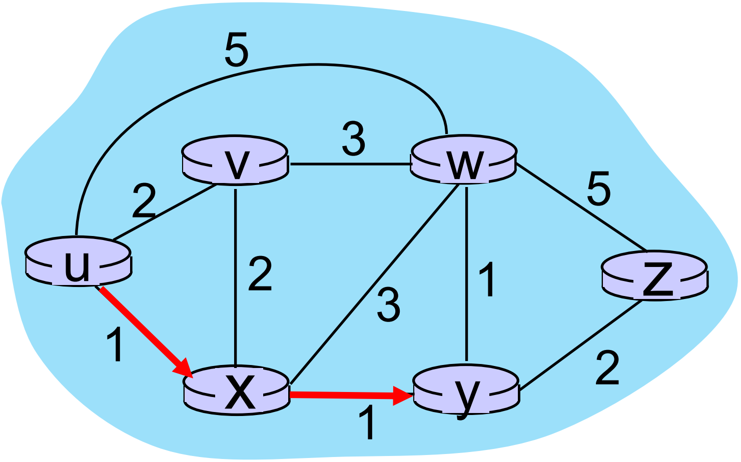
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy					
3						
4						
5						



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

Dijkstra’s algorithm: an example

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



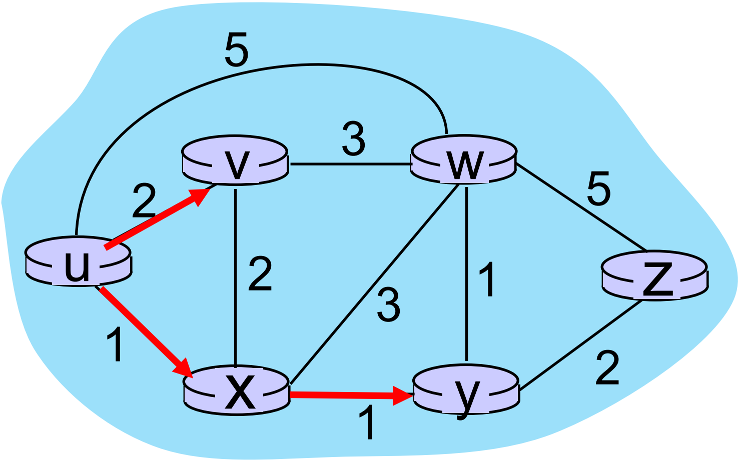
```
Loop
    find a not in N' such that D(a) is a minimum
    add a to N'
    update D(b) for all b adjacent to a and not in N' :
        D(b) = min ( D(b), D(a) + c_{a,b} )
```

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

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

Dijkstra’s algorithm: an example

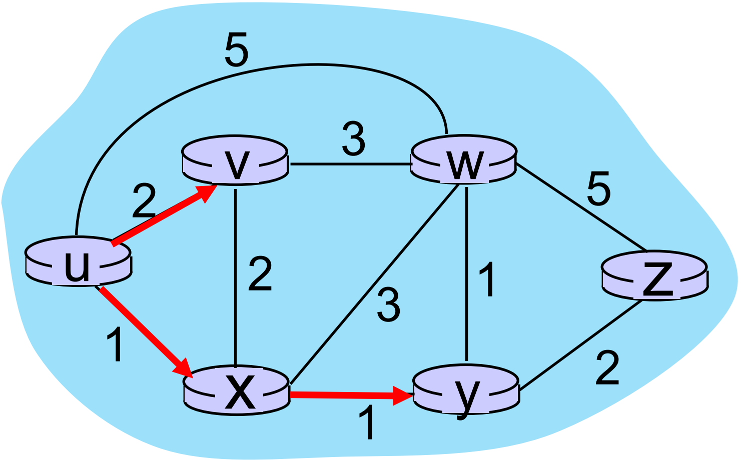
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv					
4						
5						



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

Dijkstra’s algorithm: an example

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

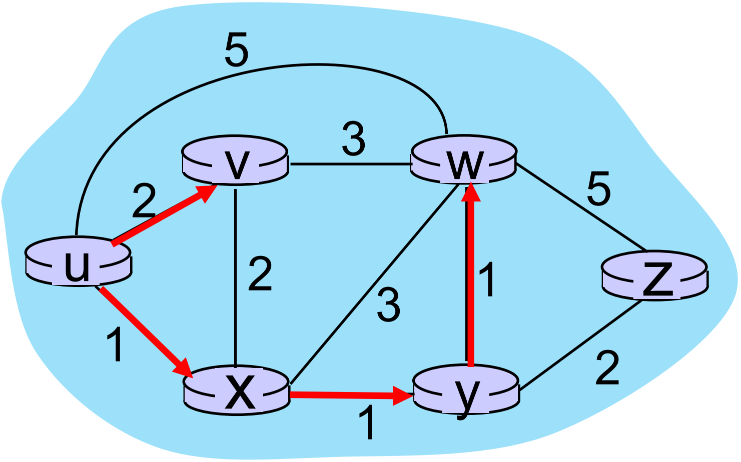


```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
    update $D(b)$ for all $b$ adjacent to $a$ and not in $N'$ :
        $D(b) = \min ( D(b), D(a) + c_{a,b} )$
```

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

Dijkstra’s algorithm: an example

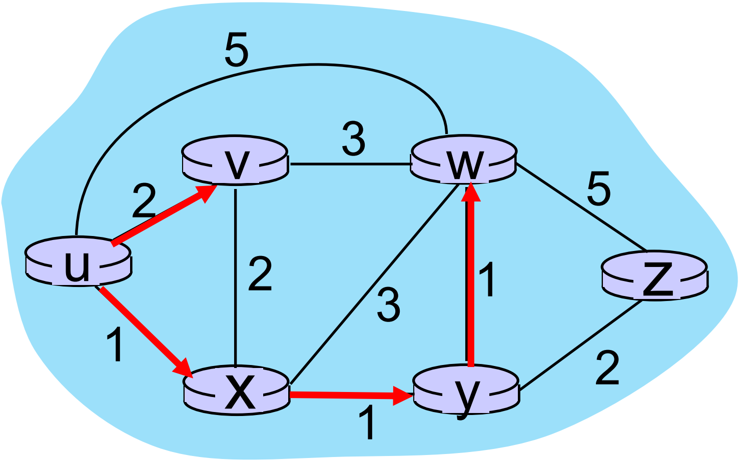
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					
5						



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

Dijkstra’s algorithm: an example

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

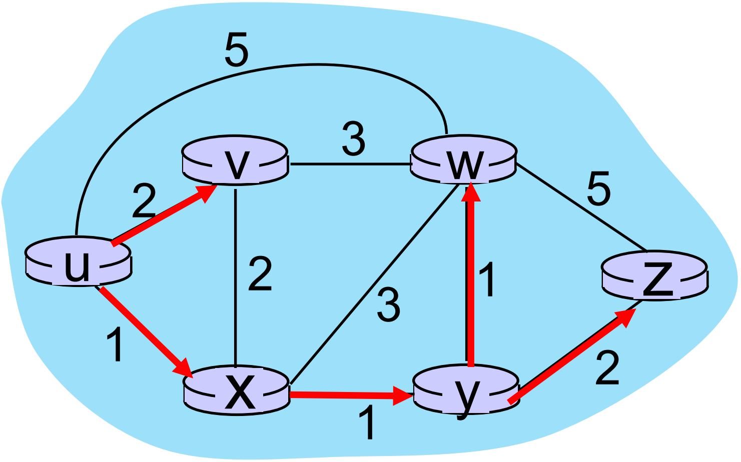


```
Loop
    find a not in N' such that D(a) is a minimum
    add a to N'
    update D(b) for all b adjacent to a and not in N' :
        D(b) = min ( D(b), D(a) + c_{a,b} )
```

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

Dijkstra’s algorithm: an example

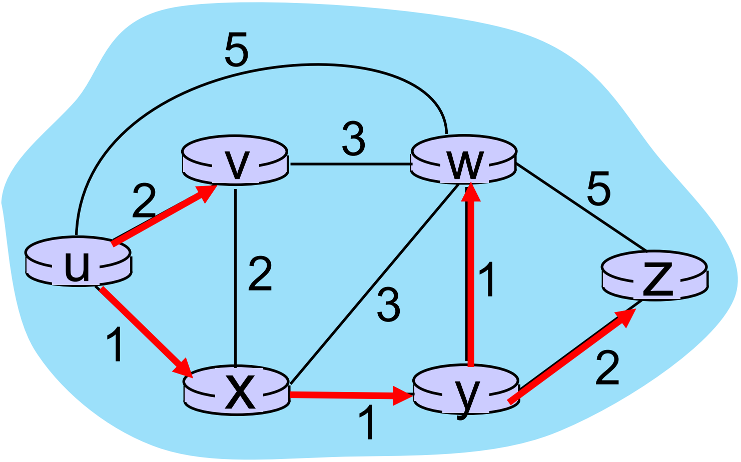
		v	w	x	y	z
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



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

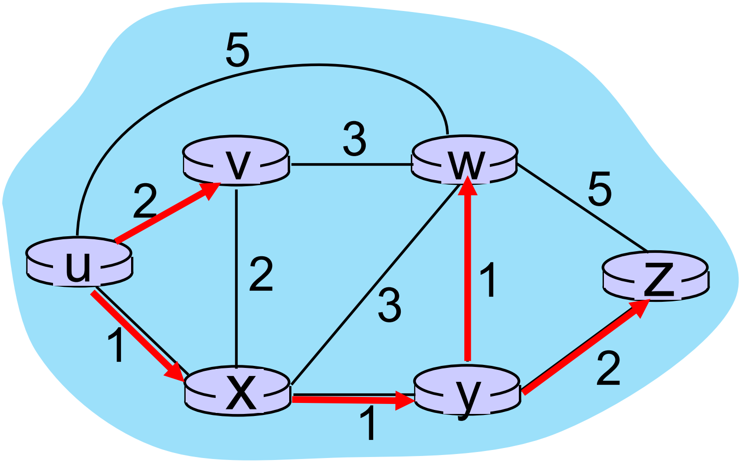
Dijkstra’s algorithm: an example

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

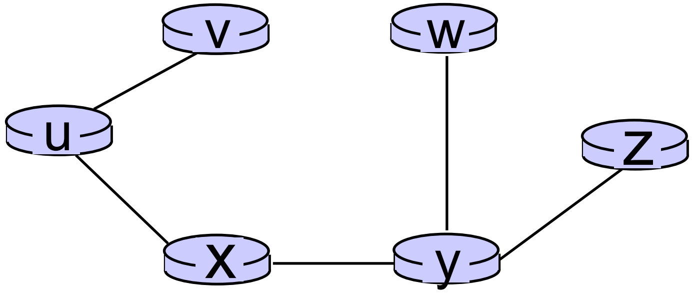


```
Loop
    find $a$ not in $N'$ such that $D(a)$ is a minimum
    add $a$ to $N'$
    update $D(b)$ for all $b$ adjacent to $a$ and not in $N'$ :
        $D(b) = \min ( D(b), D(a) + c_{a,b} )$
```

Dijkstra’s algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

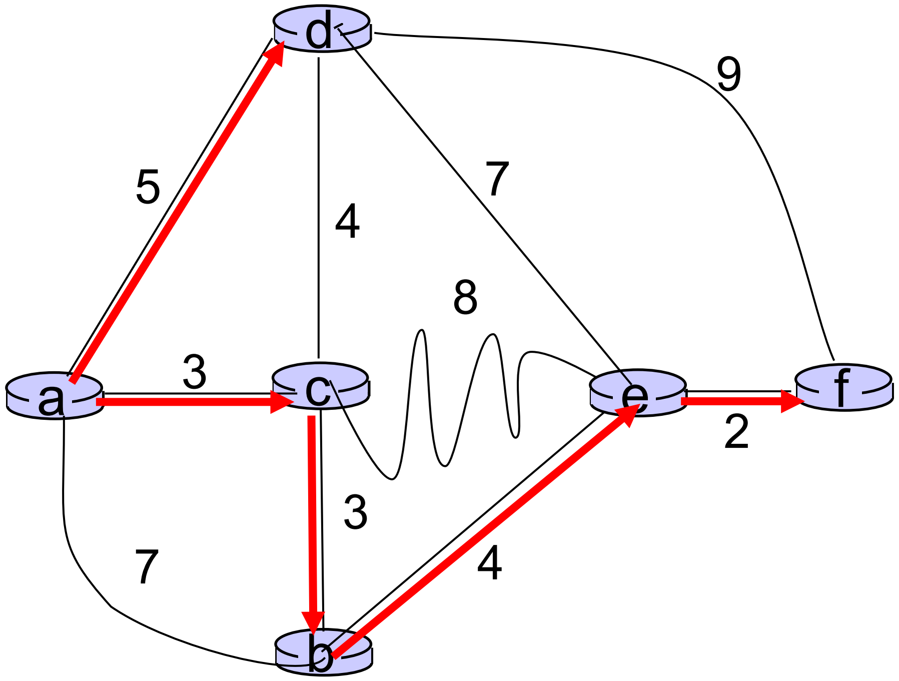
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Dijkstra’s algorithm: another example

		b	*c*	*d*	*e*	*f*
Step	*N'*	*D(b), p(b)*	*D(c), p(c)*	*D(d), p(d)*	*D(e), p(e)*	*D(f), p(f)*
0	a	7,a	3,a	5,a	∞	∞
1	ac	6,c		5,a	11,c	∞
2	acd	6,c			11,c	14,d
3	acdb				10,b	14,d
4	acdbe					12,e
5	acdbef					



notes:

- construct least-cost-path tree by tracing predecessor nodes