

Chapter 5: network layer control plane

chapter goals: understand principles behind network control plane

- traditional routing algorithms
- SDN controllers
- Internet Control Message Protocol
- network management

and their instantiation, implementation in the Internet:

- OSPF, BGP, OpenFlow, ODL and ONOS controllers, ICMP, SNMP

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

Network-layer functions

Recall: two 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

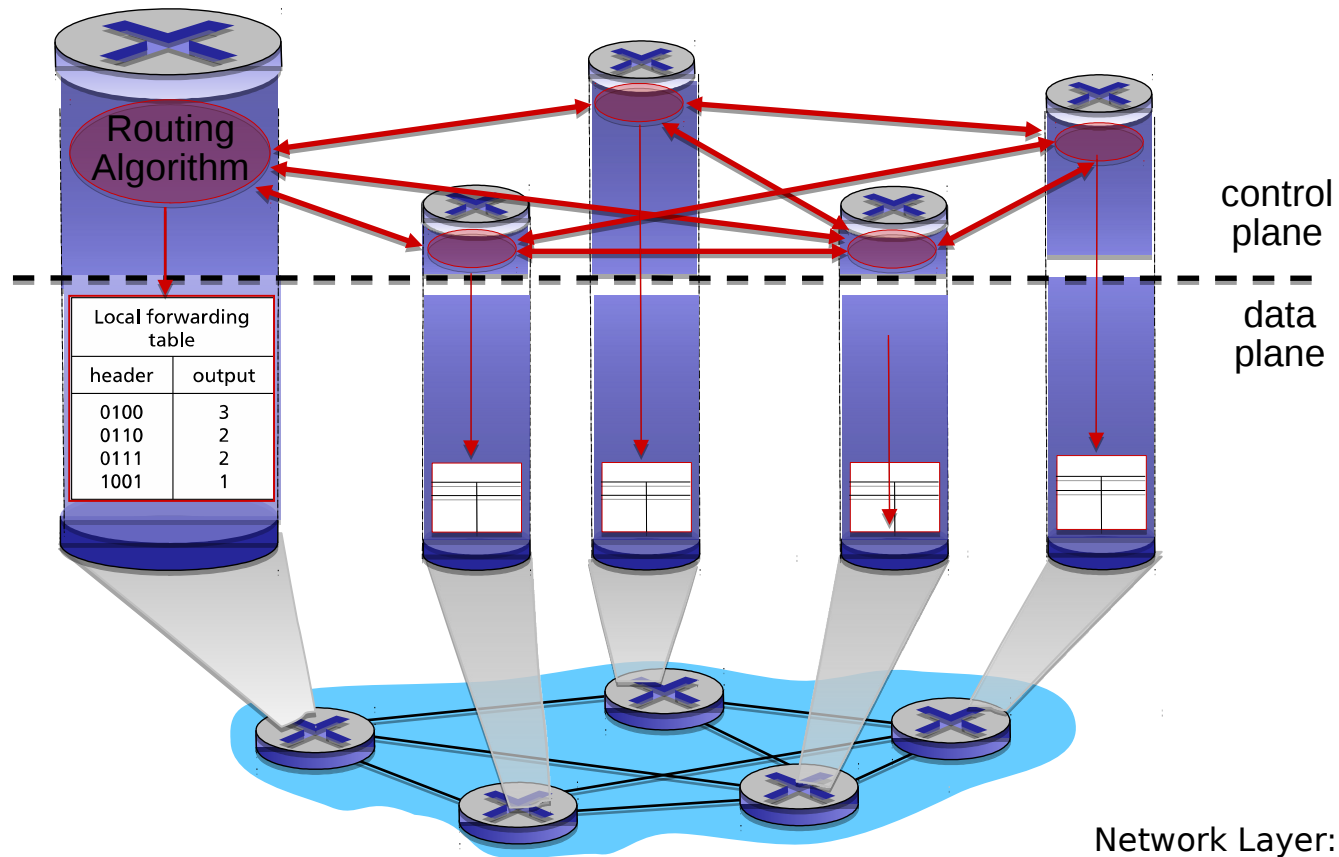
control plane

Two approaches to structuring network control plane:

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

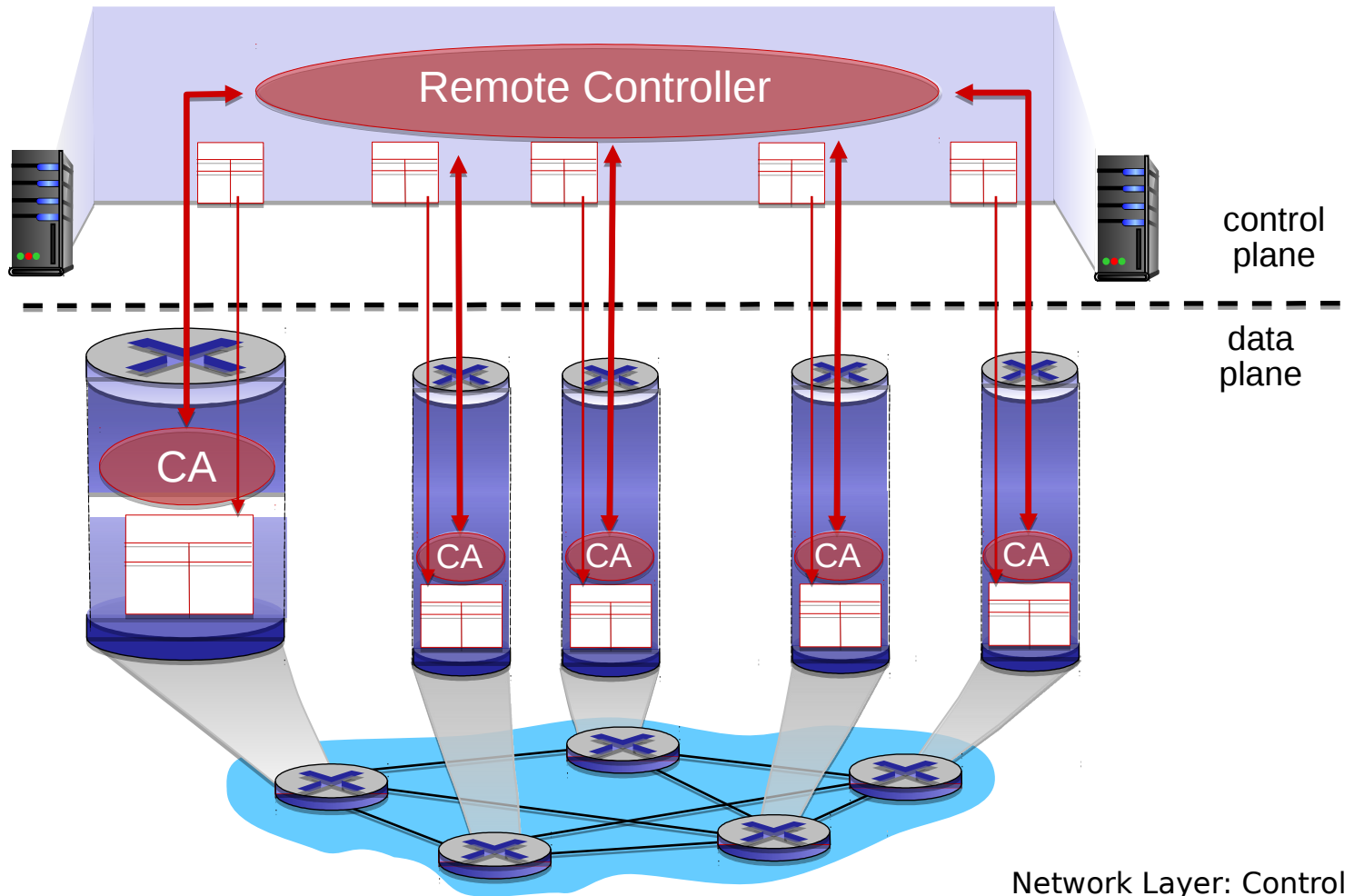
Per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

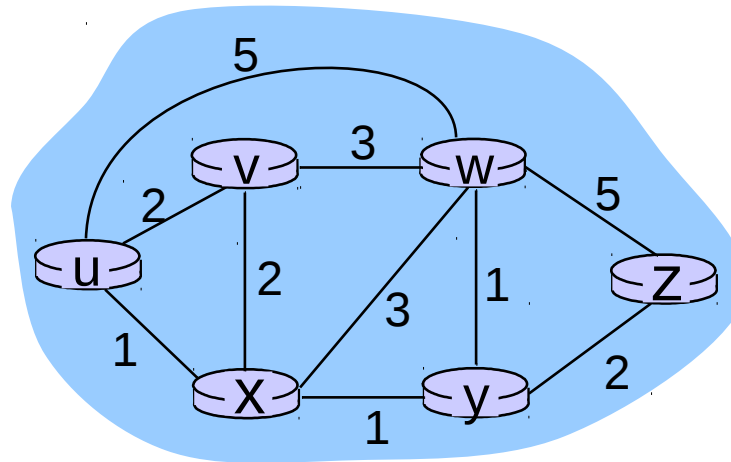
5.7 Network management and SNMP

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 will traverse in going from given initial source host to given final destination host
- “good”: least “cost”, “fastest”, “least congested”

Graph abstraction of the network



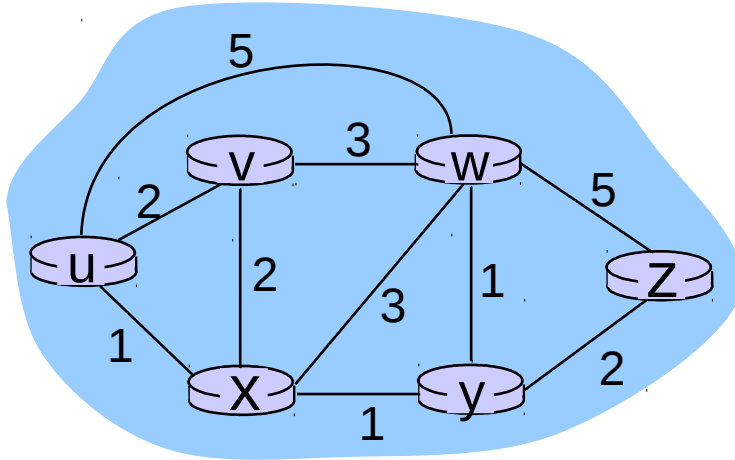
graph: $G = (N,E)$

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

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

aside: graph abstraction is useful in other network contexts, e.g., P2P, where N is set of peers and E is set of TCP connections

Graph abstraction: costs



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

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

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

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

Q: global or decentralized information?

global:

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

decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

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

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

A link-state routing algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
 - accomplished via “link state broadcast”
 - all nodes have same info
- computes least cost paths from one node (“source”) to all other nodes
 - gives *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.’s

notation:

- $c(x,y)$: link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: current value of cost of path from source to dest. v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least cost path definitively known

Dijkstra's algorithm

1 **Initialization:**

2 $N' = \{u\}$

3 for all nodes v

4 if v adjacent to u

5 then $D(v) = c(u,v)$

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 cost to v is either old cost to v or known

14 shortest path cost to w plus cost from w to v */

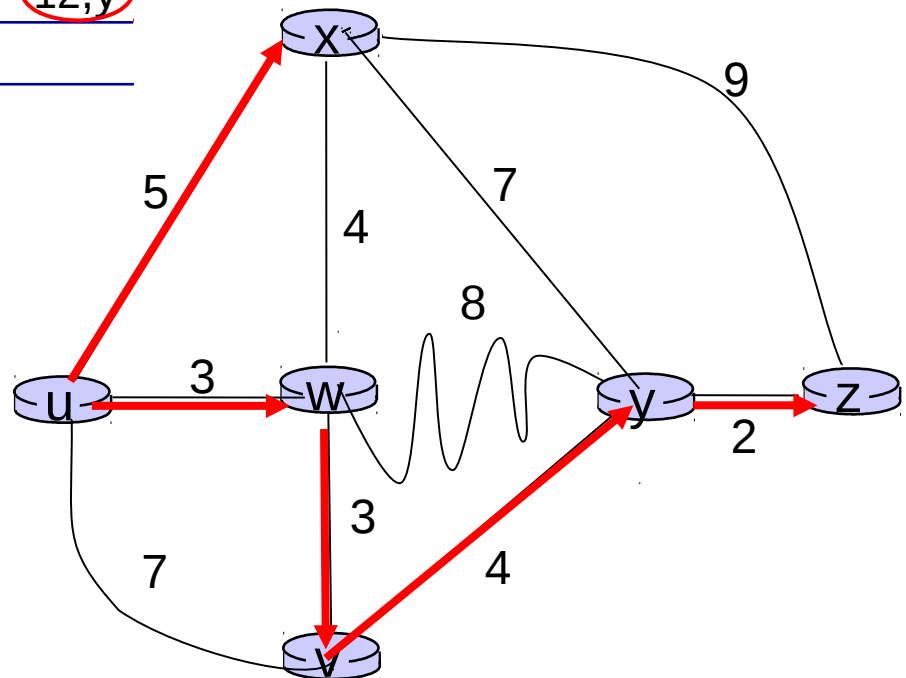
15 **until all nodes in N'**

Dijkstra's algorithm: example

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

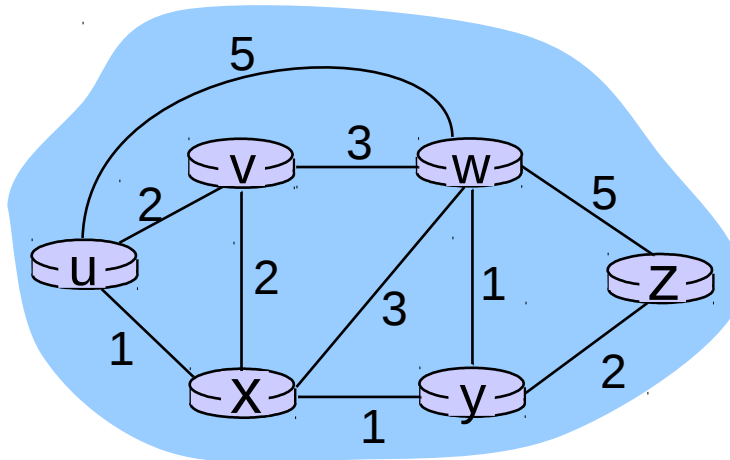
notes:

- ❖ construct shortest path tree by tracing predecessor nodes
- ❖ ties can exist (can be broken arbitrarily)



Dijkstra's algorithm: another example

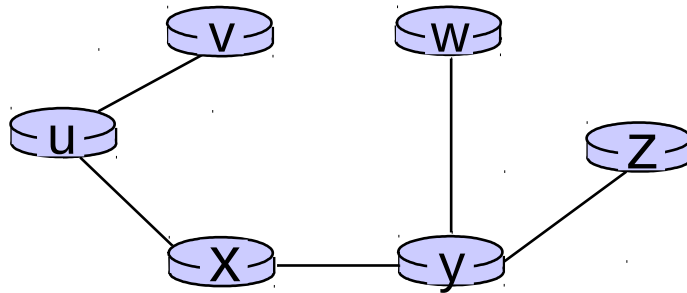
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



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

Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:



resulting forwarding table in u:

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

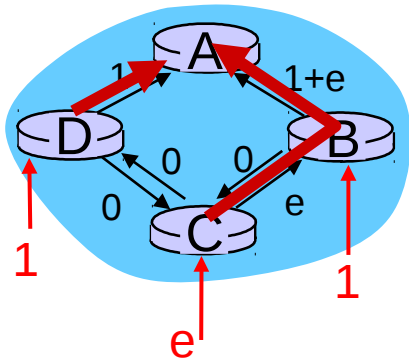
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

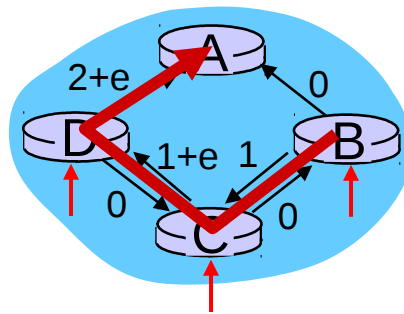
- each iteration: need to check all nodes, w, not in N
- $n(n+1)/2$ comparisons: $O(n^2)$
- more efficient implementations possible: $O(|E| + |V|\log|V|)$

oscillations possible:

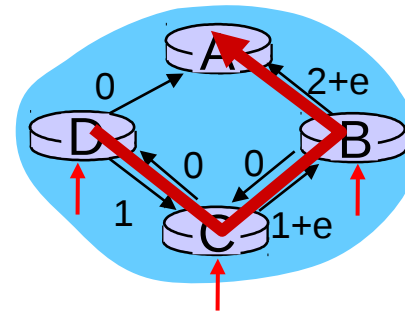
- e.g., support link cost equals amount of carried traffic:



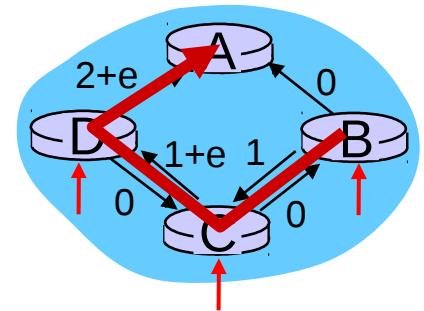
initially



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

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

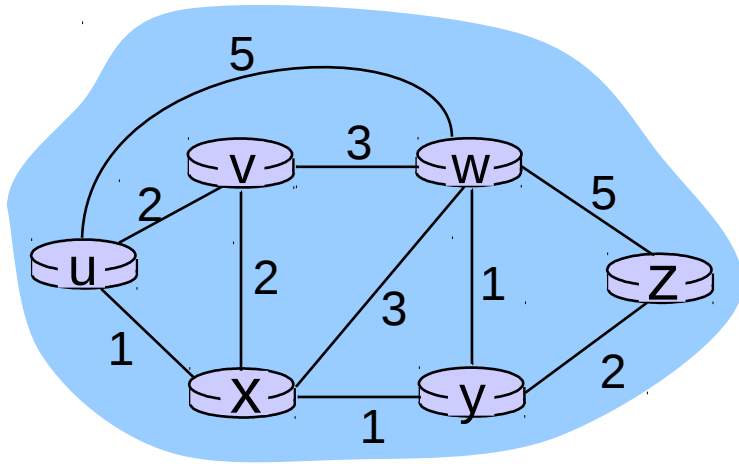
$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example



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

B-F equation says:

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

node achieving minimum is next
hop in shortest path, used in forwarding table

Distance vector algorithm

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

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:
$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$
- ❖ under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

Distance vector algorithm

iterative,

asynchronous:

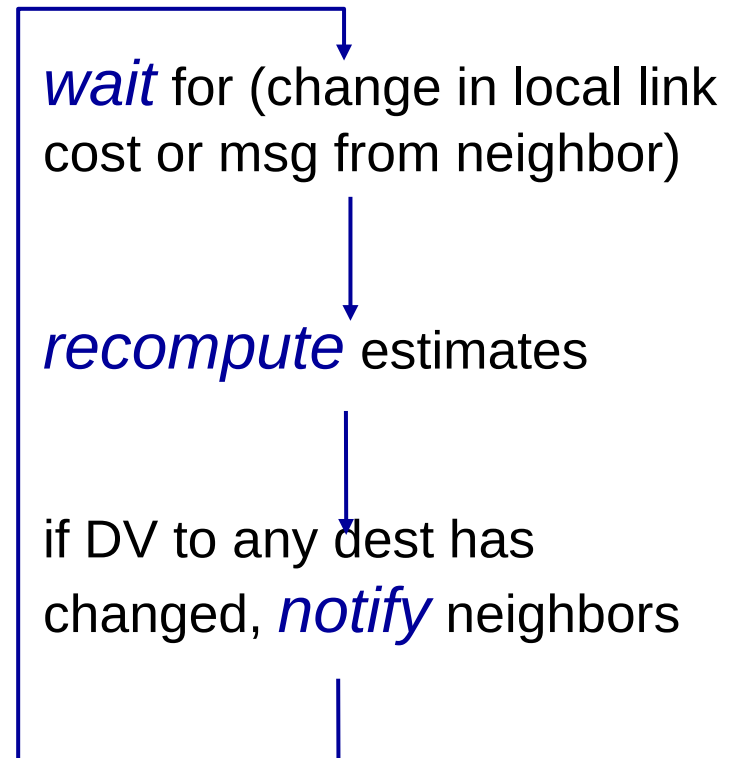
each local iteration
caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
 - neighbors then notify their neighbors if necessary

each node:



$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

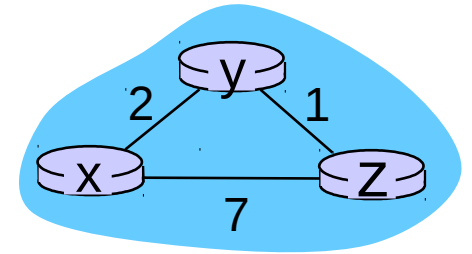
		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

node y table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

node z table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0



time →

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

node y table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

node z table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

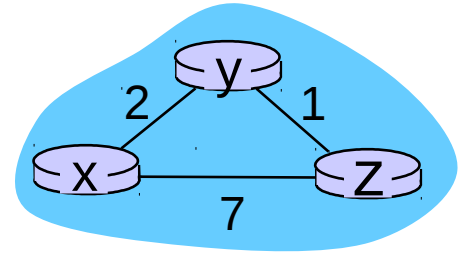
		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

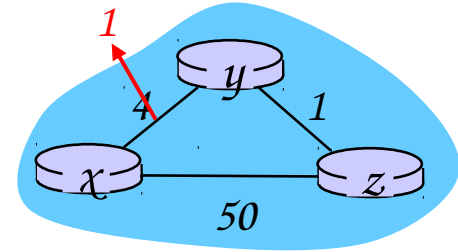


time →

Distance vector: link cost changes

link cost changes:

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



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.

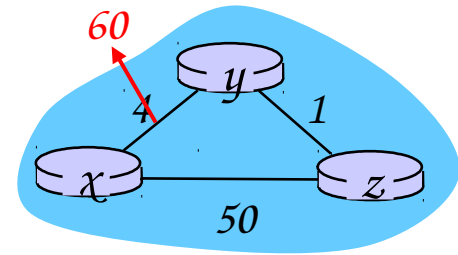
t_2 : y receives z 's update, updates its distance table. y 's least costs do *not* change, so y does *not* send a message to z .

“good news travels fast”

Distance vector: link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ *bad news travels slow* - “count to infinity” problem!
- ❖ 44 iterations before algorithm stabilizes: see text



poisoned reverse:

- ❖ If Z routes through Y to get to X :
 - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- ❖ will this completely solve count to infinity problem?

Comparison of LS and DV algorithms

message complexity

- **LS:** with n nodes, E links, $O(nE)$ msgs sent
- **DV:** exchange between neighbors only
 - convergence time varies

speed of convergence

- **LS:** $O(n^2)$ algorithm requires $O(nE)$ msgs
 - may have oscillations
- **DV:** convergence time varies
 - may be routing loops
 - count-to-infinity problem

robustness: what happens if router malfunctions?

LS:

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

DV:

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

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

Making routing scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

... *not* true in practice

scale: with billions of destinations:

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

administrative autonomy

- internet = 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”)

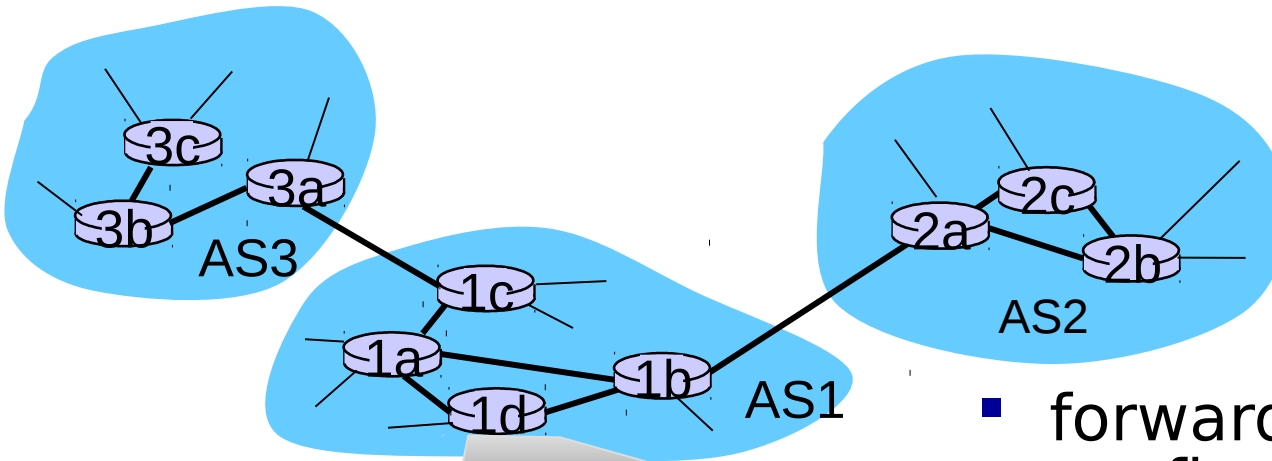
intra-AS routing

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

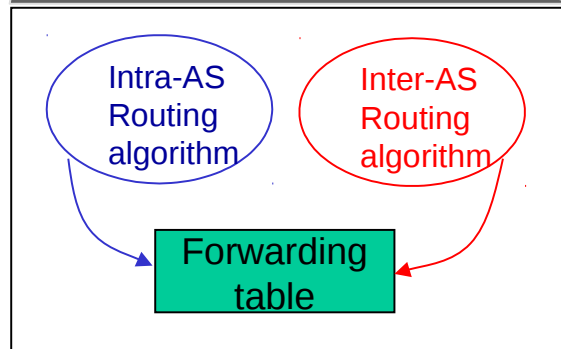
inter-AS routing

- routing among AS'es
- gateways perform inter-domain routing (as well as intra-domain routing)

Interconnected ASes



- forwarding table configured by both intra- and inter-AS routing algorithm
 - intra-AS routing determine entries for destinations within AS
 - inter-AS & intra-AS determine entries for external destinations



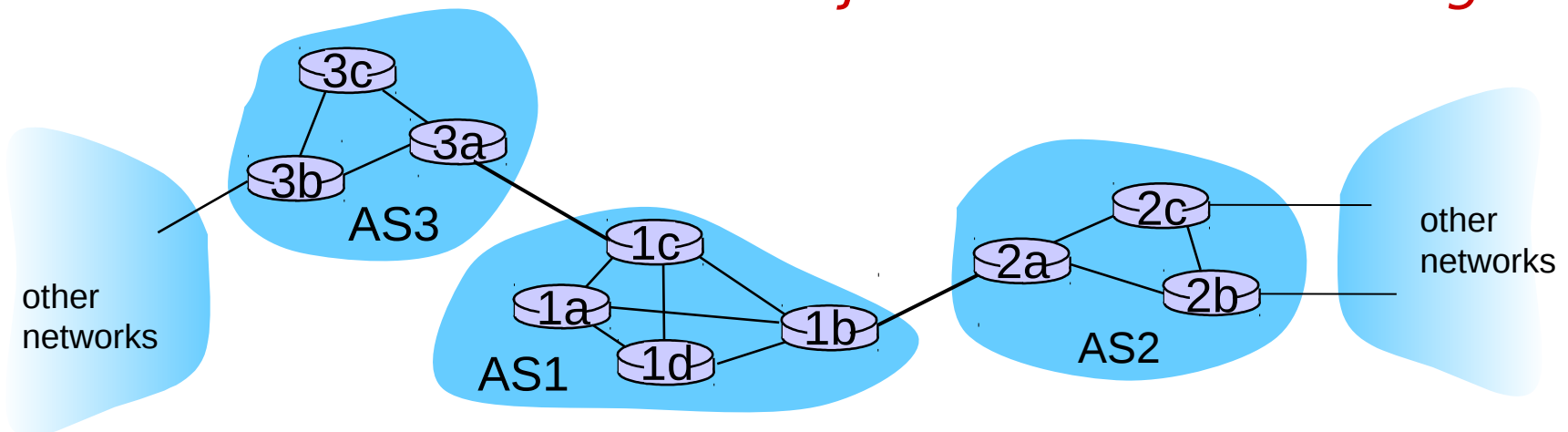
Inter-AS tasks

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

AS1 must:

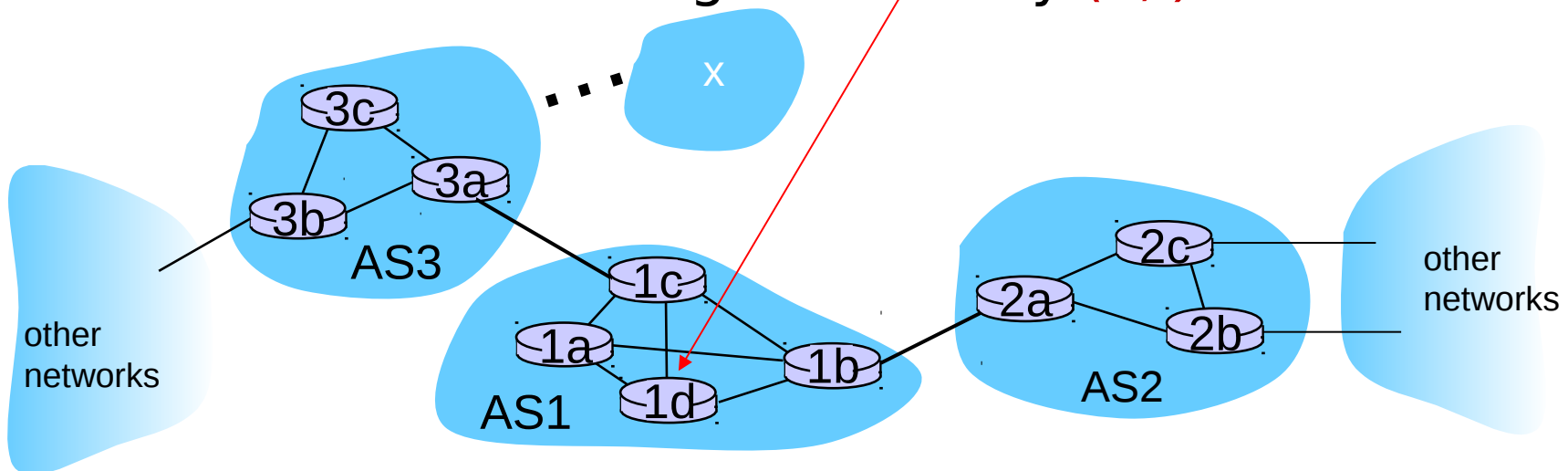
1. learn which destds are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



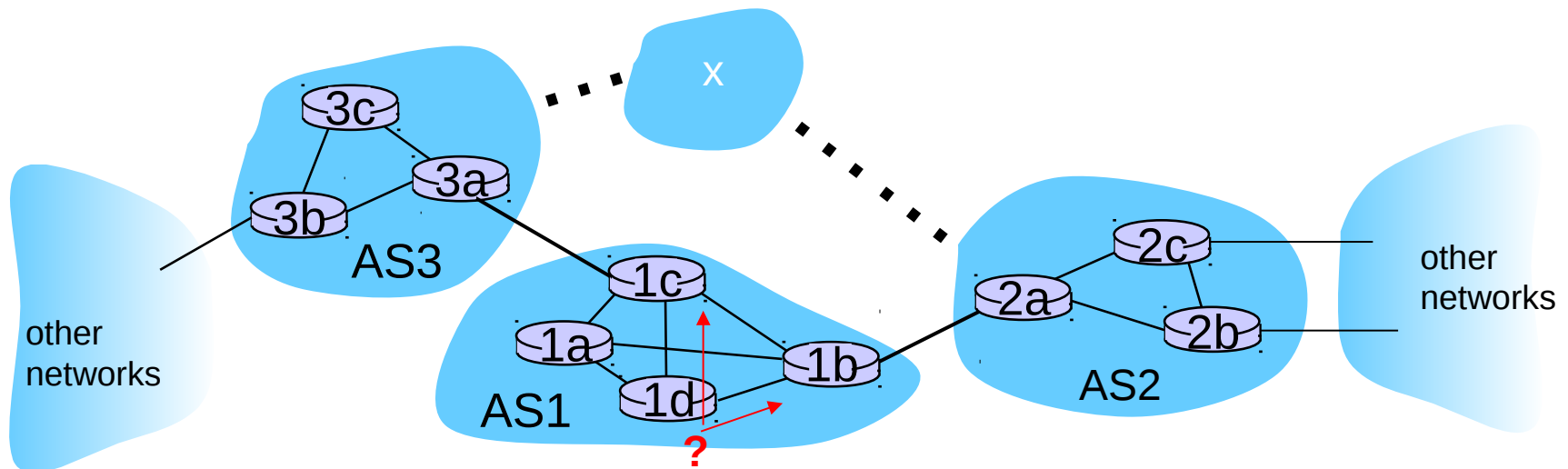
Example: setting forwarding table in router 1d

- ❖ suppose AS1 learns (via inter-AS protocol) that subnet x reachable via AS3 (gateway 1c), but not via AS2
 - inter-AS protocol propagates reachability info to all internal routers
- ❖ router 1d determines from intra-AS routing info that its interface l is on the least cost path to 1c
 - installs forwarding table entry (x, l)



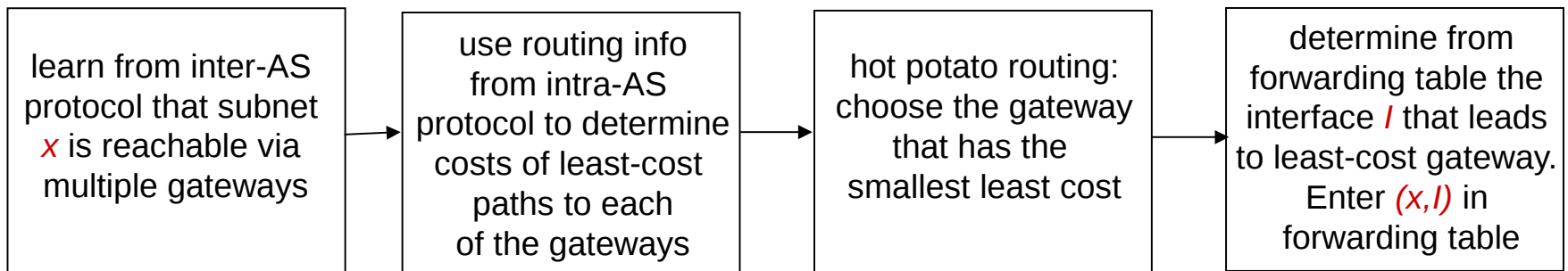
Example: choosing among multiple ASes

- ❖ now suppose AS1 learns from inter-AS protocol that subnet x is reachable from AS3 *and* from AS2.
- ❖ to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest x
 - this is also job of inter-AS routing protocol!



Example: choosing among multiple ASes

- ❖ now suppose AS1 learns from inter-AS protocol that subnet x is reachable from AS3 *and* from AS2.
- ❖ to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest x
 - this is also job of inter-AS routing protocol!
- ❖ *hot potato routing: send* packet towards closest of two routers.

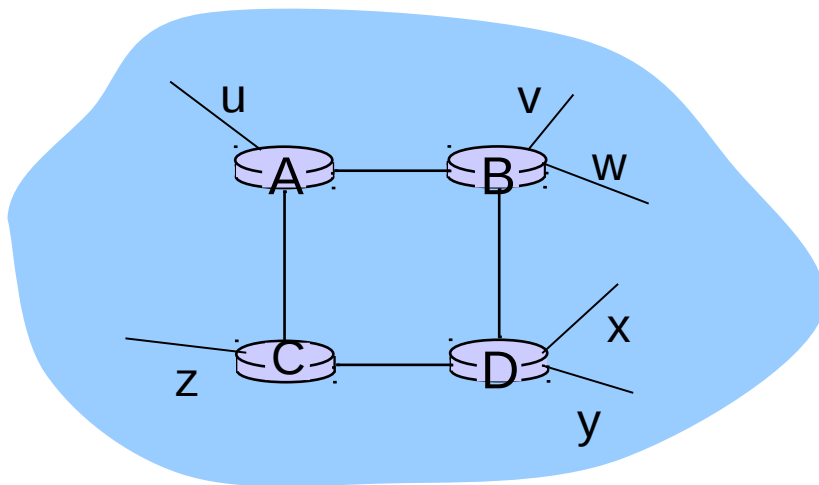


Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
 - RIP: Routing Information Protocol **(DV)**
 - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF) **(LS)**
 - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

RIP (Routing Information Protocol)

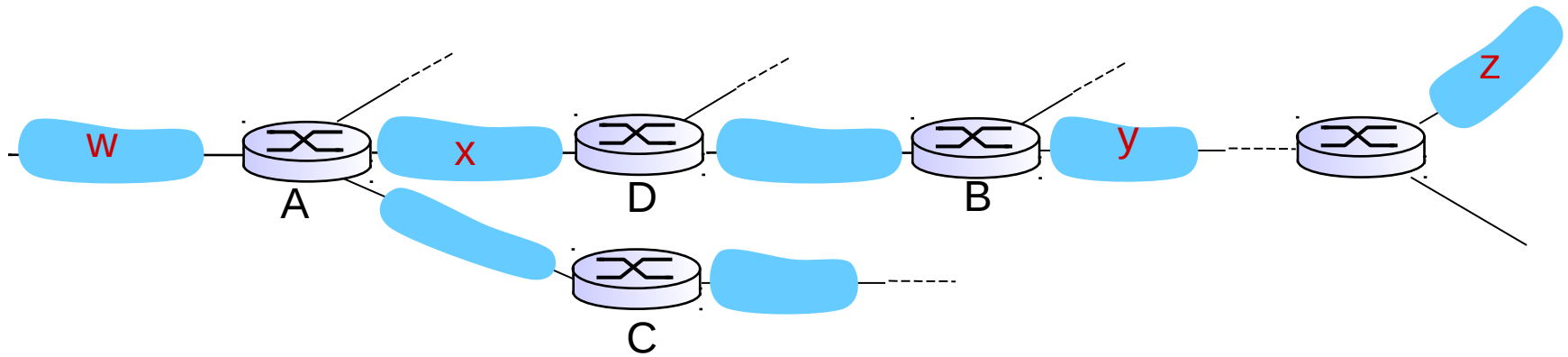
- ❖ included in BSD-UNIX distribution in 1982
- ❖ distance vector algorithm
 - distance metric: # hops (max = 15 hops), each link has cost 1
 - DVs exchanged with neighbors every 30 sec in response message (aka **advertisement**)
 - each advertisement: list of up to 25 destination **subnets** (in IP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



from router A to destination **subnets**:

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

RIP: example



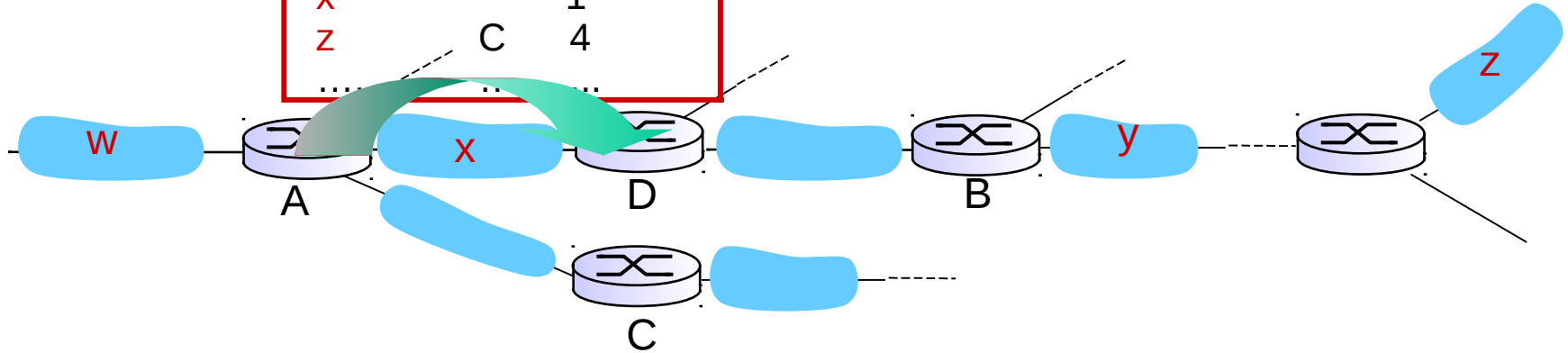
routing table in router D

destination subnet	next router	# hops to dest
W	A	2
Y	B	2
Z	B	7
X	--	1
....

RIP: example

A-to-D advertisement

dest	next hops	hops
W	-	1
X	-	1
Z	C	4
....



routing table in router D

destination subnet	next router	# hops to dest
W	A	2
Y	B	2
Z	B → A	7 → 5
X	--	1
....

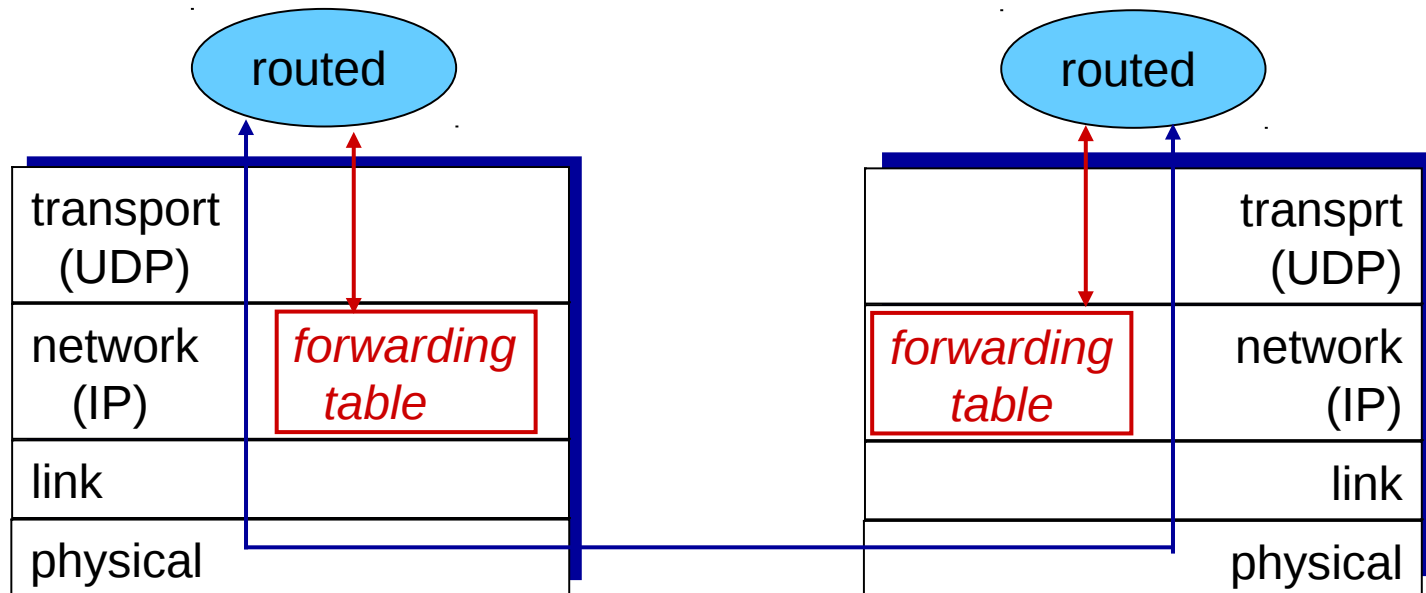
RIP: link failure, recovery

if no advertisement heard after 180 sec -->
neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly (?) propagates to entire net
- *poison reverse* used to prevent ping-pong loops (infinite distance = 16 hops)

RIP table processing

- ❖ RIP routing tables managed by *application-level* process called route-d (daemon)
- ❖ advertisements sent in UDP packets, periodically repeated



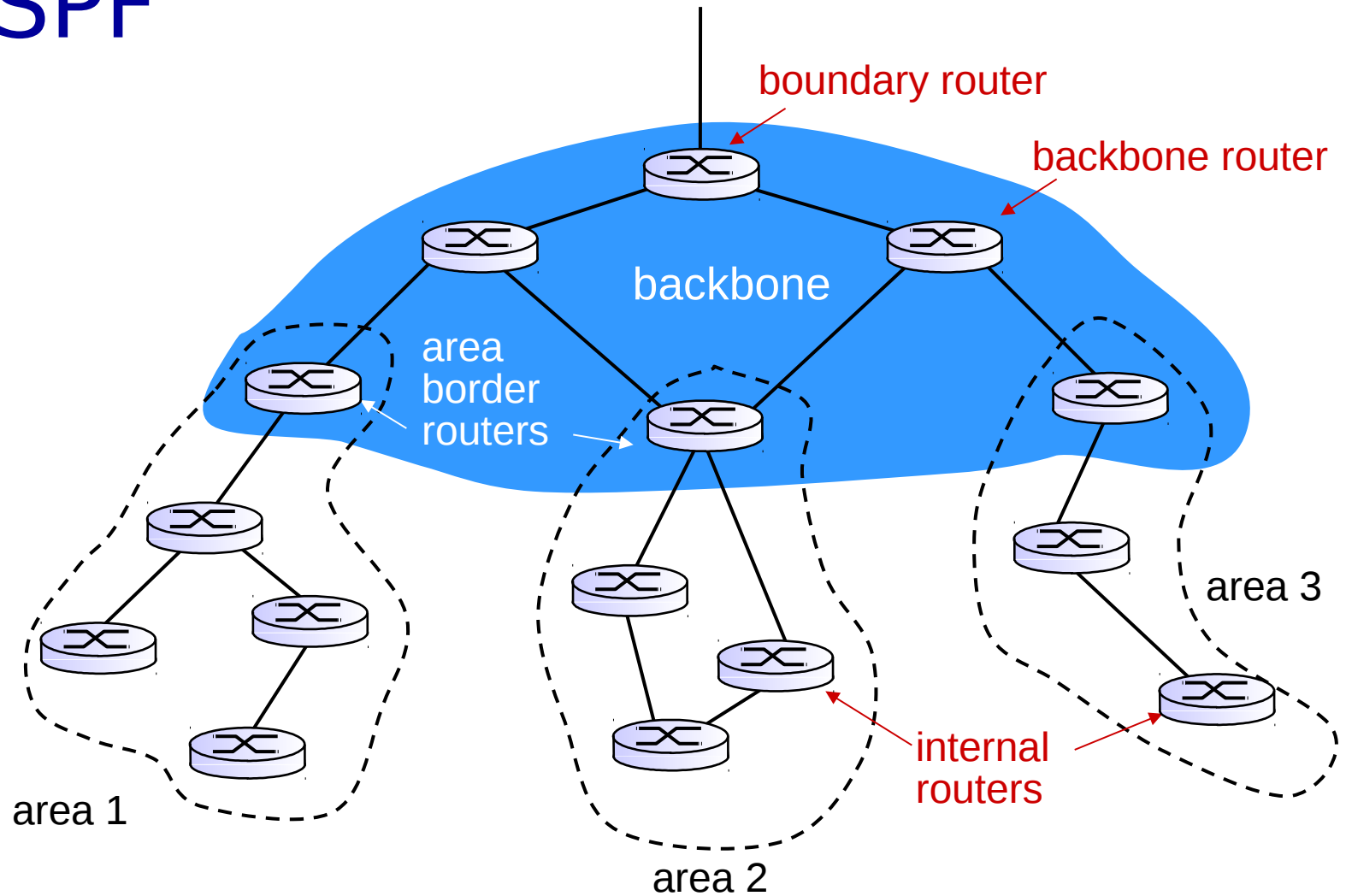
OSPF (Open Shortest Path First)

- “open”: publicly available
- uses link-state algorithm
 - link state packet dissemination
 - topology map at each node
 - route computation using Dijkstra’s algorithm
- router floods OSPF link-state advertisements to all other routers in *entire* AS
 - carried in OSPF messages directly over IP (rather than TCP or UDP)
 - link state: for each attached link
- *IS-IS routing* protocol: nearly identical to OSPF

OSPF “advanced” features

- *security*: all OSPF messages authenticated (to prevent malicious intrusion)
- **multiple** same-cost **paths** allowed (only one path in RIP)
- for each link, multiple cost metrics for different **TOS** (e.g., satellite link cost set low for best effort ToS; high for real-time ToS)
- integrated uni- and **multi-cast** support:
 - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- **hierarchical** OSPF in large domains.

Hierarchical OSPF



Hierarchical OSPF

- *two-level hierarchy*: local area, backbone.
 - link-state advertisements only in area
 - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- *area border routers*: “summarize” distances to nets in own area, advertise to other Area Border routers.
- *backbone routers*: run OSPF routing limited to backbone.
- *boundary routers*: connect to other AS'es.

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

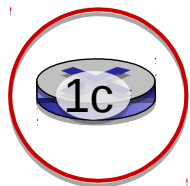
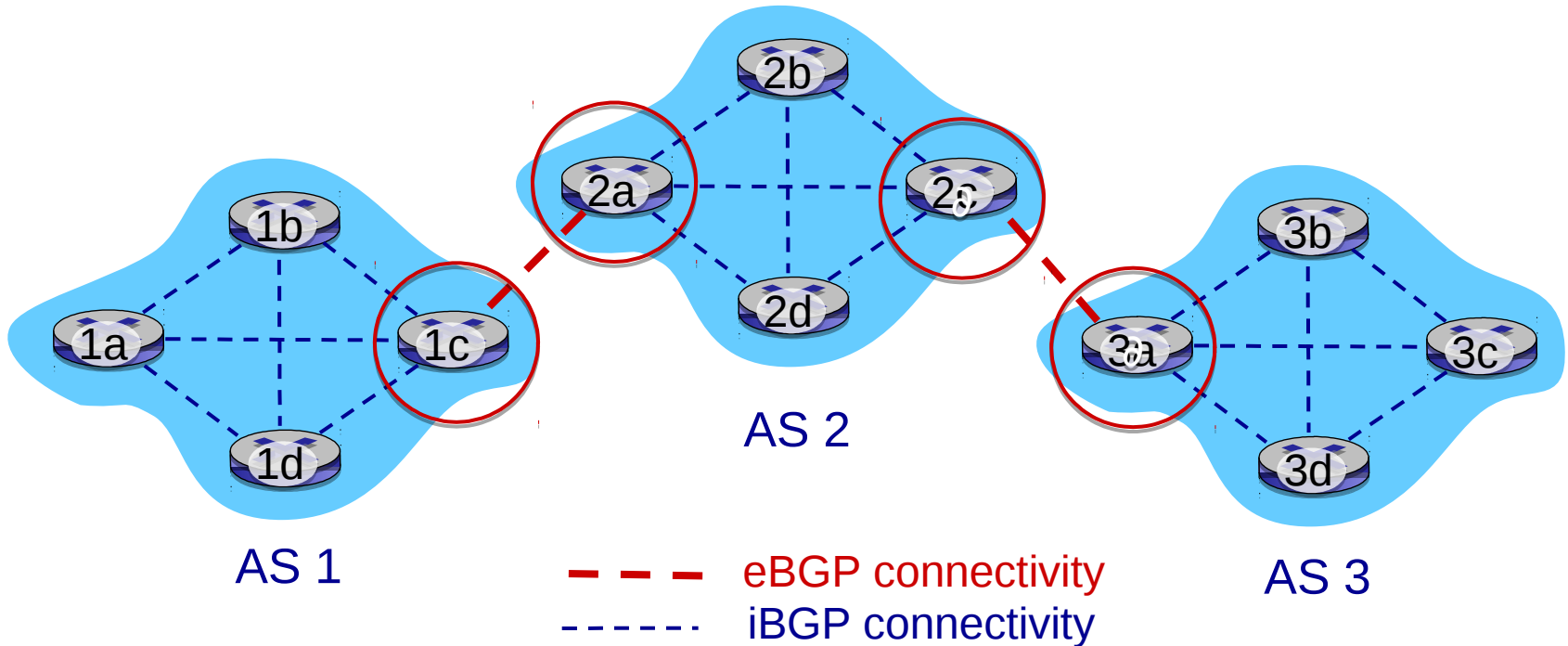
5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

Internet inter-AS routing: BGP

- **BGP (Border Gateway Protocol):** *the* de facto inter-domain routing protocol
 - “glue that holds the Internet together”
- 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*
- allows subnet to advertise its existence to rest of Internet: *“I am here”*

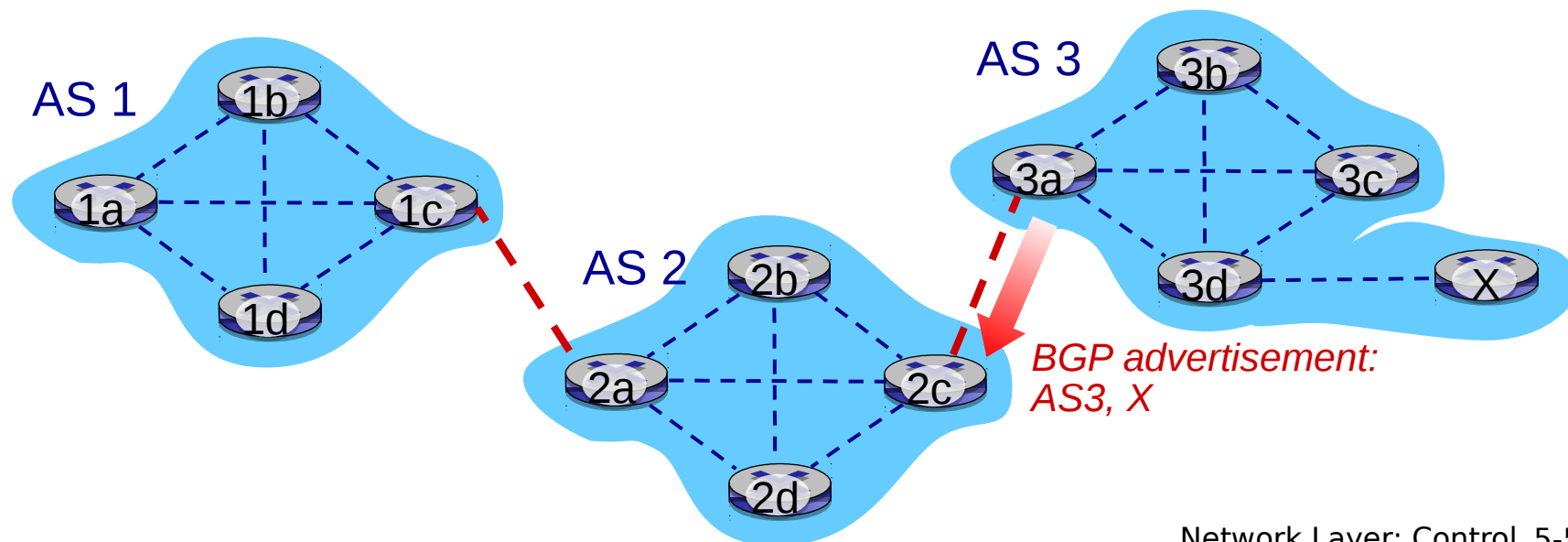
eBGP, iBGP connections



gateway routers run both eBGP and iBGP protocols

BGP basics

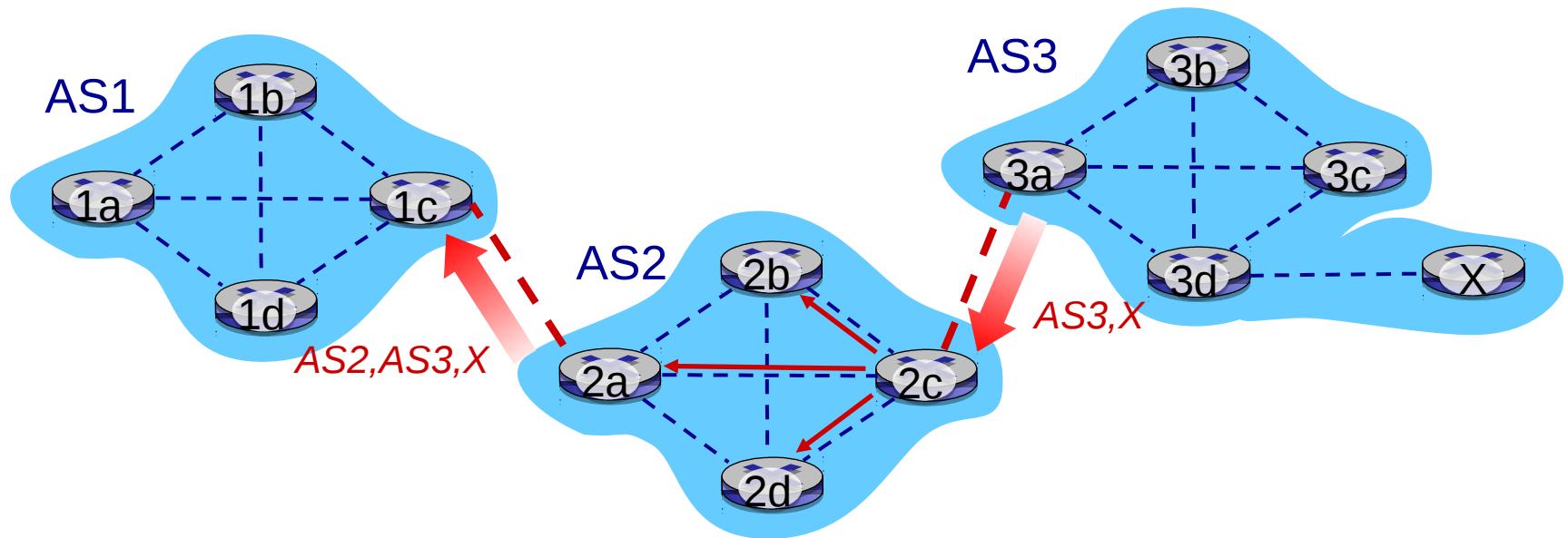
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
 - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway router 3a advertises path **AS3,X** to AS2 gateway router 2c:
 - AS3 *promises*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



Path attributes and BGP routes

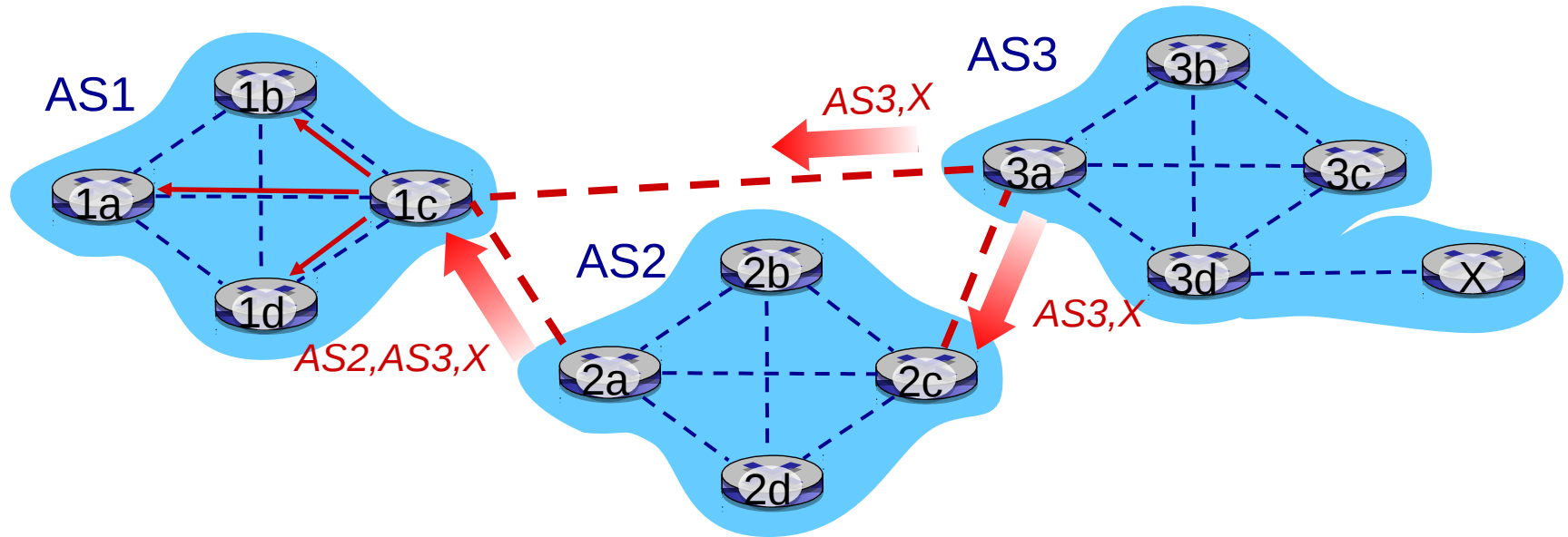
- advertised prefix includes BGP attributes
 - prefix + attributes = “route”
- two important attributes:
 - **AS-PATH**: list of ASes through which prefix advertisement has passed
 - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
- ***Policy-based routing***:
 - gateway receiving 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



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

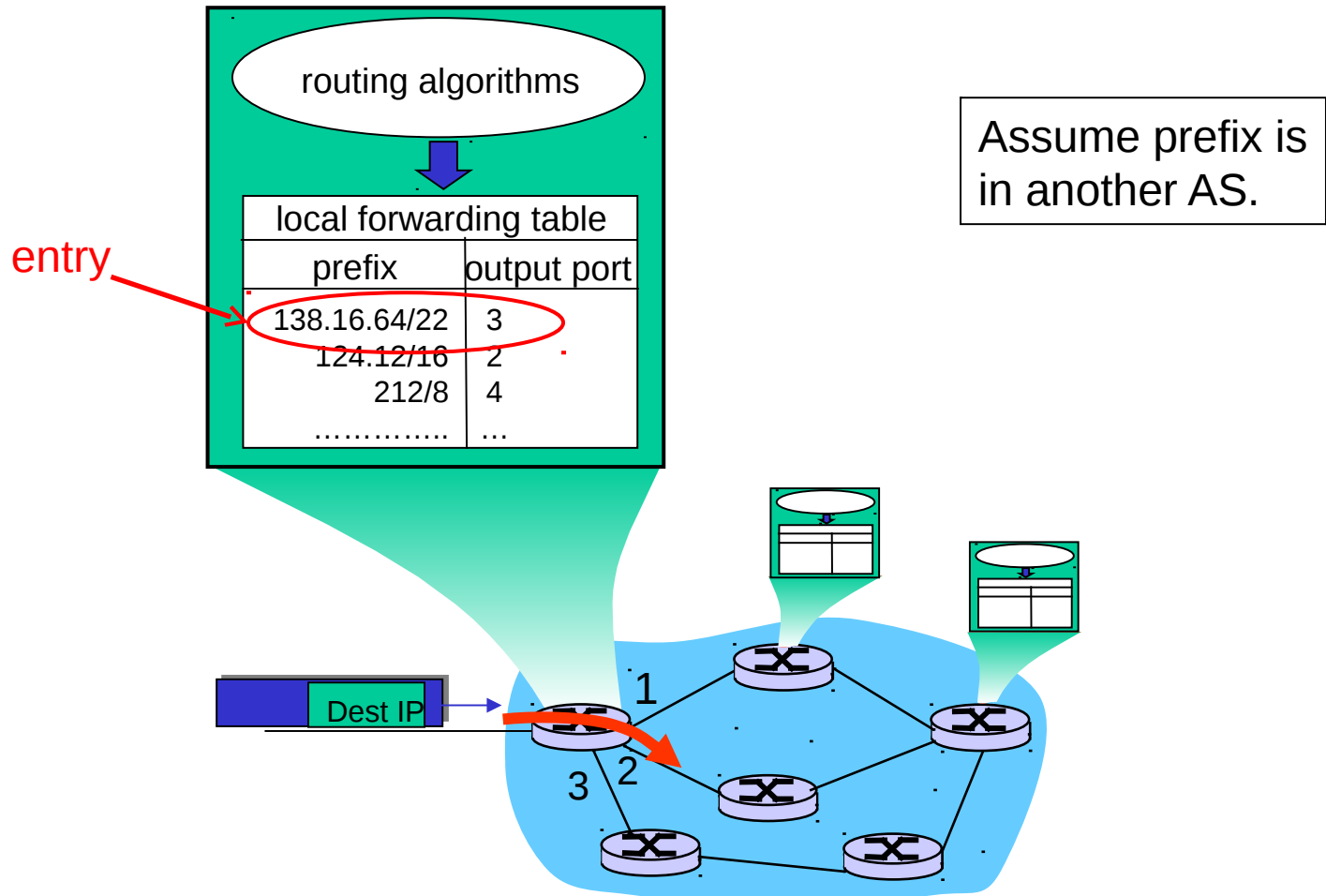
BGP messages

- 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

Putting it Altogether: *How Does an Entry Get Into a Router's Forwarding Table?*

- ❖ Answer is complicated!
- ❖ Ties together hierarchical routing with BGP and OSPF.
- ❖ Provides nice overview of BGP!

How does entry get in forwarding table?

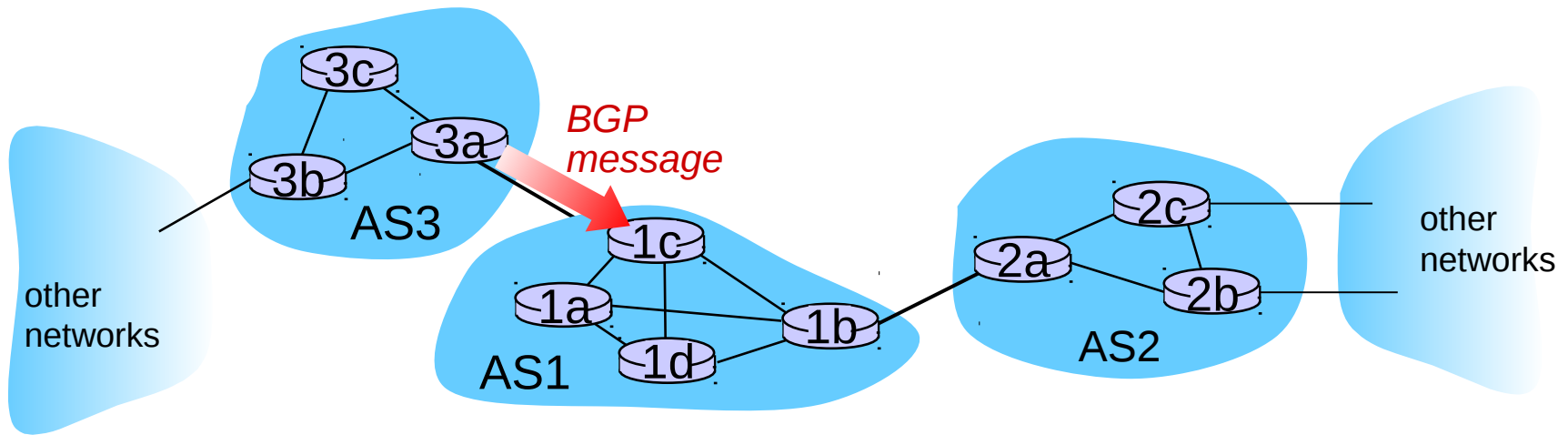


How does entry get in forwarding table?

High-level overview

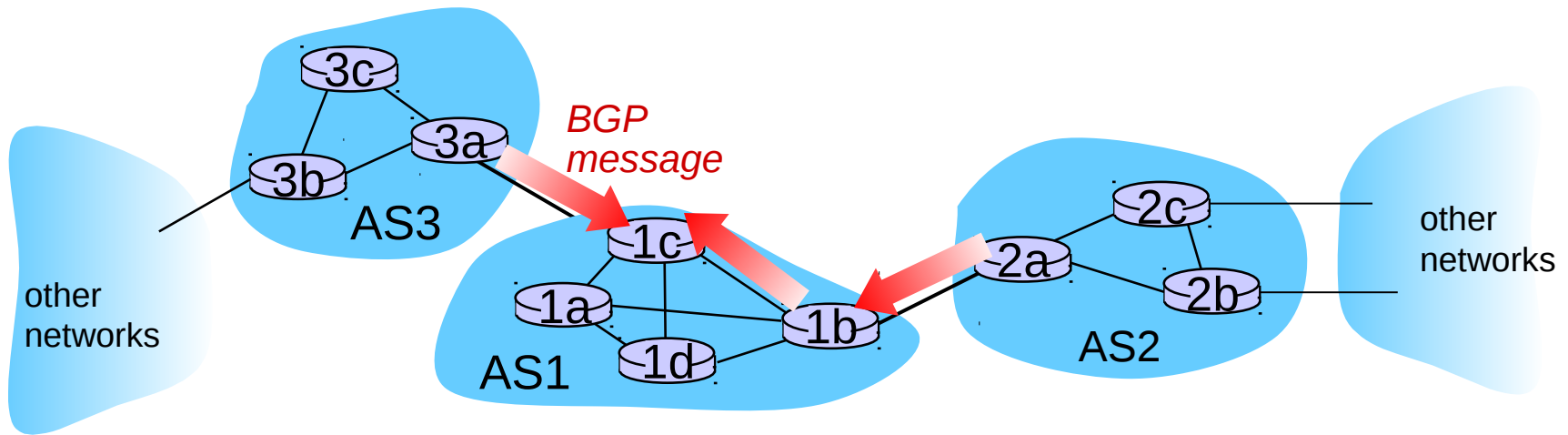
1. Router becomes aware of prefix
2. Router determines output port for prefix
3. Router enters prefix-port in forwarding table

Router becomes aware of prefix



- ❖ BGP message contains “routes”
- ❖ “route” is a prefix and attributes: AS-PATH, NEXT-HOP,...
- ❖ Example: route:
 - ❖ Prefix:138.16.64/22 ; AS-PATH: AS3 AS131 ; NEXT-HOP: 201.44.13.125

Router may receive multiple routes



- ❖ Router may receive multiple routes for same prefix
- ❖ Has to select one route

Select best BGP route to prefix

- ❖ Router selects route based on shortest AS-PATH

- ❖ Example:

- ❖ AS2 AS17 to 138.16.64/22

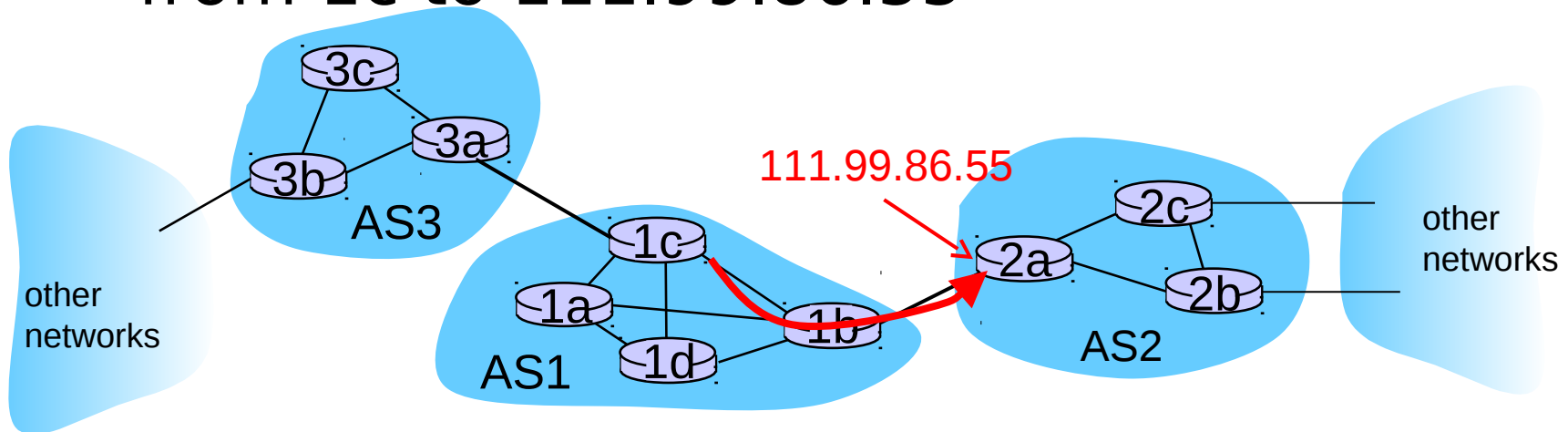
- ❖ AS3 AS131 AS201 to 138.16.64/22

select

- ❖ What if there is a tie? We'll come back to that!

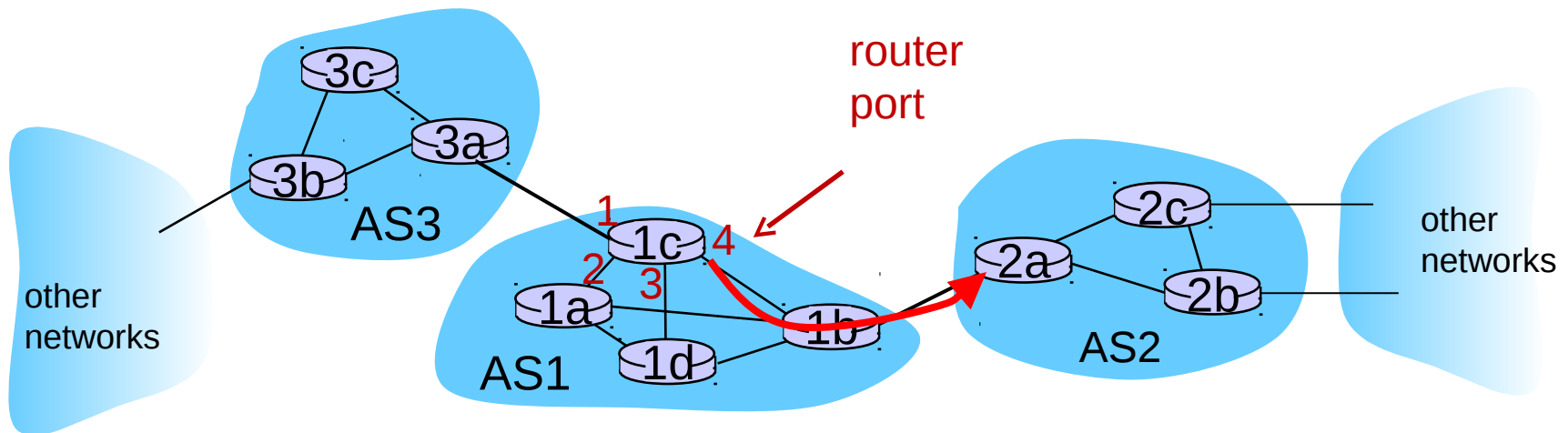
Find best intra-route to BGP route

- ❖ Use selected route's NEXT-HOP attribute
 - Route's NEXT-HOP attribute is the IP address of the router interface that begins the AS PATH.
- ❖ Example:
 - ❖ AS-PATH: AS2 AS17 ; NEXT-HOP: 111.99.86.55
- ❖ Router uses OSPF to find shortest path from 1c to 111.99.86.55



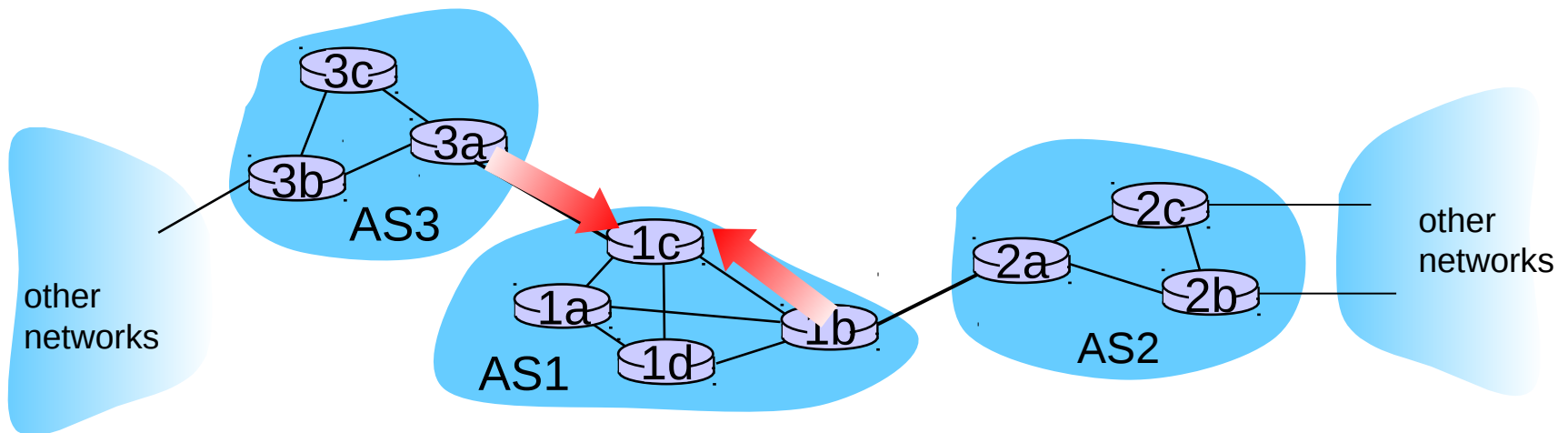
Router identifies port for route

- ❖ Identifies port along the OSPF shortest path
- ❖ Adds prefix-port entry to its forwarding table:
 - (138.16.64/22 , port 4)



Hot Potato Routing

- ❖ Suppose there two or more best inter-routes.
- ❖ Then choose route with closest NEXT-HOP
 - Use OSPF to determine which gateway is closest
 - Q: From 1c, chose AS3 AS131 or AS2 AS17?
 - A: route AS3 AS201 since it is closer

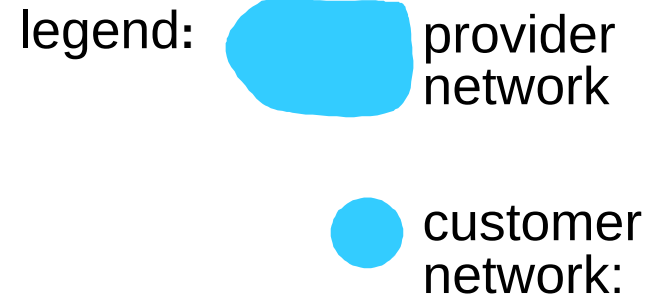
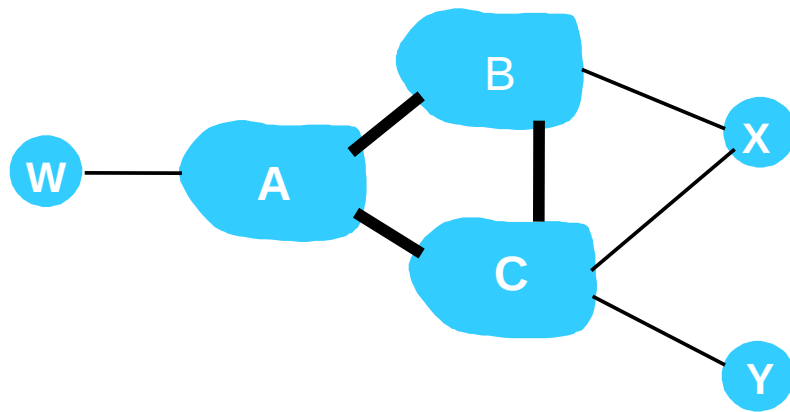


How does entry get in forwarding table?

Summary

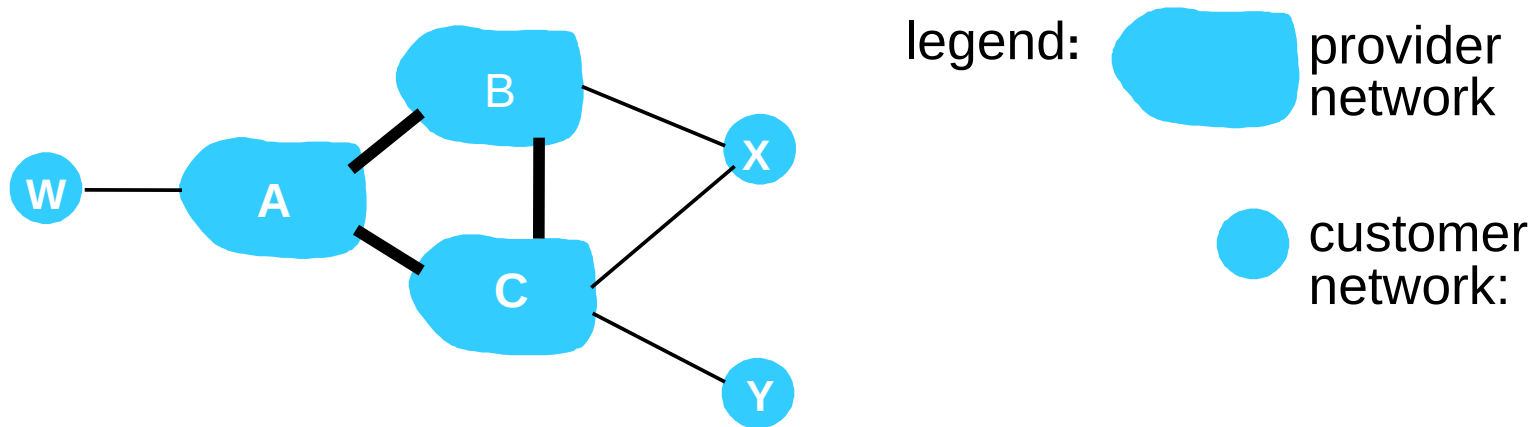
1. Router becomes aware of prefix
 - via BGP route advertisements from other routers
2. Determine router output port for prefix
 - Use BGP route selection to find best inter-AS route
 - Use OSPF to find best intra-AS route leading to best inter-AS route
 - Router identifies router port for that best route
3. Enter prefix-port entry in forwarding table

BGP routing policy



- ❖ A,B,C are *provider networks*
- ❖ X,W,Y are customer (of provider networks)
- ❖ X is *dual-homed*: attached to two networks
 - X does not want to route from B via X to C
 - .. so X will not advertise to B a route to C

BGP routing policy (2)



- ❖ A advertises path AW to B
- ❖ B advertises path BAW to X
- ❖ Should B advertise path BAW to C?
 - No way! B gets no “revenue” for routing CBAW since neither W nor C are B’s customers
 - B wants to force C to route to w via A
 - B wants to route *only* to/from its customers!

Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its net.
- intra-AS: single admin, so no policy decisions needed

scale:

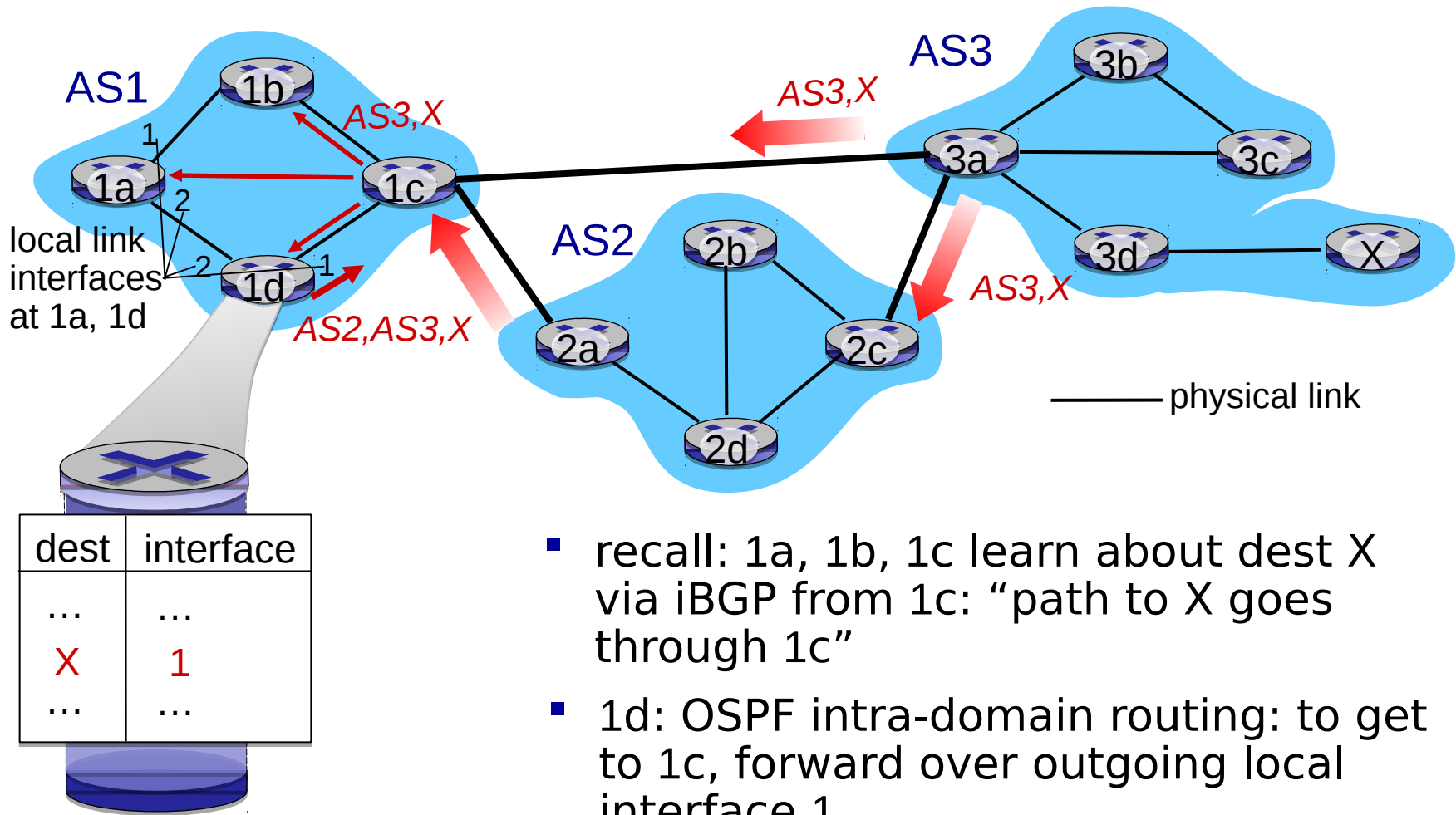
- hierarchical routing saves table size, reduced update traffic

performance:

- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance

BGP, OSPF, forwarding table entries

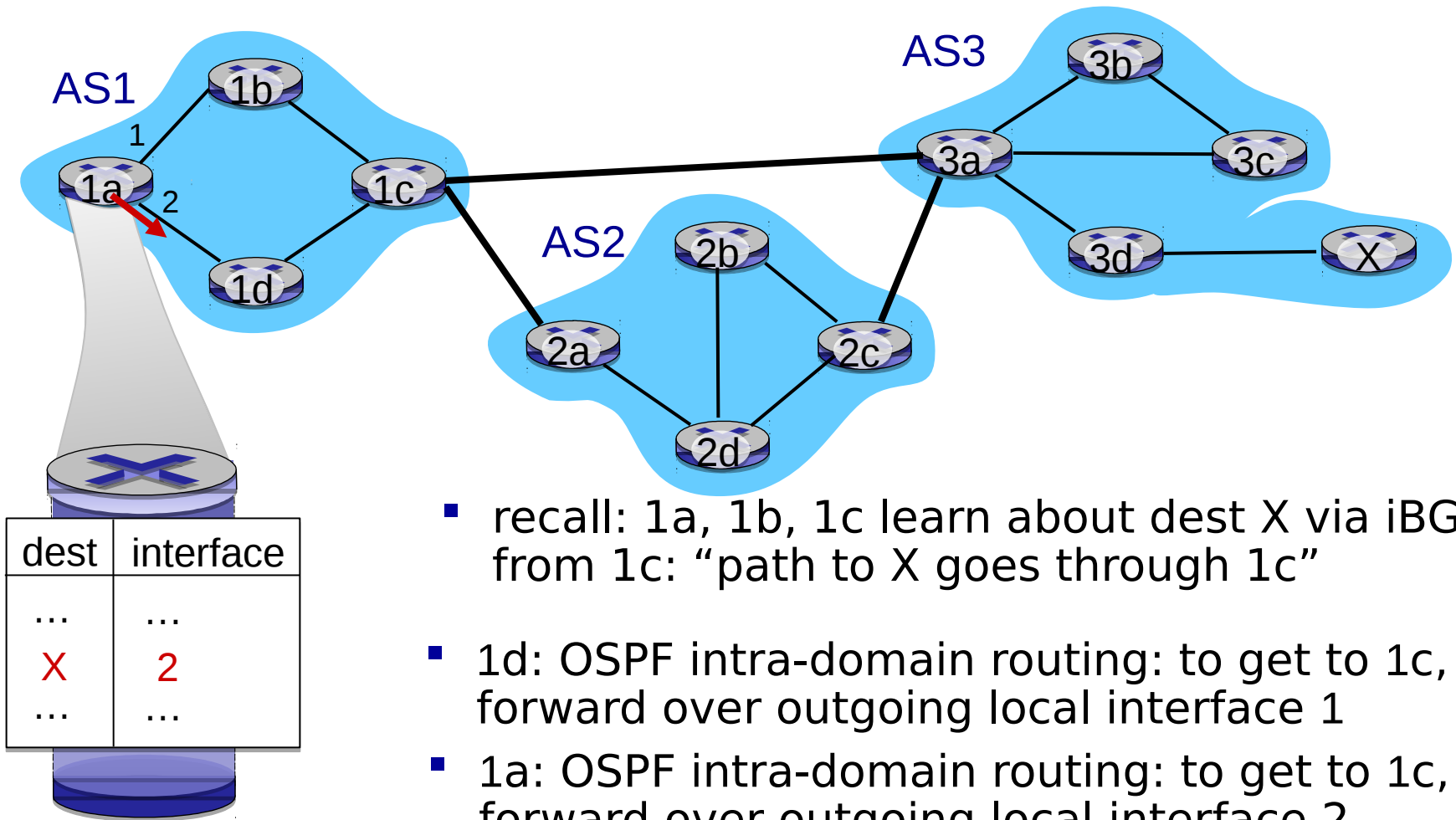
Q: how does router set forwarding table entry to distant prefix?



- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

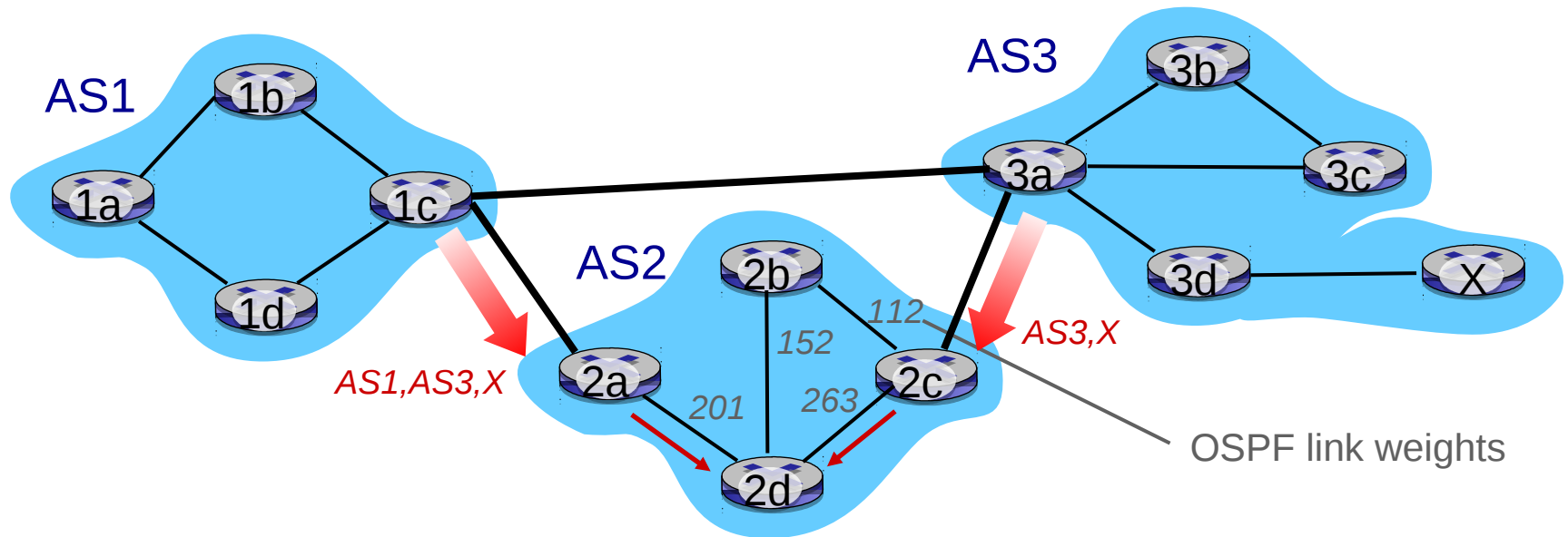


- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

BGP route selection

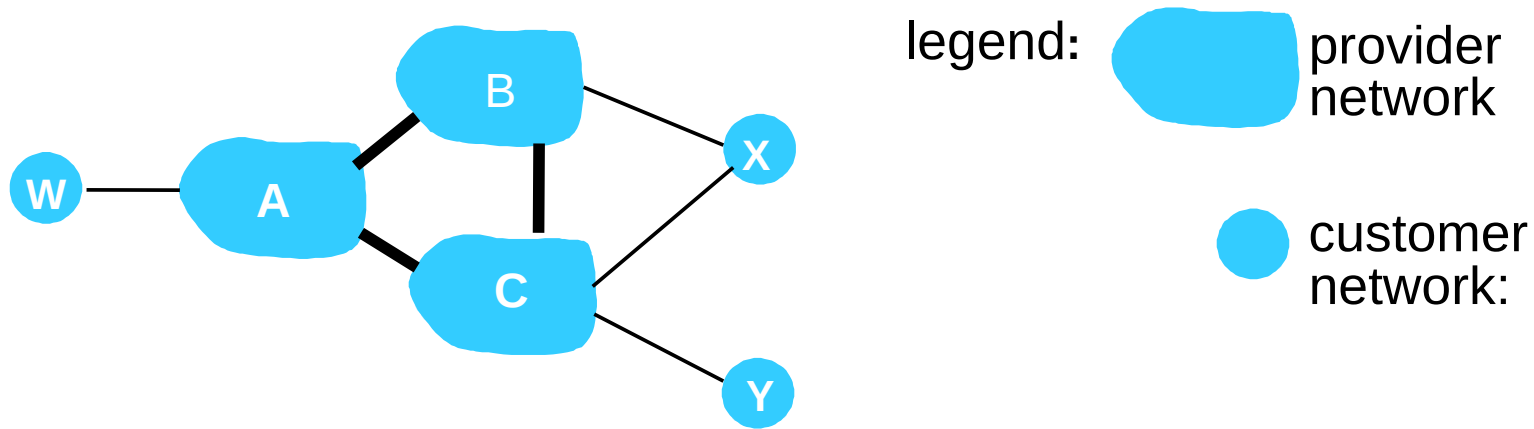
- router may learn about more than one route to destination AS, selects route based on:
 1. local preference value attribute: policy decision
 2. shortest AS-PATH
 3. closest NEXT-HOP router: hot potato routing
 4. additional criteria

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!

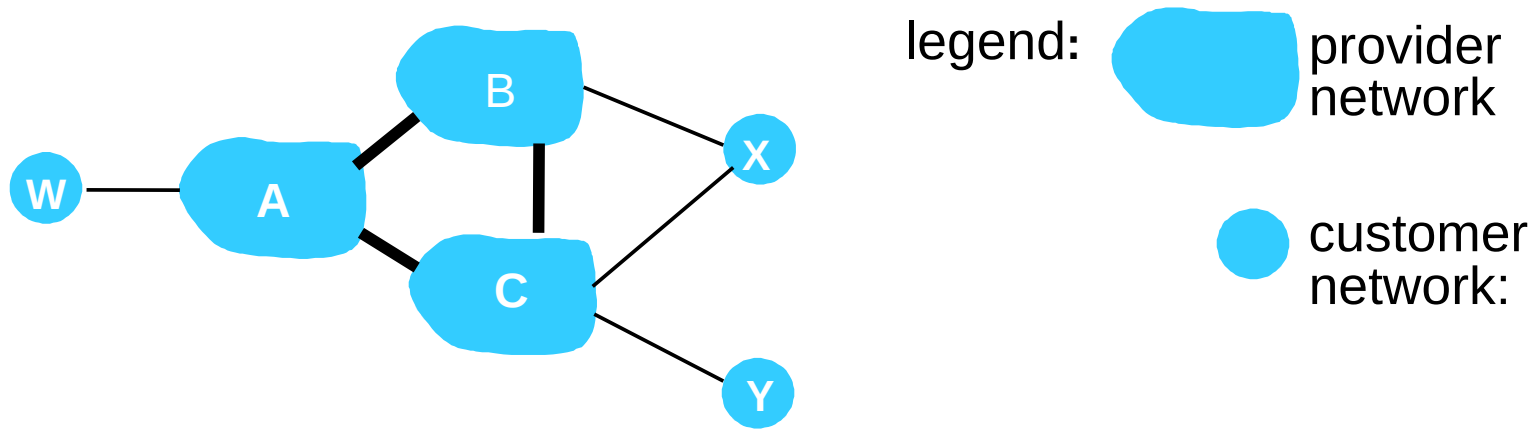
BGP: achieving policy via advertisements



Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

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

BGP: achieving policy via advertisements



Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- 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

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

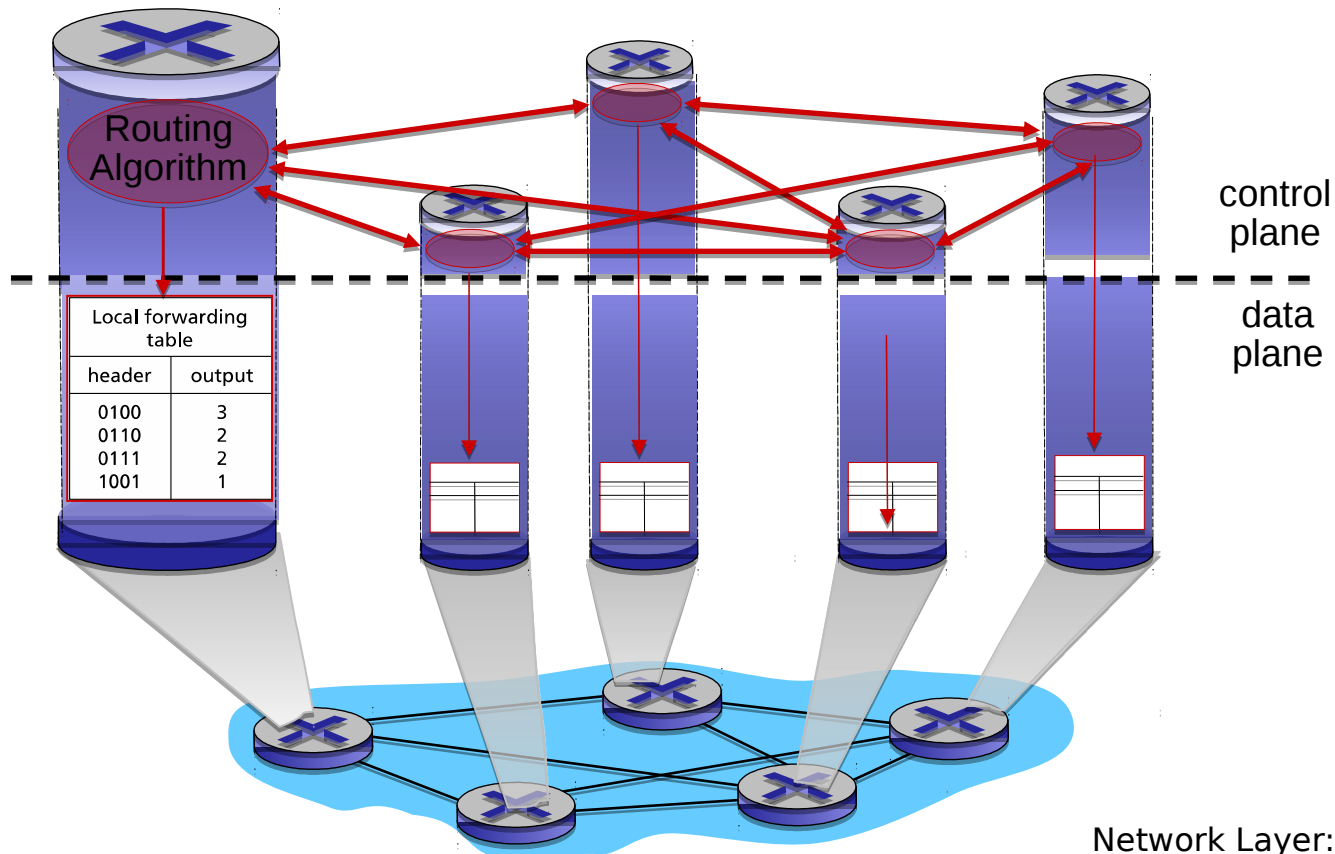
5.7 Network management and SNMP

Software defined networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router 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

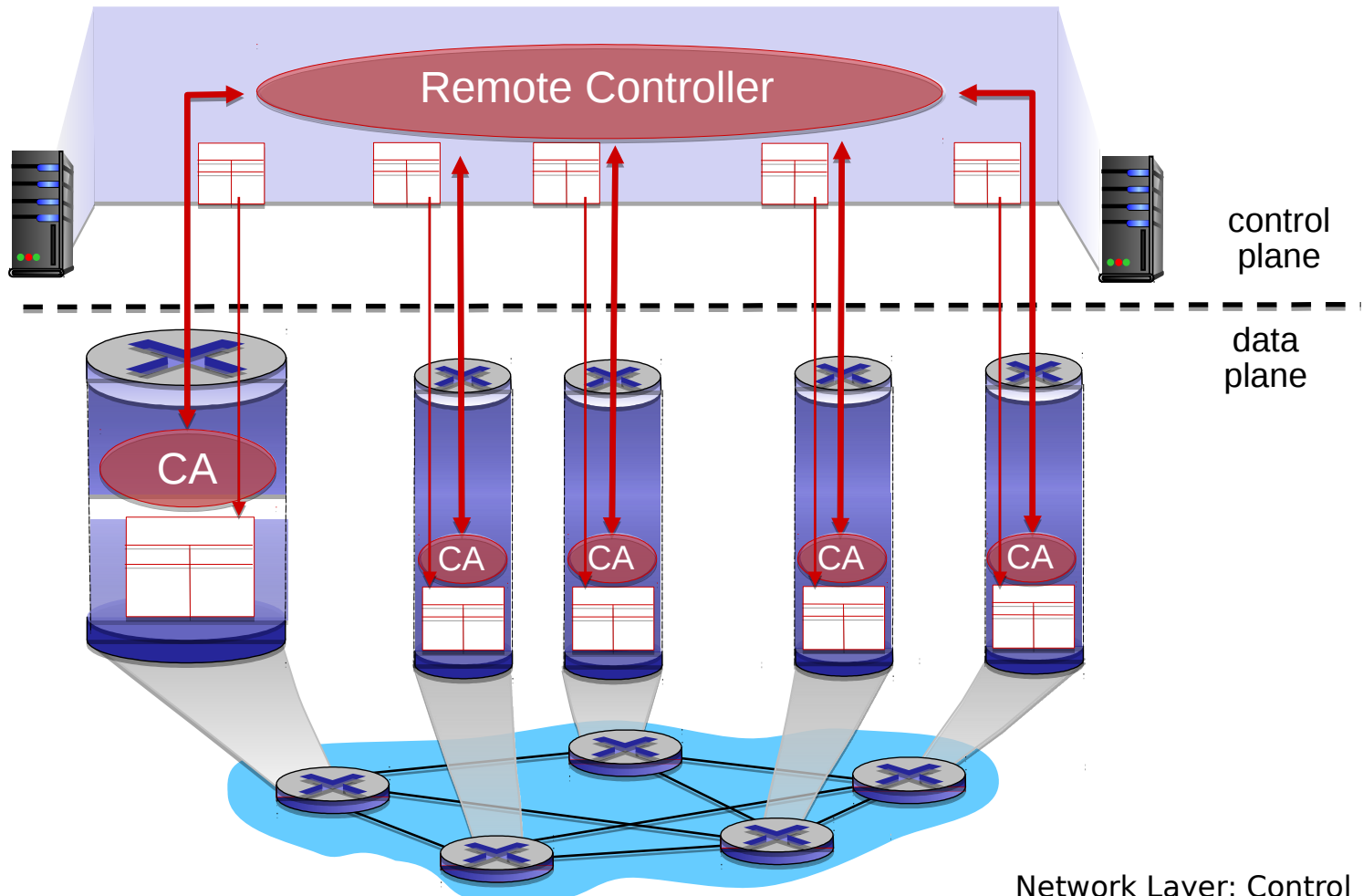
Recall: per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables

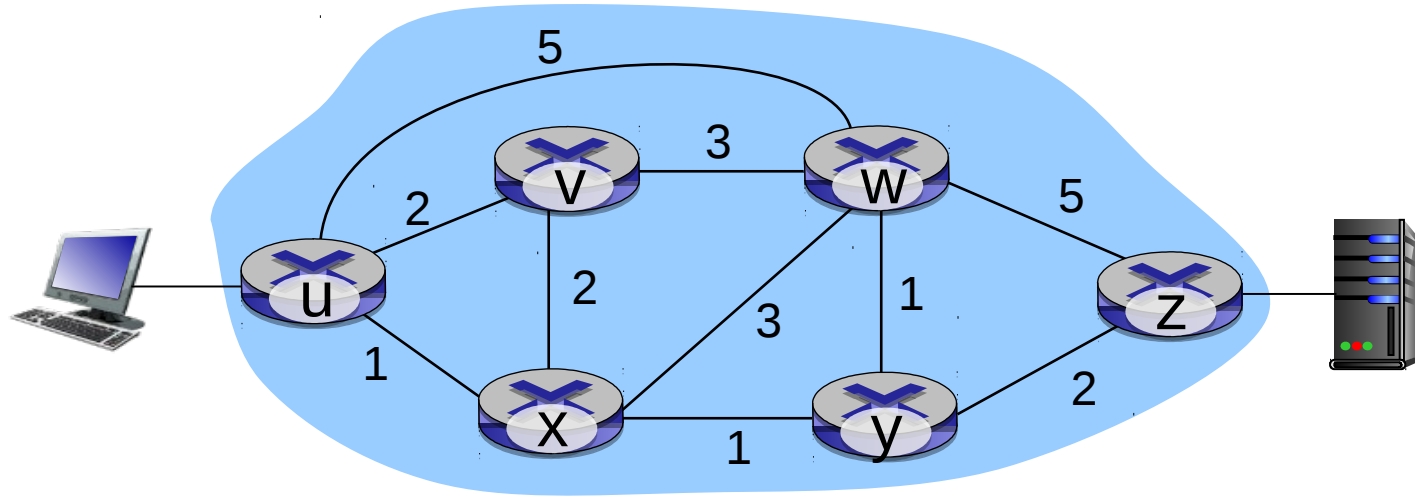


Software defined networking (SDN)

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 (ultimately independent of the SDN vs distributed)

Traffic engineering: difficult traditional routing

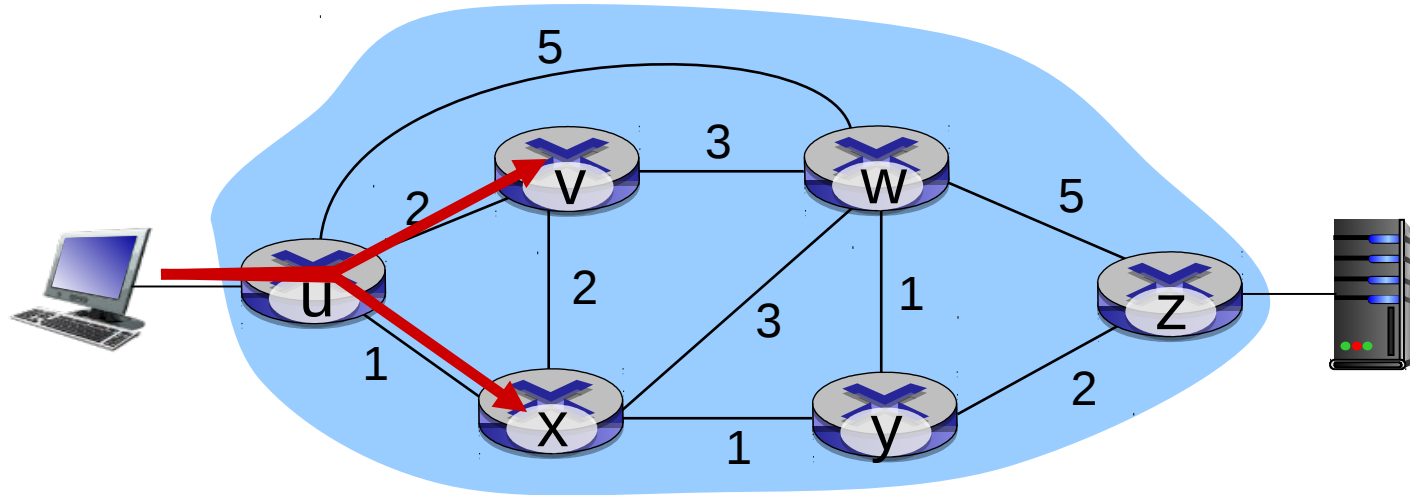


Q: what if network operator wants u-to-z traffic to flow along $uvwz$, x-to-z traffic to flow $xwyz$?

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

Link weights are only control “knobs”: not ideal!

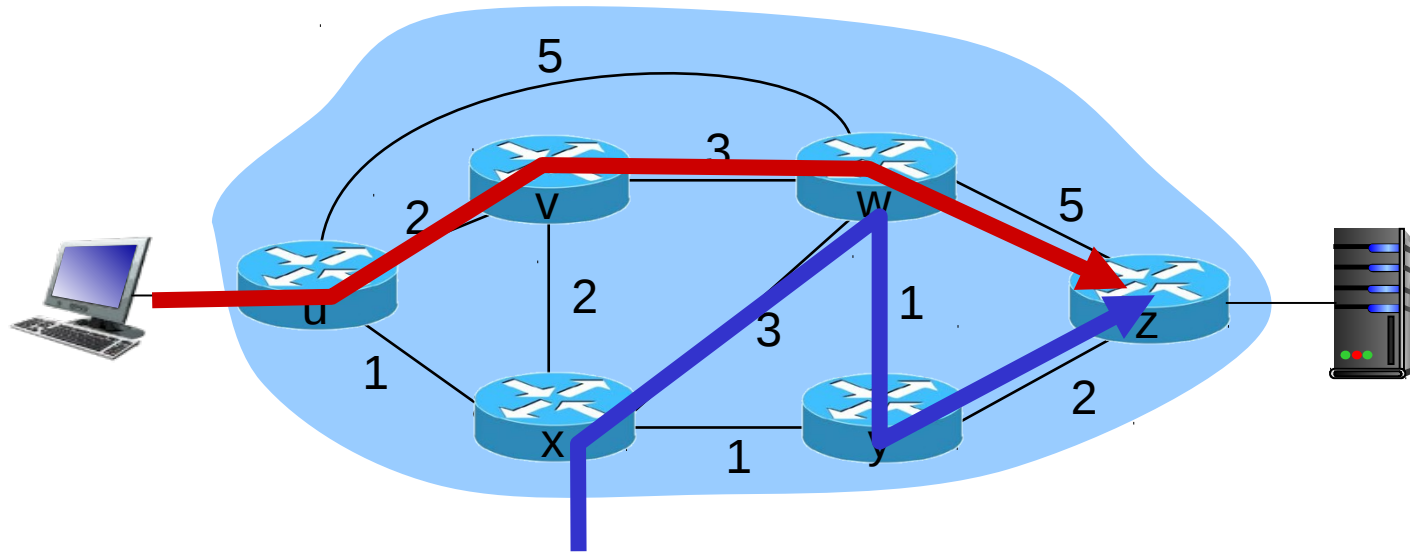
Traffic engineering: difficult



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

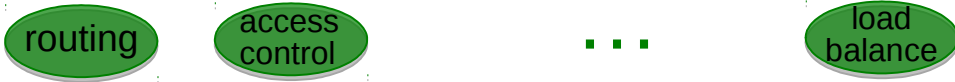


Q: what if w wants to route blue and red traffic differently?

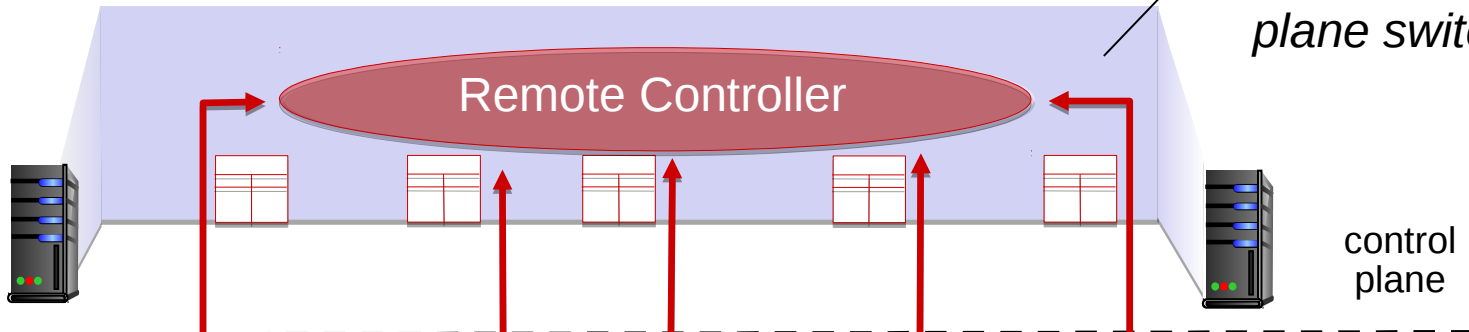
A: can't do it (with destination based forwarding, and LS, DV routing)

Software defined networking (SDN)

4. programmable control applications



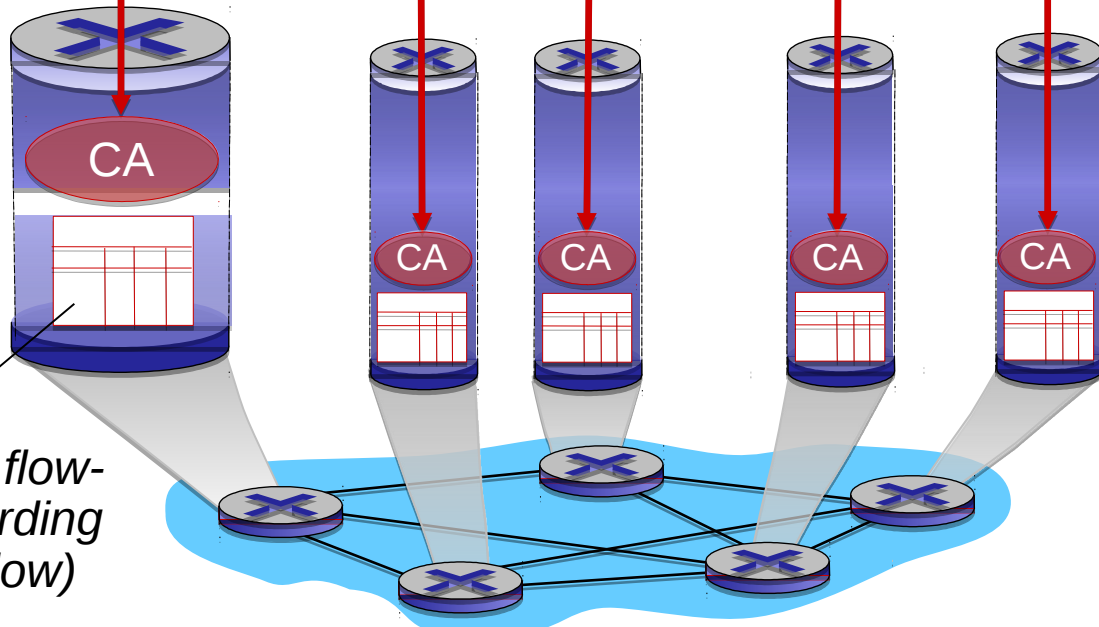
3. control plane functions external to data-plane switches



control plane
data plane

2. control, data plane separation

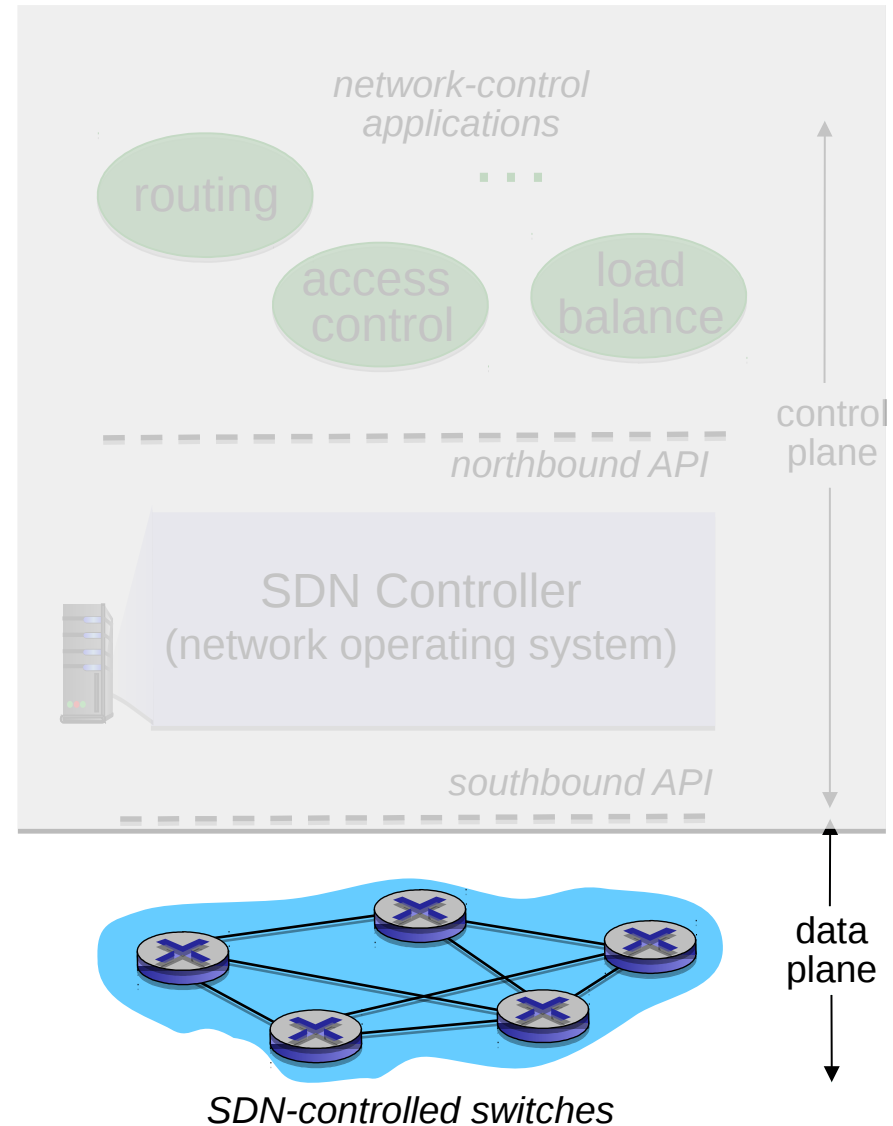
1. generalized "flow-based" forwarding (e.g., OpenFlow)



SDN perspective: data plane switches

Data plane switches

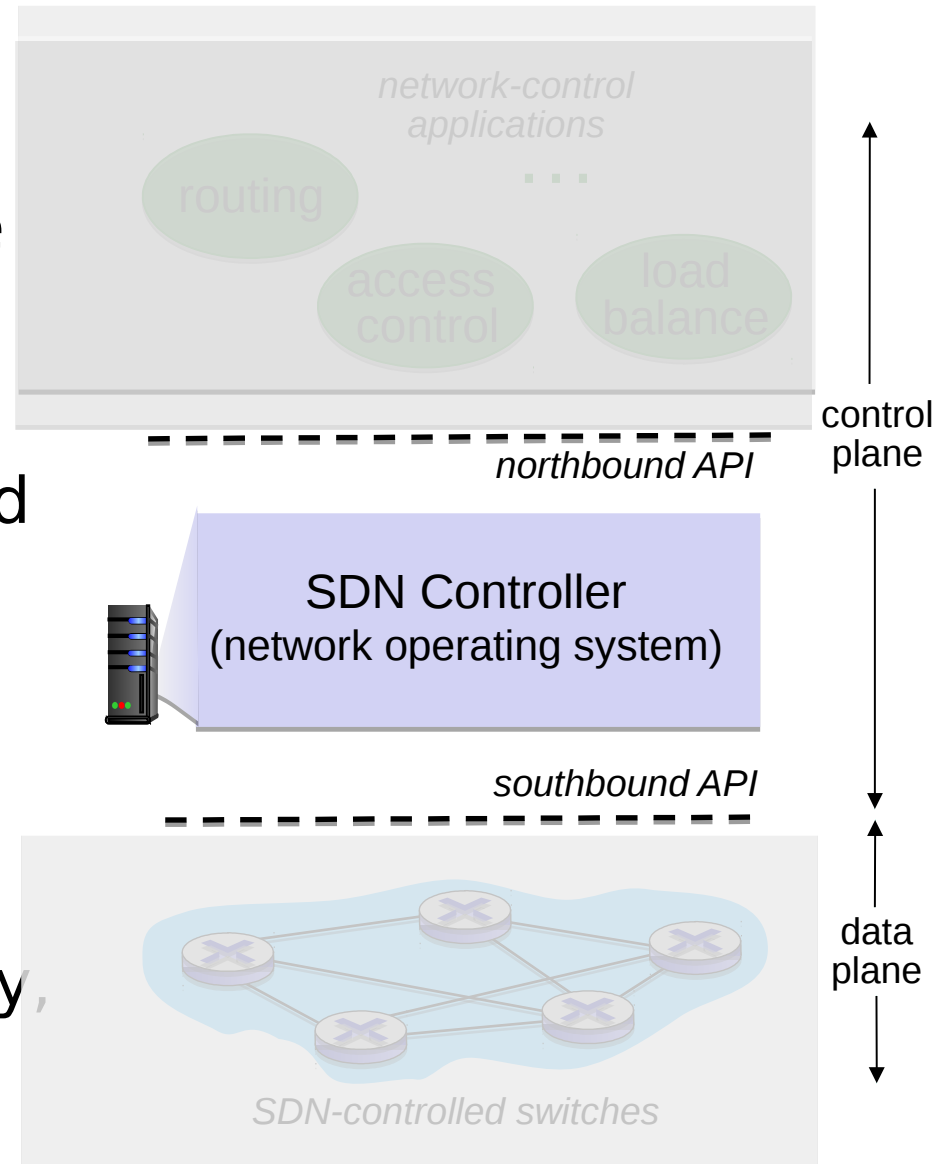
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable and what is not
- protocol for communicating with controller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



SDN perspective: SDN controller

SDN controller (network OS):

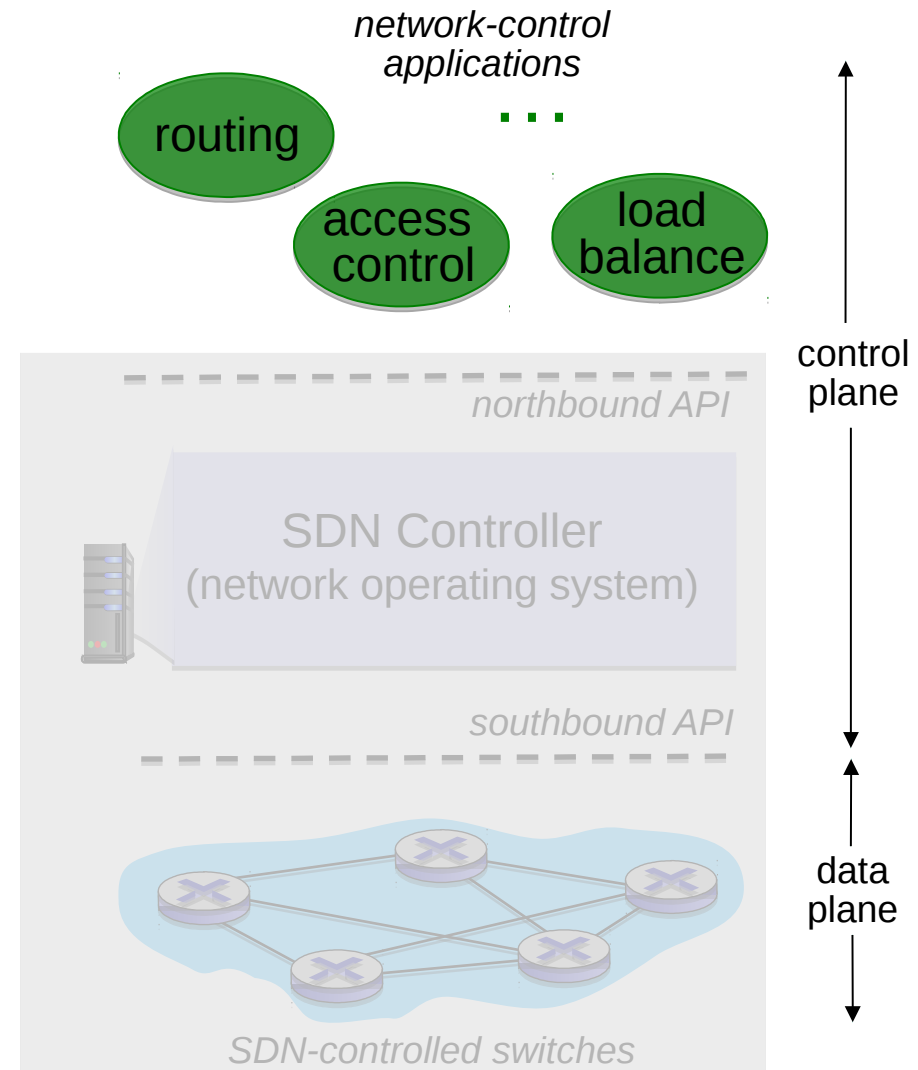
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



SDN perspective: control applications

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

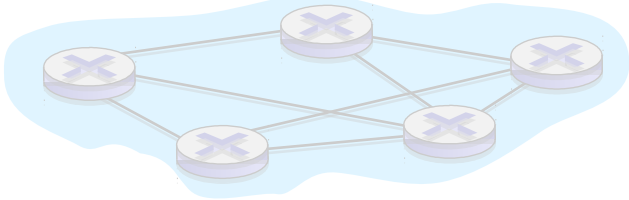
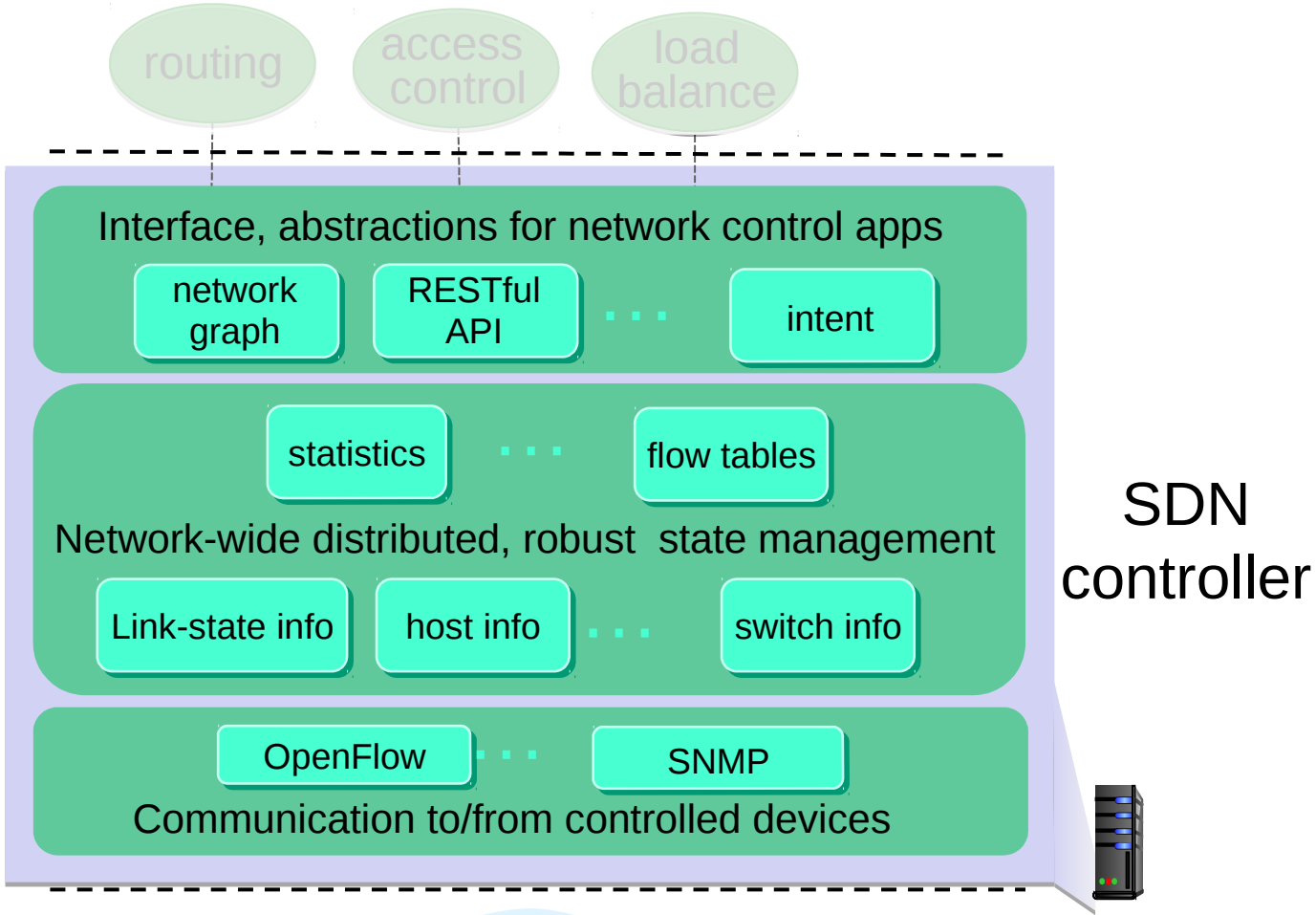


Components of SDN controller

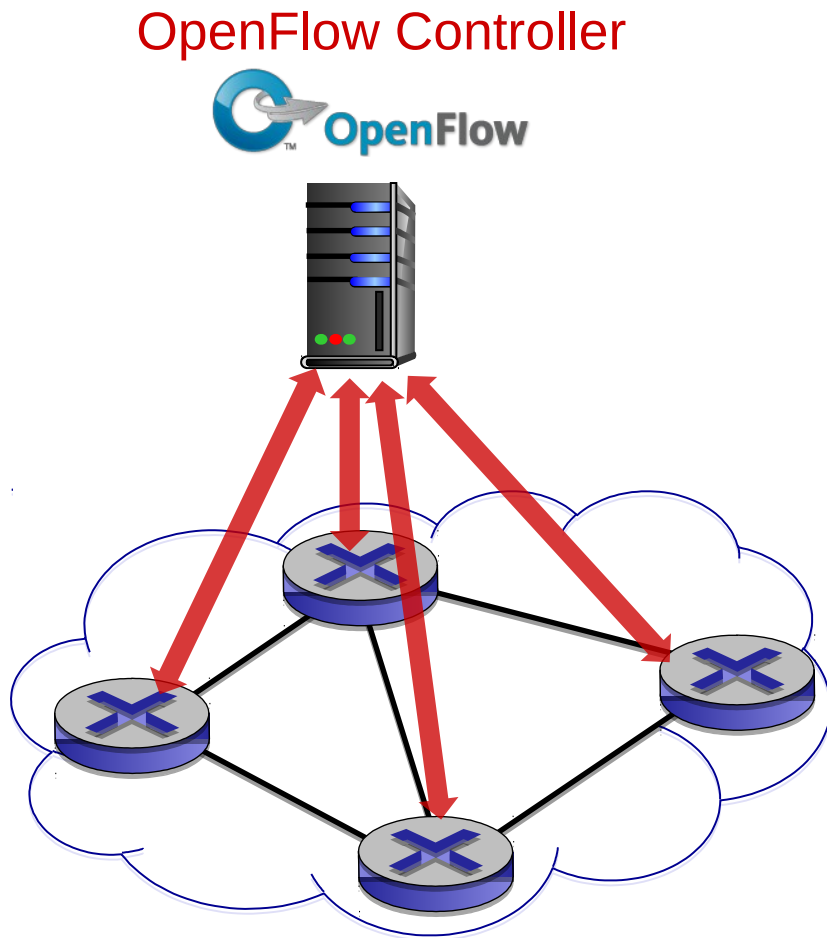
Interface layer
to network
control apps:
abstractions API

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

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



OpenFlow protocol

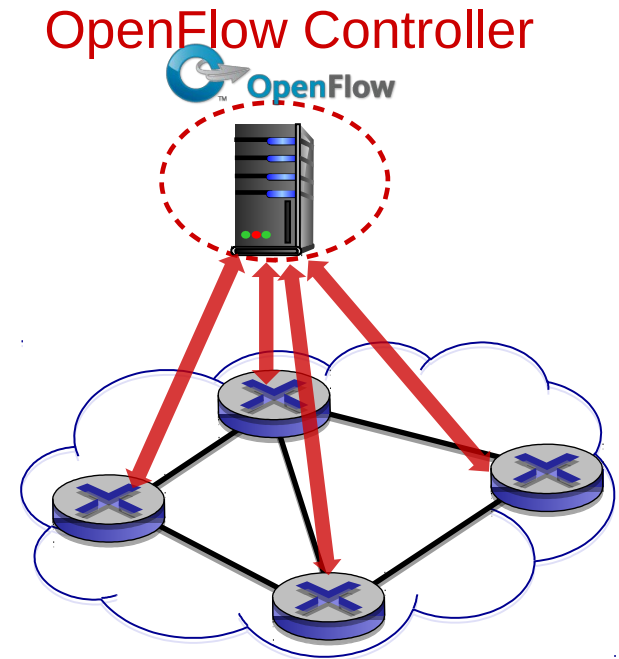


- operates between controller, switch
- TCP used to exchange messages
 - optional encryption
- three classes of OpenFlow messages:
 - controller-to-switch
 - asynchronous (switch to controller)
 - symmetric (misc)

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

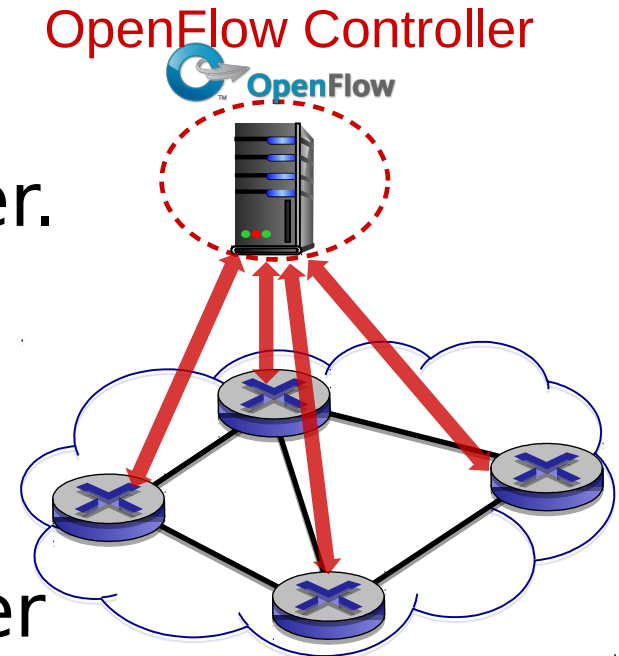


OpenFlow: switch-to-controller messages

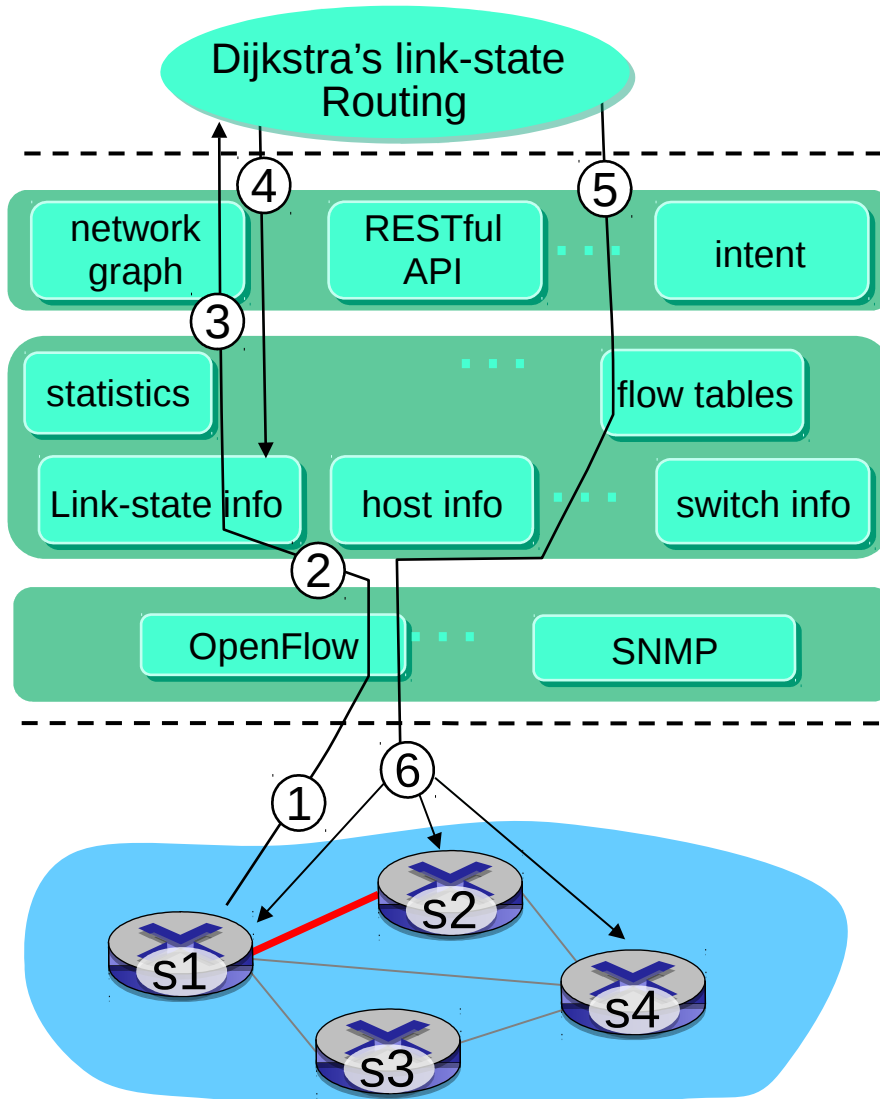
Key switch-to-controller messages

- **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.

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

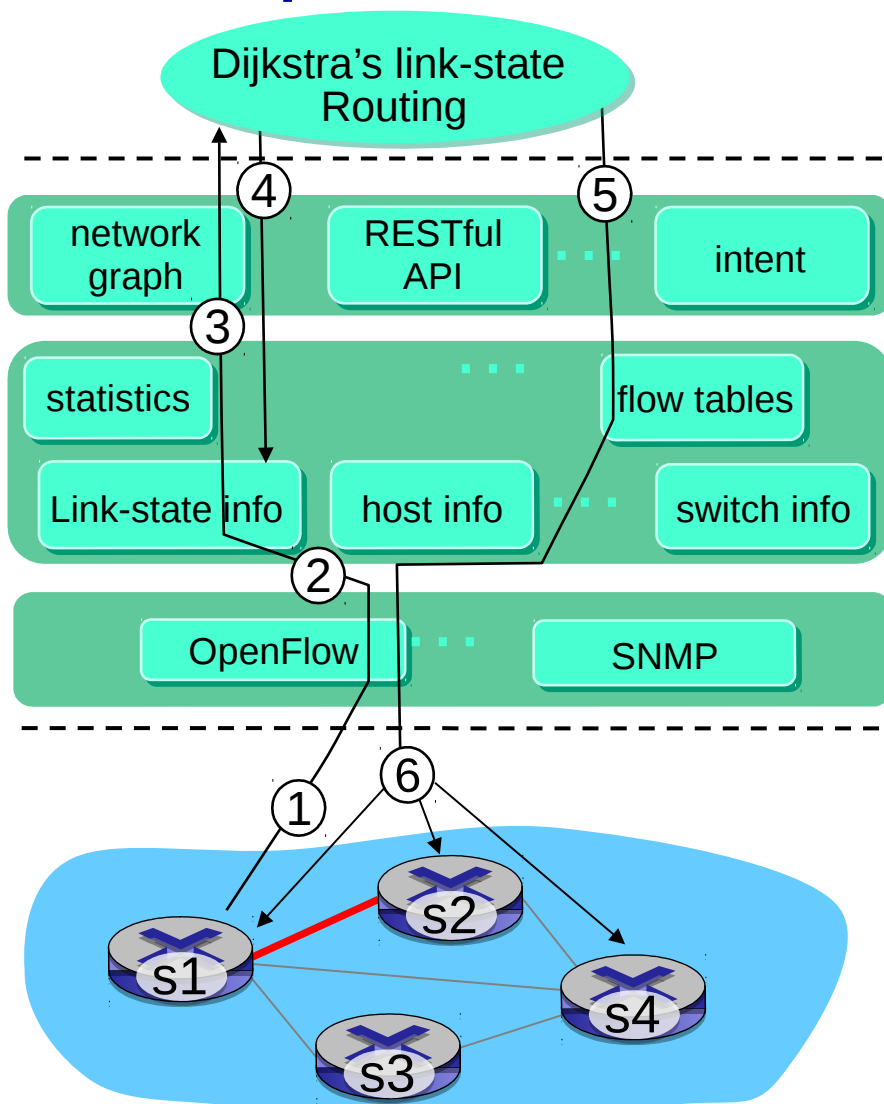


SDN: control/data plane interaction example



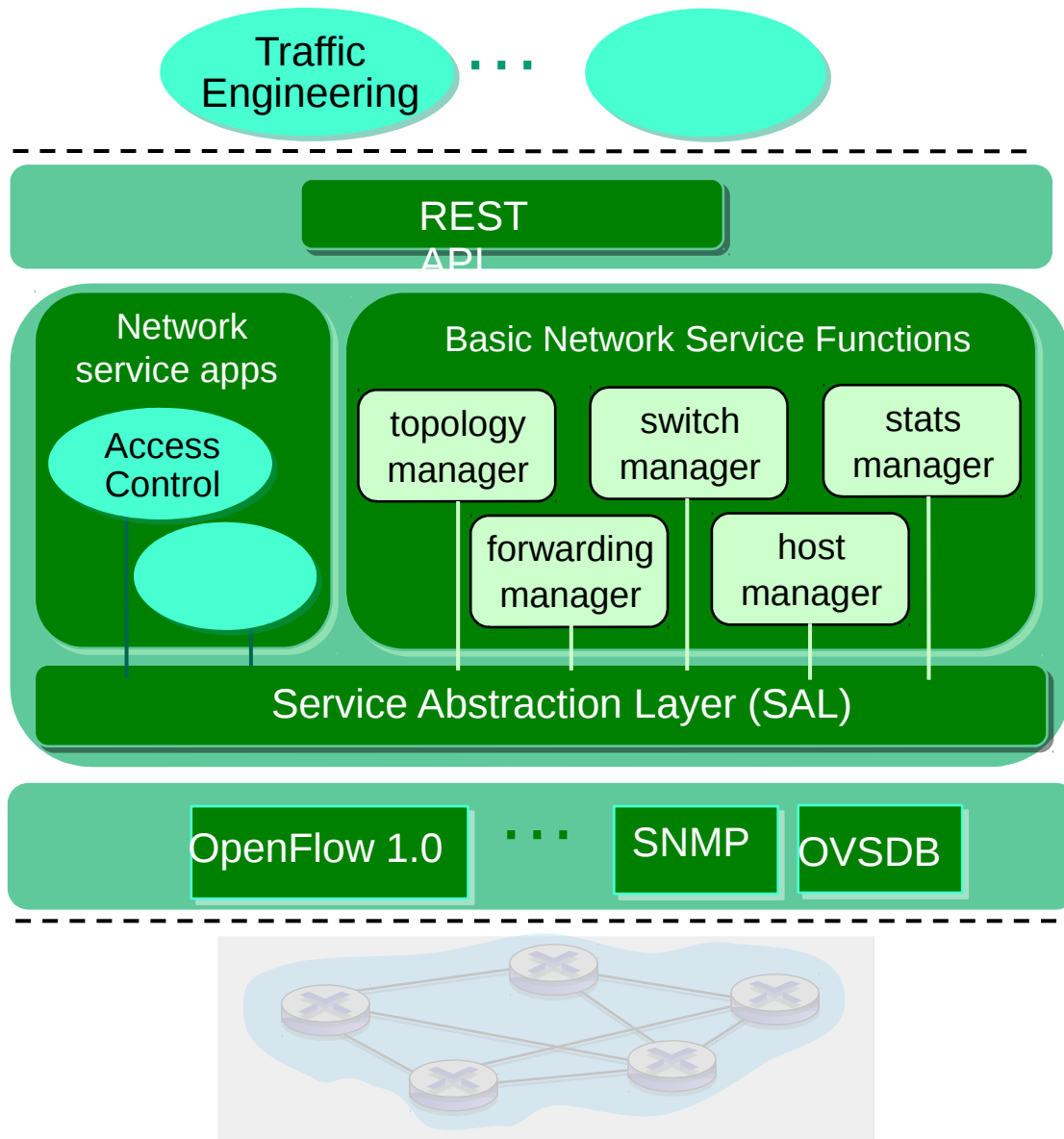
- ① S1, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
- ④ Dijkstra's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

SDN: control/data plane interaction example



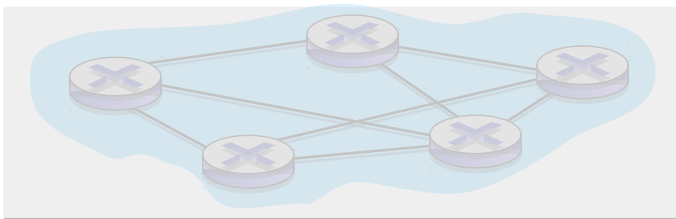
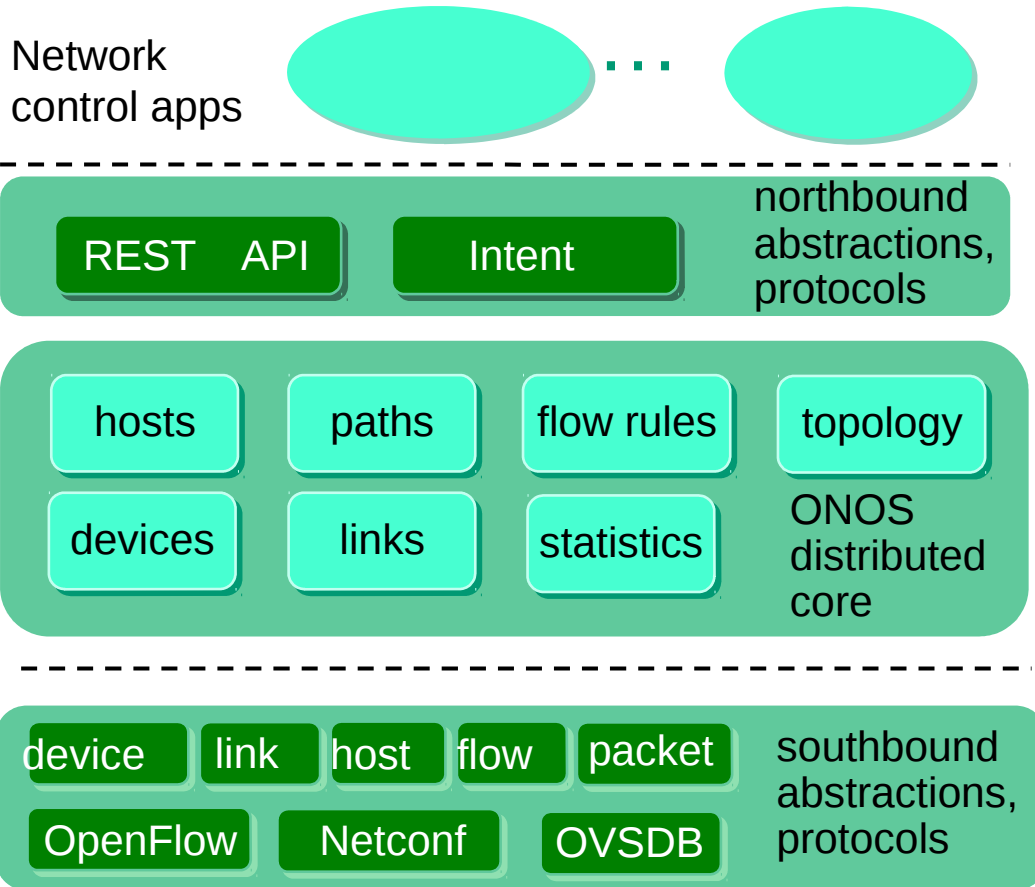
- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

OpenDaylight (ODL) controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN 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

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

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

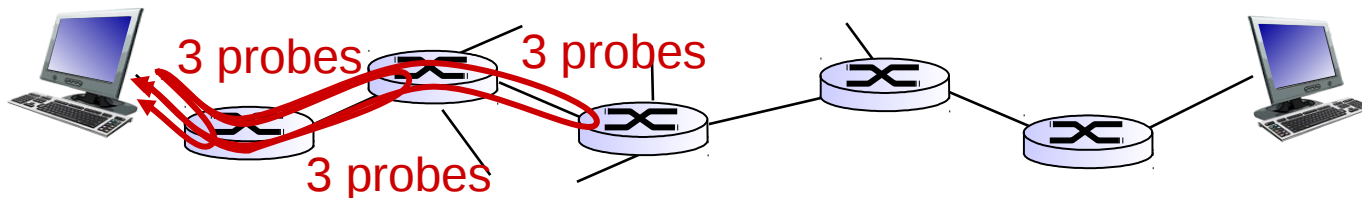
ICMP: internet control message protocol

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

<u>Type</u>	<u>Code</u>	<u>description</u>
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 series of UDP segments to destination
 - first set has TTL = 1
 - second set has TTL=2, etc.
 - unlikely port number
 - when datagram in n th set arrives to n th router:
 - router discards datagram and sends source ICMP message (type 11, code 0)
 - ICMP message include name of router & IP address
 - when ICMP message arrives, source records RTTs
- stopping criteria:*
- UDP segment eventually arrives at destination host
 - destination returns ICMP “port unreachable” message (type 3, code 3)
 - source stops



Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

What is network management?

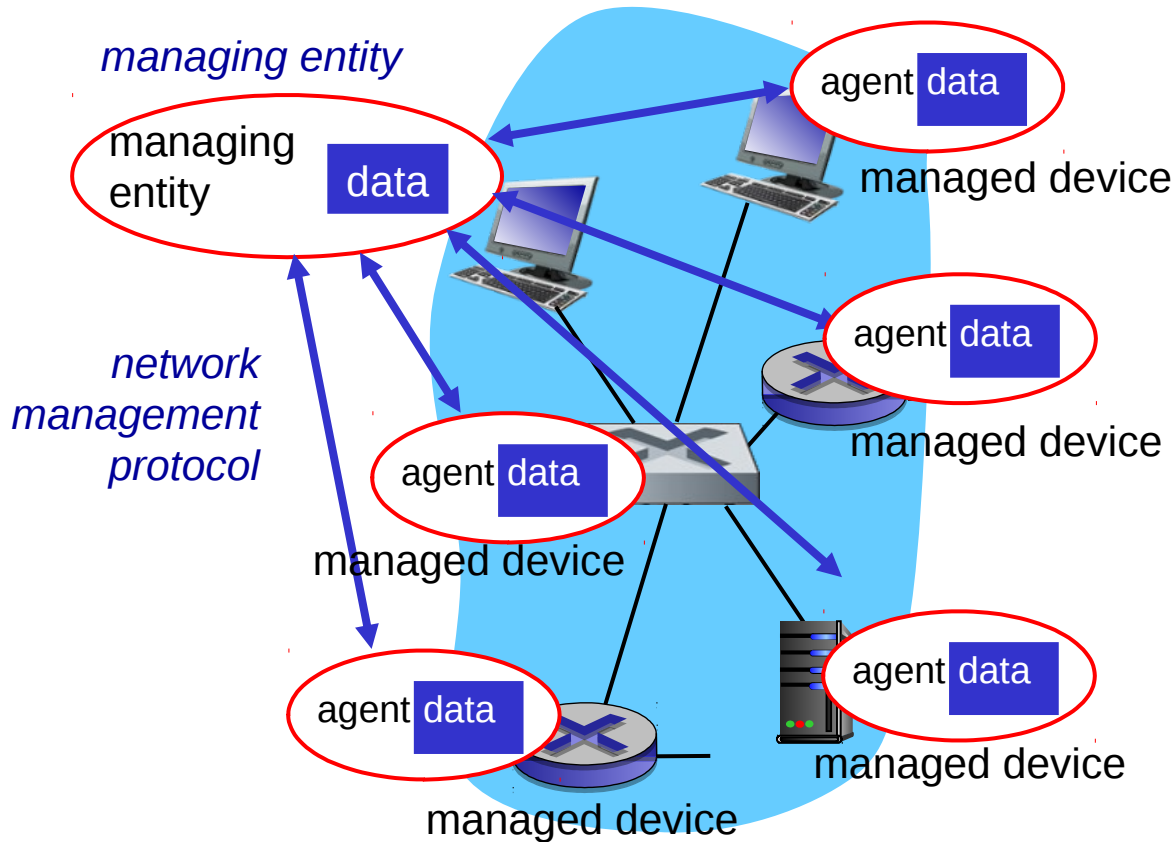
- **autonomous systems (aka “network”):** 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
 - jet airplane
 - nuclear power plant
 - others?



"**Network management** includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

Infrastructure for network management

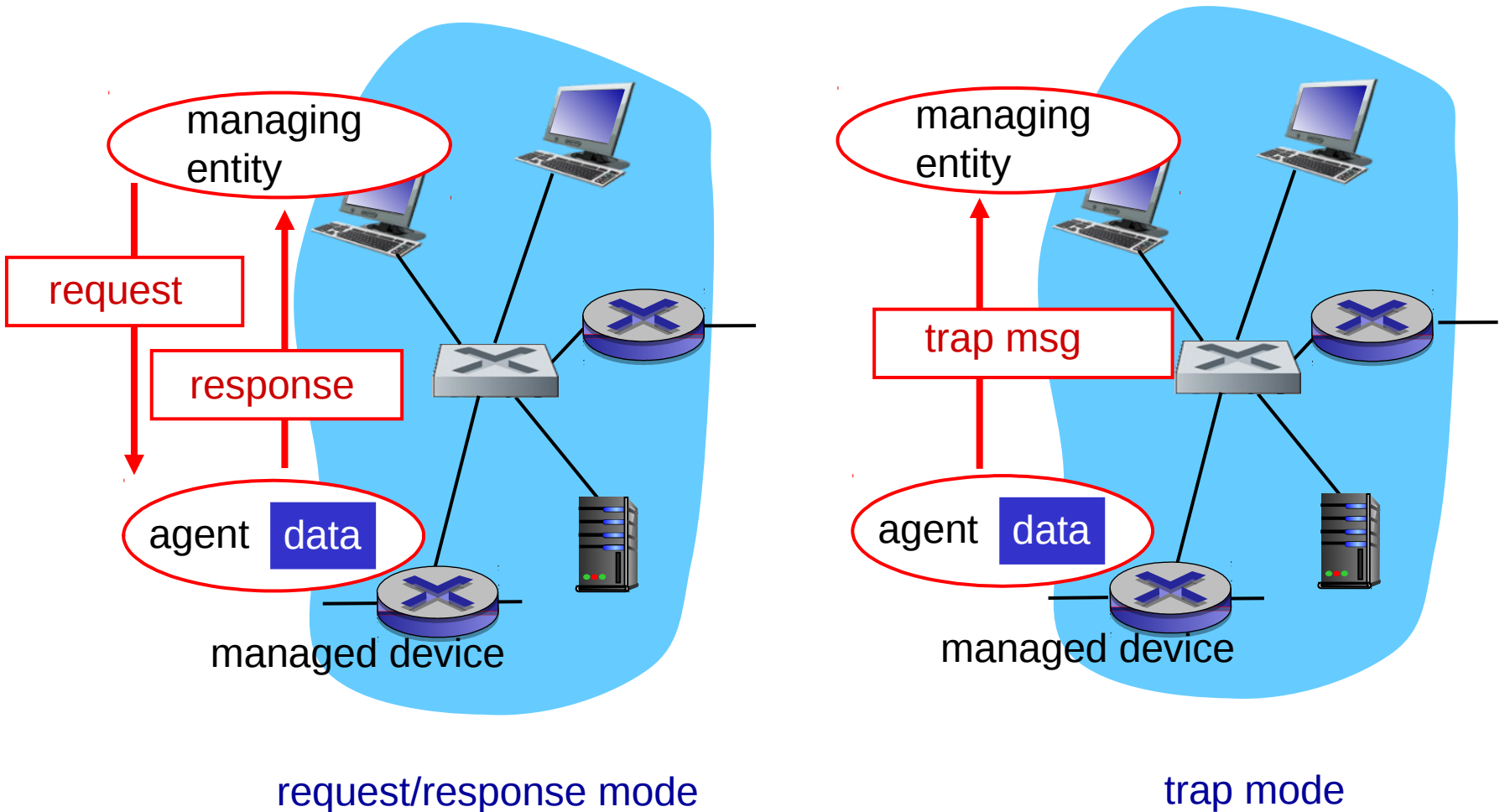
definitions:



managed devices contain *managed objects* whose data is gathered into a **Management Information Base (MIB)**

SNMP protocol

Two ways to convey MIB info, commands:



SNMP protocol: message types

Message type

Function

GetRequest
GetNextRequest
GetBulkRequest

manager-to-agent: “get me data”
(data instance, next data in list, block of data)

InformRequest

manager-to-manager: here’s MIB value

SetRequest

manager-to-agent: set MIB value

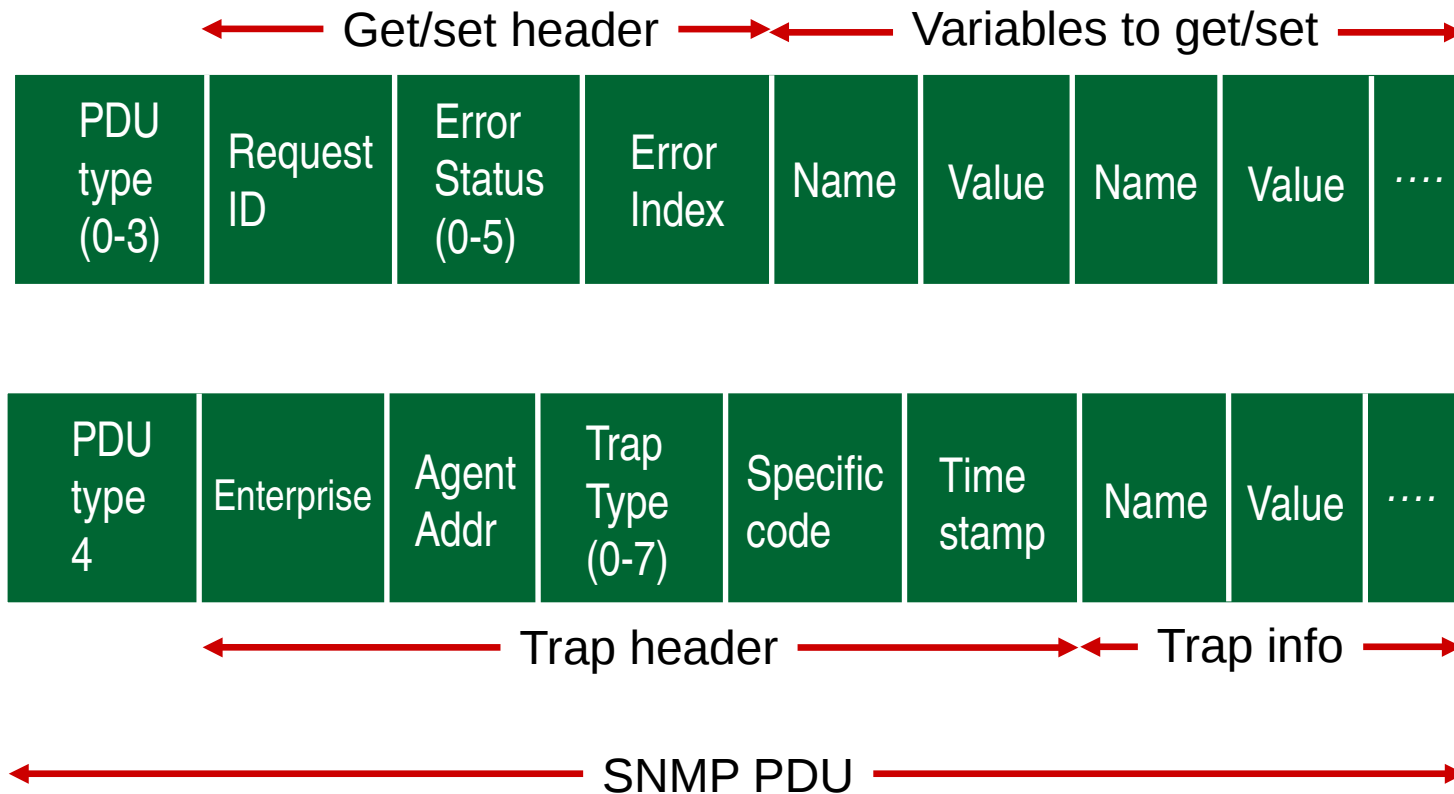
Response

Agent-to-manager: value, response to Request

Trap

Agent-to-manager: inform manager of exceptional event

SNMP protocol: message formats



More on network management: see earlier editions of text!

Chapter 5: outline

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

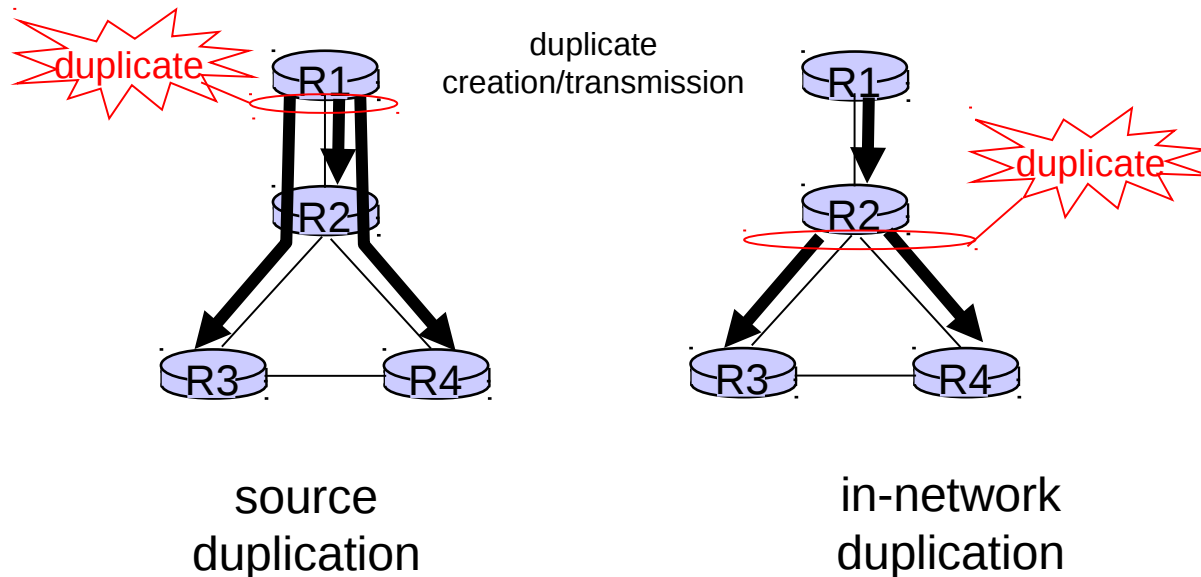
4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

Broadcast routing

- ❖ deliver packets from source to all other nodes
- ❖ source duplication is inefficient:



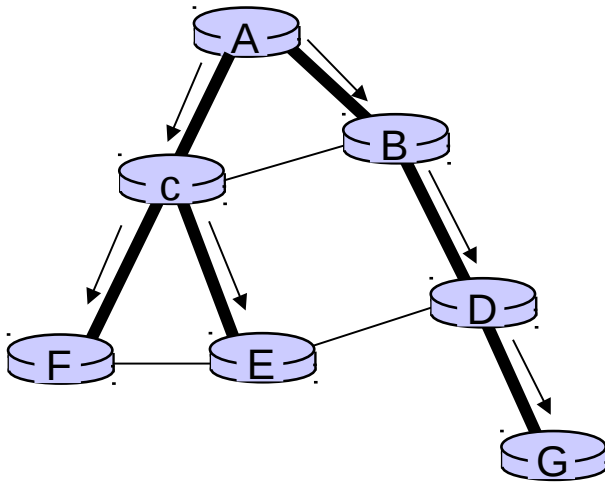
- ❖ source duplication: how does source determine recipient addresses?

In-network duplication

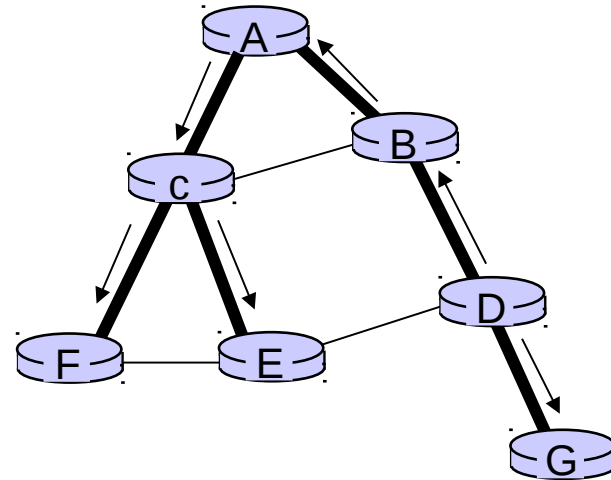
- ❖ *flooding*: when node receives broadcast packet, sends copy to all neighbors
 - problems: cycles & broadcast storm
- ❖ *controlled flooding*: node only broadcasts pkt if it hasn't broadcast same packet before
 - node keeps track of packet ids already broadcasted
 - or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- ❖ *spanning tree*:
 - no redundant packets received by any node

Spanning tree

- ❖ first construct a spanning tree
- ❖ nodes then forward/make copies only along spanning tree



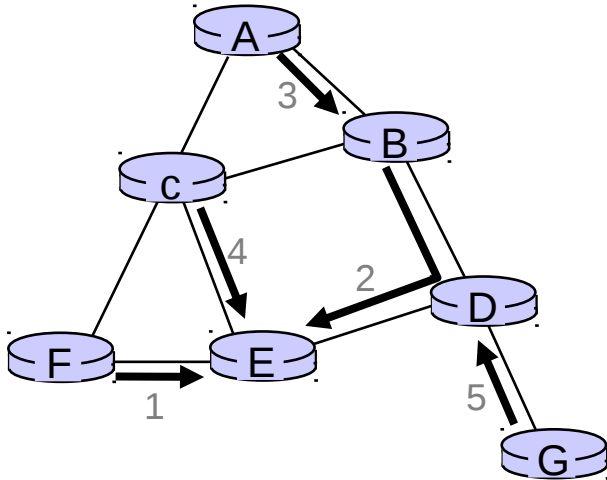
(a) broadcast initiated at A



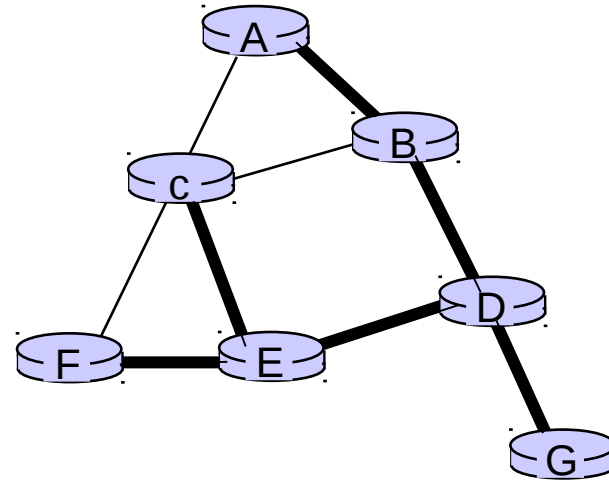
(b) broadcast initiated at D

Spanning tree: creation

- ❖ center node
- ❖ each node sends unicast join message to center node
 - message forwarded until it arrives at a node already belonging to spanning tree



(a) stepwise construction of spanning tree (center: E)

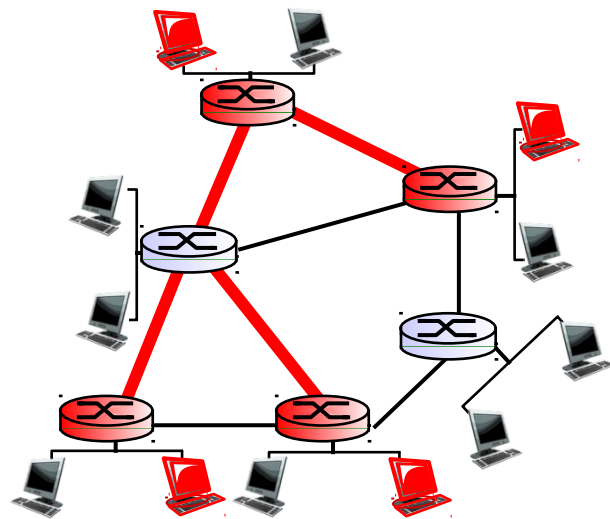


(b) constructed spanning tree

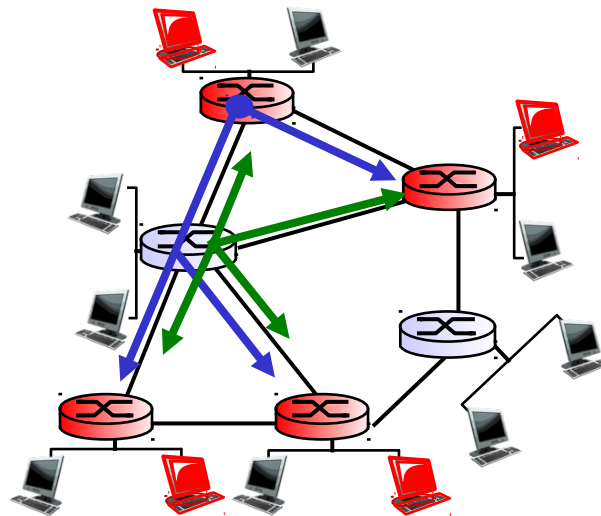
Multicast routing: problem statement

goal: find a tree (or trees) connecting routers having local mcast group members

- ❖ *tree:* not all paths between routers used
- ❖ *shared-tree:* same tree used by all group members
- ❖ *source-based:* different tree from each sender to rcvrs



shared tree



source-based trees

legend



group member



not group member



router with a group member



router without group member

Approaches for building mcast trees

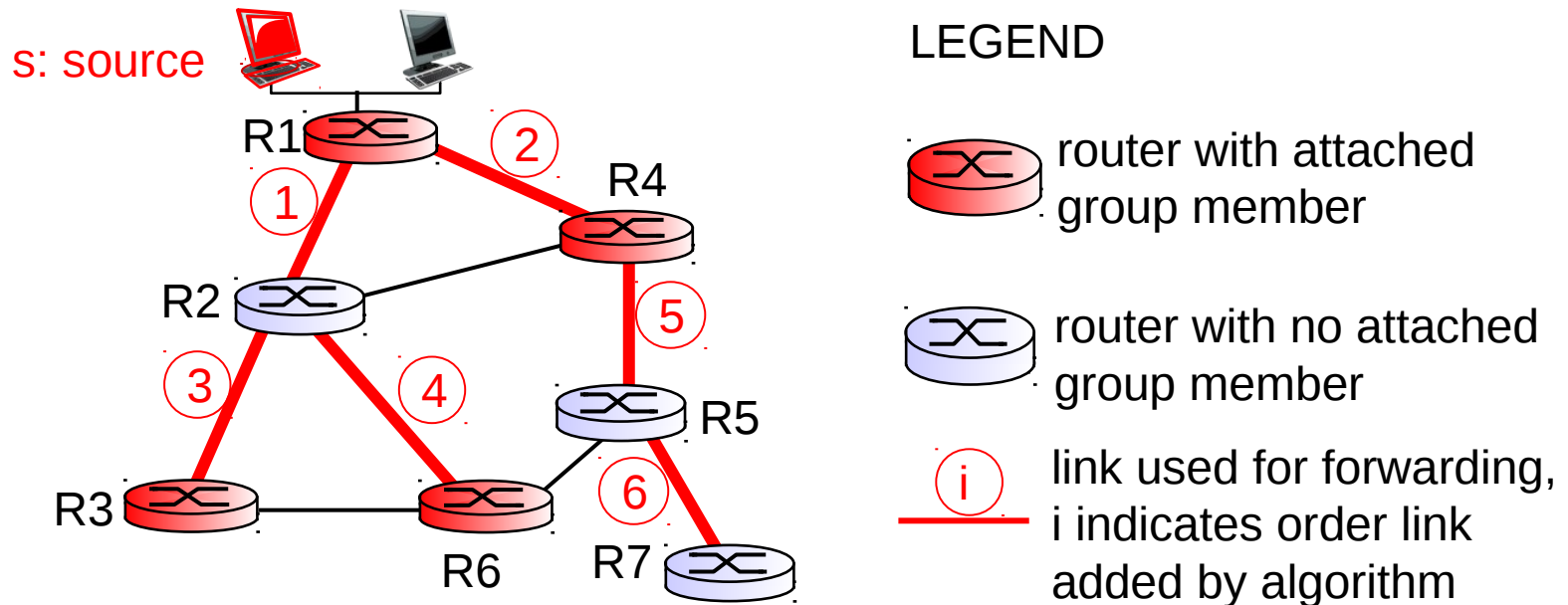
approaches:

- ❖ *source-based tree*: one tree per source
 - shortest path trees
 - reverse path forwarding
- ❖ *group-shared tree*: group uses one tree
 - minimal spanning (Steiner)
 - center-based trees

...we first look at basic approaches, then specific protocols adopting these approaches

Shortest path tree

- ❖ mcast forwarding tree: tree of shortest path routes from source to all receivers
 - Dijkstra's algorithm

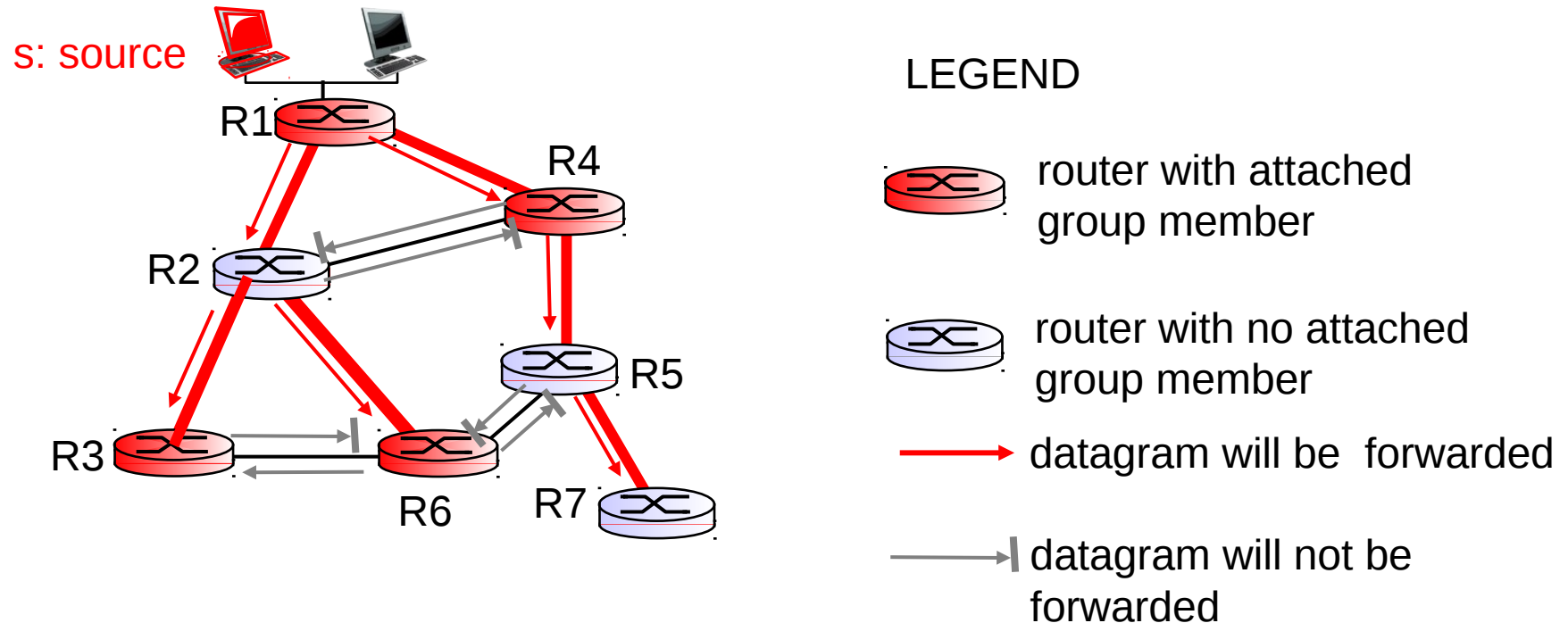


Reverse path forwarding

- ❖ rely on router's knowledge of unicast shortest path from it to sender
- ❖ each router has simple forwarding behavior:

if (mcast datagram received on incoming link on shortest path back to center)
then flood datagram onto all outgoing links
else ignore datagram

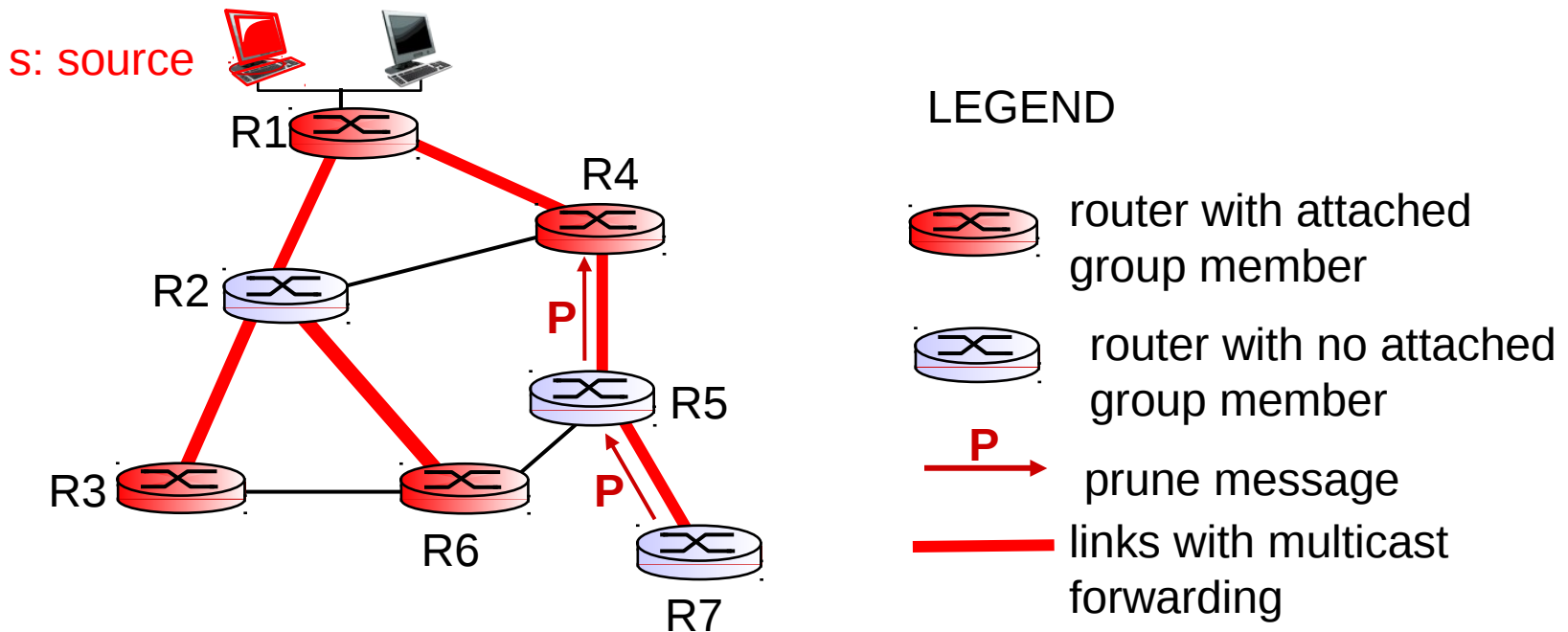
Reverse path forwarding: example



- ❖ result is a source-specific *reverse* SPT
 - may be a bad choice with asymmetric links

Reverse path forwarding: pruning

- ❖ forwarding tree contains subtrees with no mcast group members
 - no need to forward datagrams down subtree
 - “prune” msgs sent upstream by router with no downstream group members



Shared-tree: steiner tree

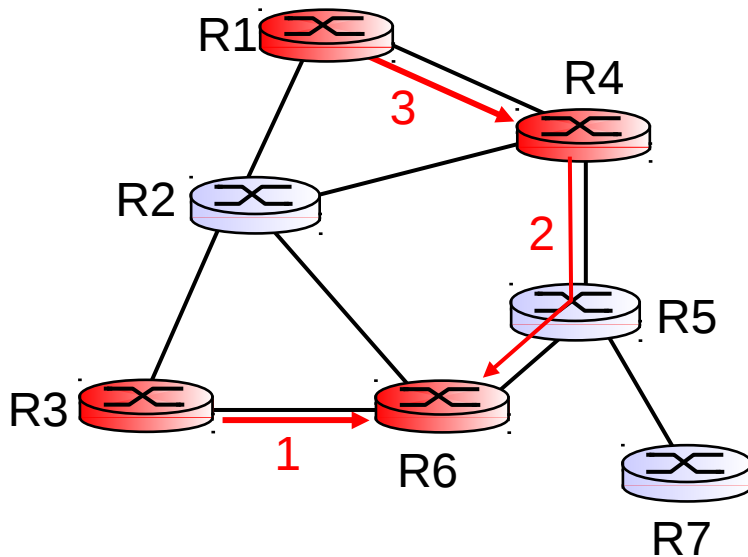
- ❖ *steiner tree*: minimum cost tree connecting all routers with attached group members
- ❖ problem is NP-complete
- ❖ excellent heuristics exists
- ❖ not used in practice:
 - computational complexity
 - information about entire network needed
 - monolithic: rerun whenever a router needs to join/leave

Center-based trees




- ❖ single delivery tree shared by all
- ❖ one router identified as “*center*” of tree
- ❖ to join:
 - edge router sends unicast *join-msg* addressed to center router
 - *join-msg* “processed” by intermediate routers and forwarded towards center
 - *join-msg* either hits existing tree branch for this center, or arrives at center
 - path taken by *join-msg* becomes new branch of tree for this router

Center-based trees: example

suppose R6 chosen as center:



LEGEND

-  router with attached group member
-  router with no attached group member
-  path order in which join messages generated

Internet Multicasting Routing: DVMRP

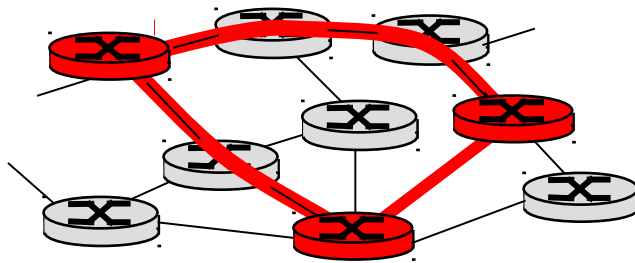
- ❖ **DVMRP**: distance vector multicast routing protocol, RFC1075
- ❖ *flood and prune*: reverse path forwarding, source-based tree
 - RPF tree based on DVMRP's own routing tables constructed by communicating DVMRP routers
 - no assumptions about underlying unicast
 - initial datagram to mcast group flooded everywhere via RPF
 - routers not wanting group: send upstream prune msgs

DVMRP: continued...

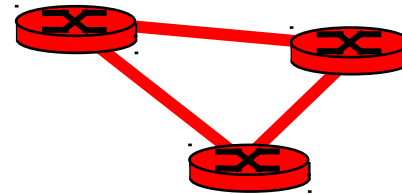
- ❖ *soft state*: DVMRP router periodically (1 min.) “forgets” branches are pruned:
 - mcast data again flows down unpruned branch
 - downstream router: re prune or else continue to receive data
- ❖ routers can quickly regraft to tree
 - following IGMP join at leaf
- ❖ odds and ends
 - commonly implemented in commercial router

Tunneling

Q: how to connect “islands” of multicast routers in a “sea” of unicast routers?



physical topology



logical topology

- ❖ mcast datagram encapsulated inside “normal” (non-multicast-addressed) datagram
- ❖ normal IP datagram sent thru “tunnel” via regular IP unicast to receiving mcast router (recall IPv6 inside IPv4 tunneling)
- ❖ receiving mcast router unencapsulates to get mcast datagram

PIM: Protocol Independent Multicast

- ❖ not dependent on any specific underlying unicast routing algorithm (works with all)
- ❖ two different multicast distribution scenarios :

dense:

- ❖ group members densely packed, in “close” proximity.
- ❖ bandwidth more plentiful

sparse:

- ❖ # networks with group members small wrt # interconnected networks
- ❖ group members “widely dispersed”
- ❖ bandwidth not plentiful

Consequences of sparse-dense dichotomy:

dense

- ❖ group membership by routers *assumed* until routers explicitly prune
- ❖ *data-driven* construction on mcast tree (e.g., RPF)
- ❖ bandwidth and non-group-router processing *profligate*

sparse:

- ❖ no membership until routers explicitly join
- ❖ *receiver-driven* construction of mcast tree (e.g., center-based)
- ❖ bandwidth and non-group-router processing *conservative*

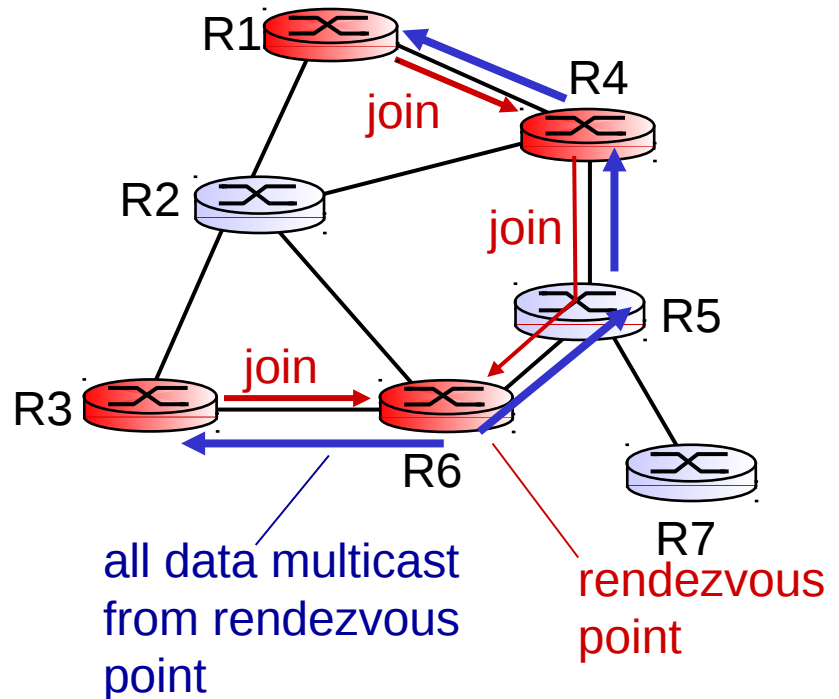
PIM- dense mode

flood-and-prune RPF: similar to DVMRP but...

- ❖ underlying unicast protocol provides RPF info for incoming datagram
- ❖ less complicated (less efficient) downstream flood than DVMRP reduces reliance on underlying routing algorithm
- ❖ has protocol mechanism for router to detect it is a leaf-node router

PIM - sparse mode

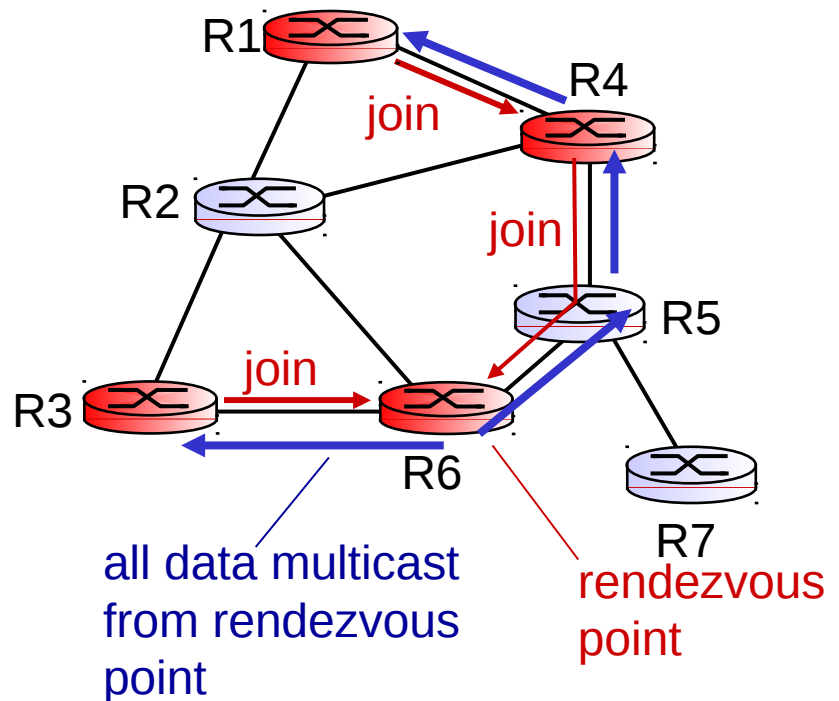
- ❖ center-based approach
- ❖ router sends *join* msg to rendezvous point (RP)
 - intermediate routers update state and forward *join*
- ❖ after joining via RP, router can switch to source-specific tree
 - increased performance: less concentration, shorter paths



PIM - sparse mode

sender(s):

- ❖ unicast data to RP, which distributes down RP-rooted tree
- ❖ RP can extend mcast tree upstream to source
- ❖ RP can send *stop* msg if no attached receivers
 - “no one is listening!”



Chapter 5: 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!