

# 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:*

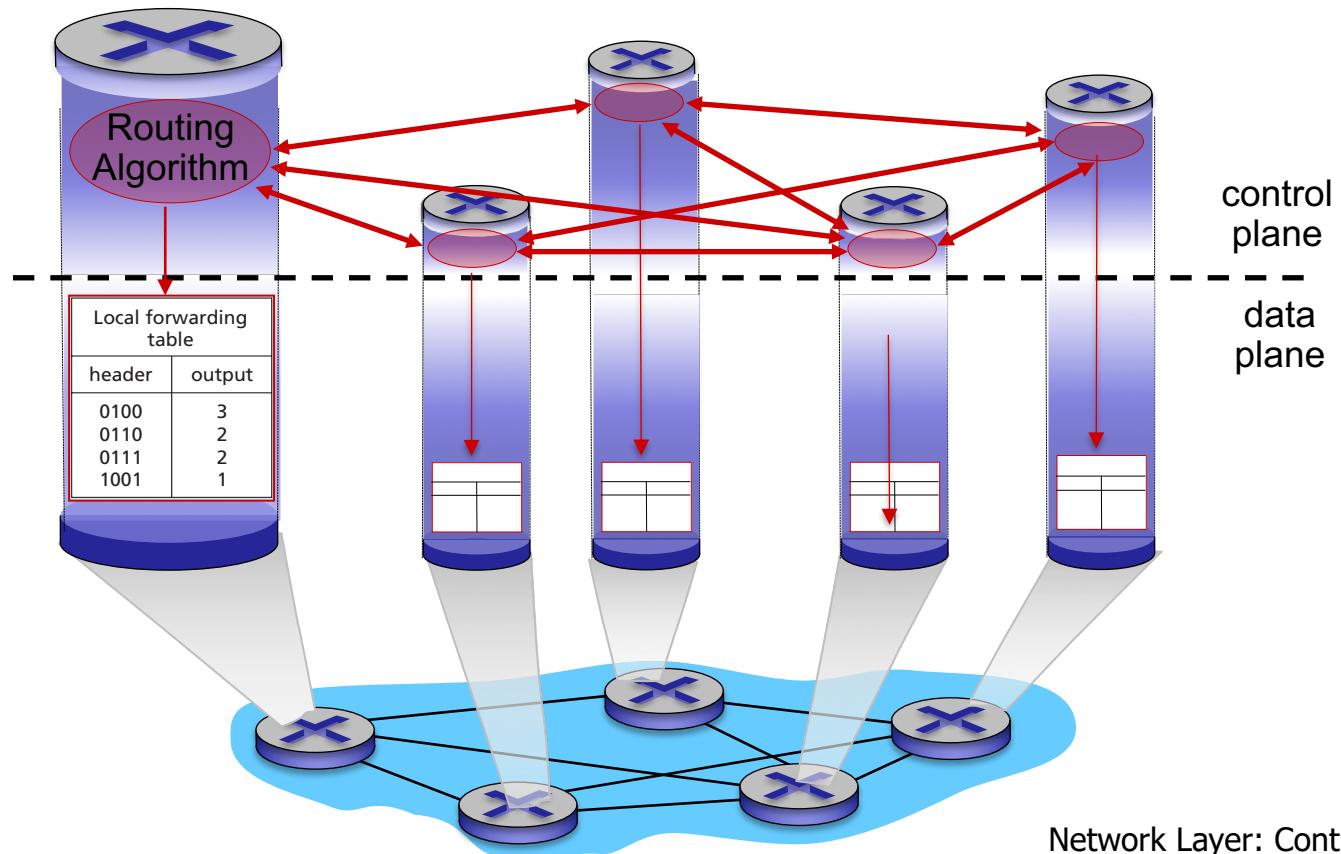
- *forwarding*: move packets from router's input to appropriate router output *data plane*
- *routing*: determine route taken by packets from source to destination *control plane*

*Two approaches to structuring network control plane:*

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

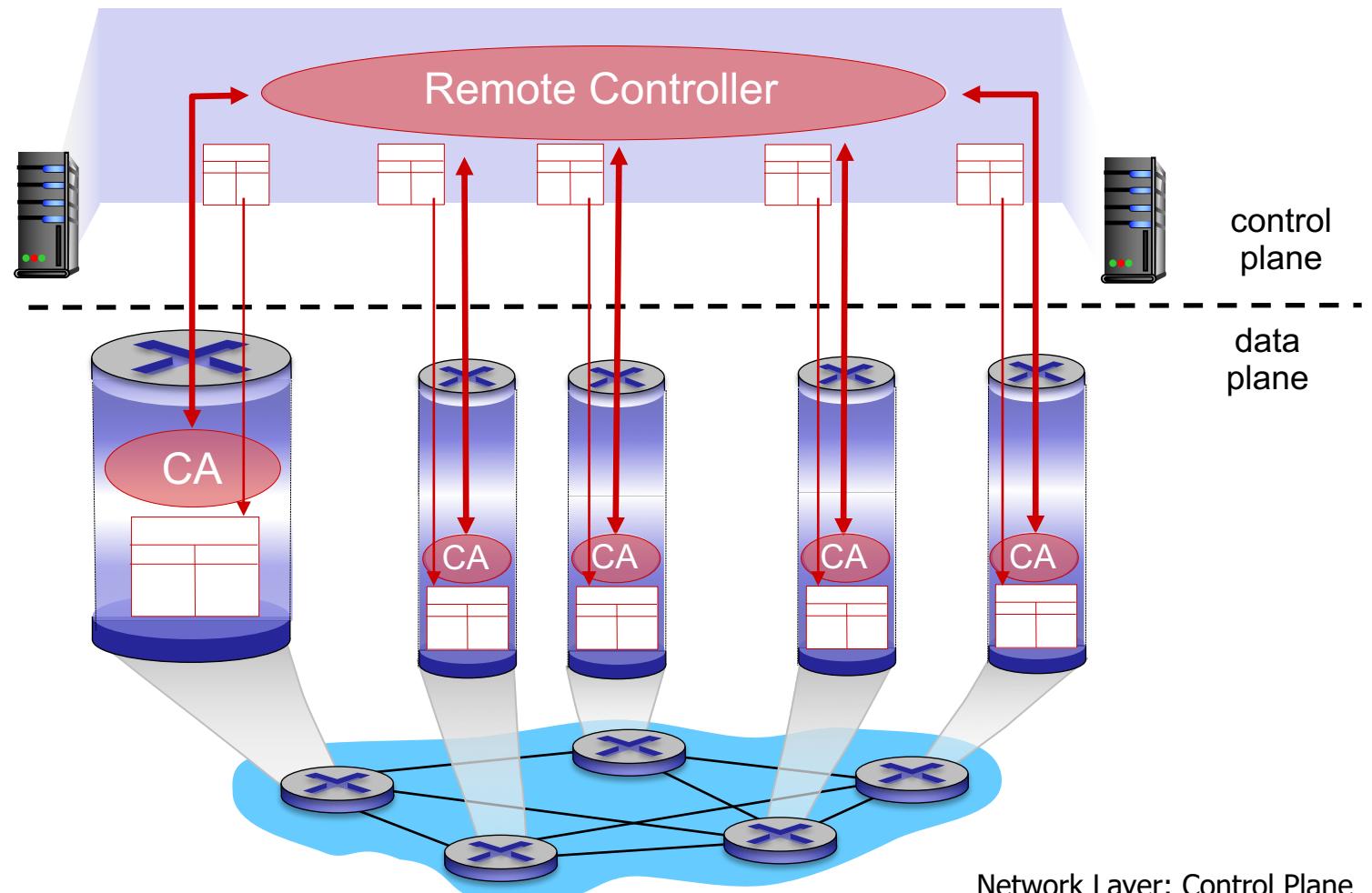
# Per-router control plane

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

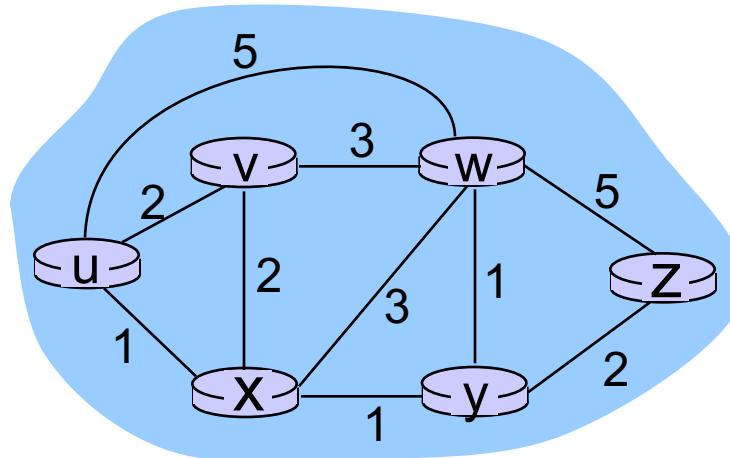
5.7 Network management and SNMP

# 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”
- routing: a “top-10” networking challenge!

# Graph abstraction of the network



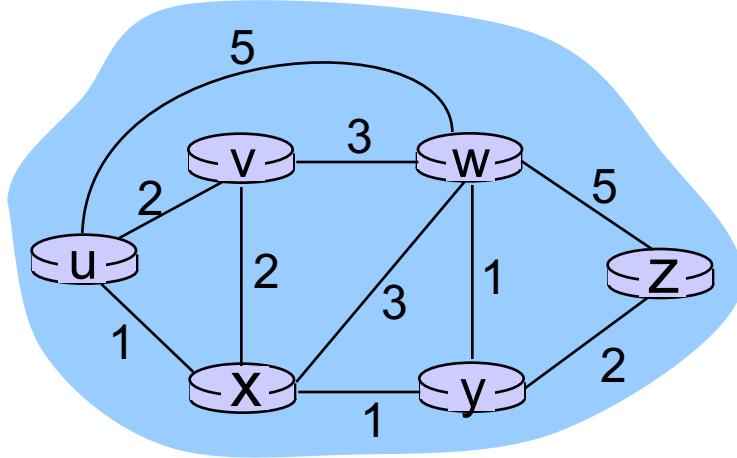
graph:  $G = (N, E)$

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

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

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  $v$  \*/

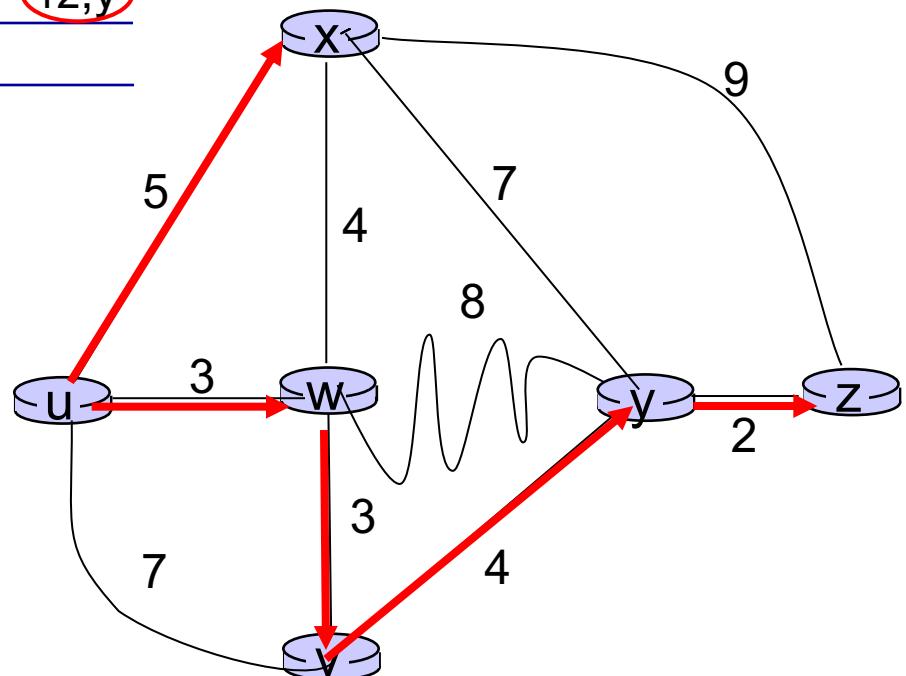
15 **until all nodes in  $N'$**

# Dijkstra's algorithm: example

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

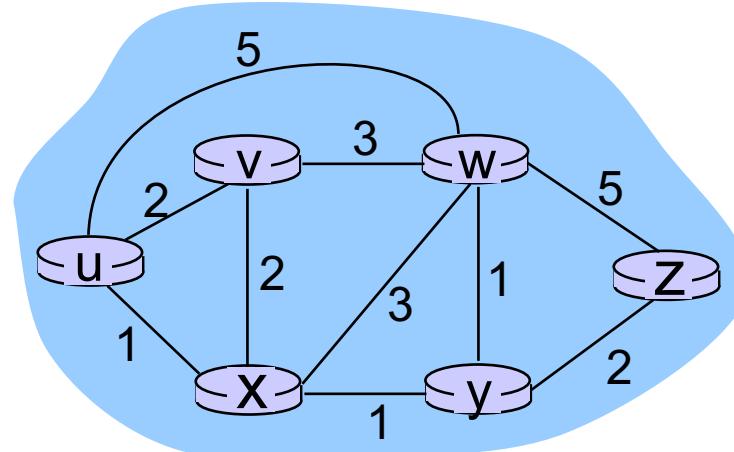
*notes:*

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



# Dijkstra's algorithm: another example

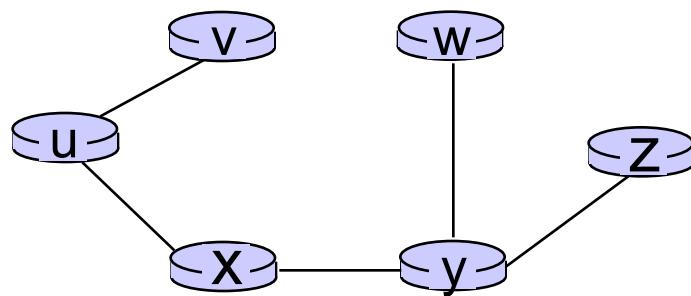
Step	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	$\infty$	$\infty$
1	ux	2,u	4,x		2,x	$\infty$
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



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

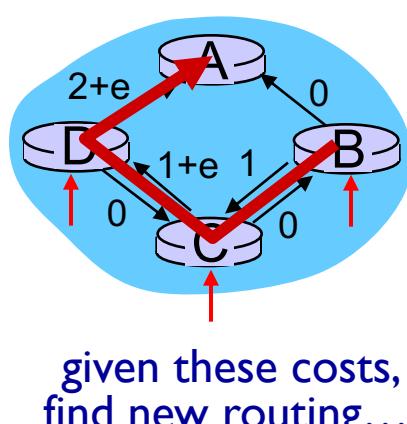
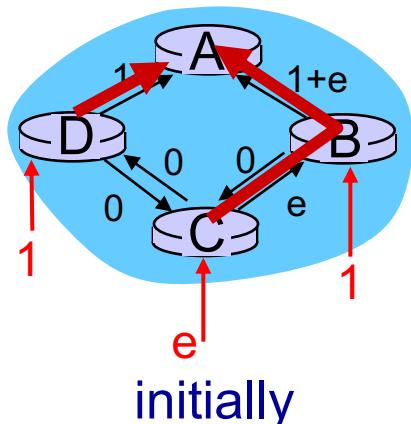
# Dijkstra's algorithm, discussion

*algorithm complexity:* n nodes

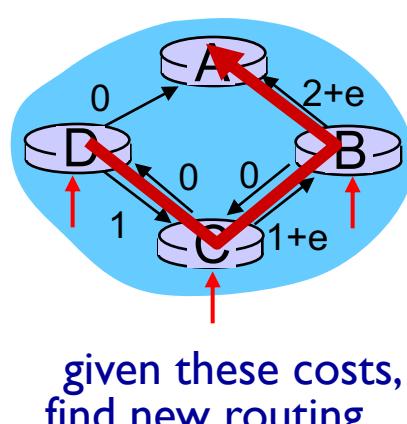
- each iteration: need to check all nodes, w, not in N
- $n(n+1)/2$  comparisons:  $O(n^2)$
- more efficient implementations possible:  $O(n \log n)$

*oscillations possible:*

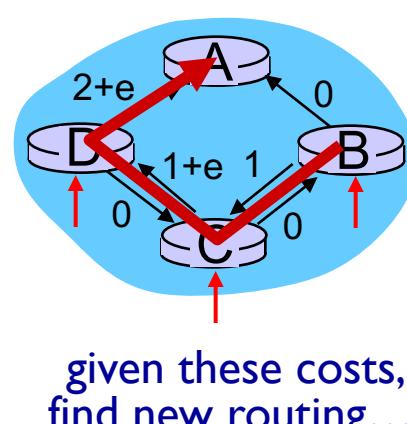
- e.g., support link cost equals amount of carried traffic:



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

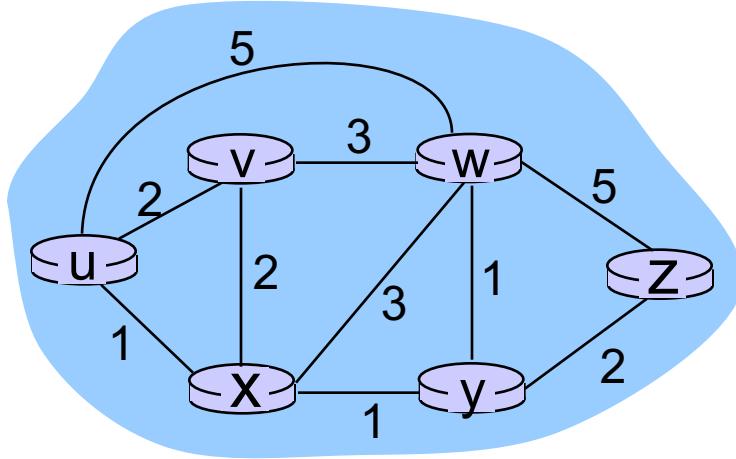
$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

then

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

|            |            |  
|            |            |  
 $v$            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:*

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

*recompute* estimates

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

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

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

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

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

**node x  
table**

	x	y	z
x	0	2	7
y	$\infty$	$\infty$	$\infty$
z	$\infty$	$\infty$	$\infty$

**node y  
table**

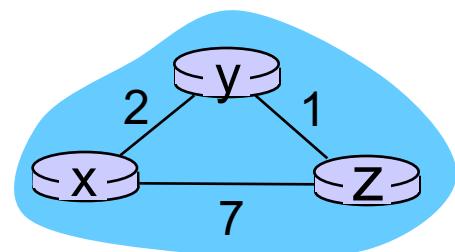
	x	y	z
x	$\infty$	$\infty$	$\infty$
y	2	0	1
z	$\infty$	$\infty$	$\infty$

**node z  
table**

	x	y	z
x	$\infty$	$\infty$	$\infty$
y	$\infty$	$\infty$	$\infty$
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
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**

	x	y	z
from	0	2	7
x	0	2	7
y	$\infty$	$\infty$	$\infty$
z	$\infty$	$\infty$	$\infty$

**node y  
table**

	x	y	z
from	$\infty$	$\infty$	$\infty$
x	$\infty$	$\infty$	$\infty$
y	2	0	1
z	$\infty$	$\infty$	$\infty$

**node z  
table**

	x	y	z
from	$\infty$	$\infty$	$\infty$
x	$\infty$	$\infty$	$\infty$
y	$\infty$	$\infty$	$\infty$
z	7	1	0

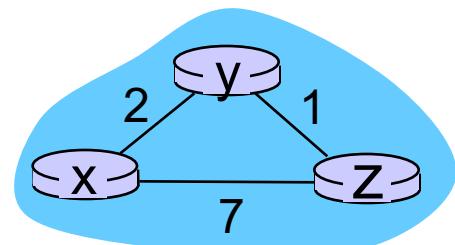
	x	y	z
from	0	2	3
x	0	2	3
y	2	0	1
z	7	1	0

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

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

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

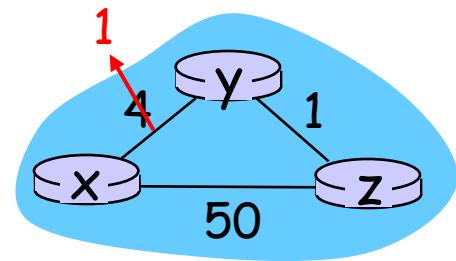
time



# 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



**“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.

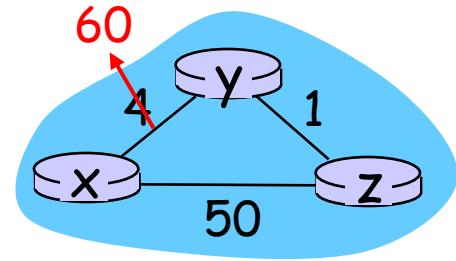
$t_2$ : y receives z's update, updates its distance table. y's least costs do *not* change, so y does *not* send a message to z.

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

# Distance vector: 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?

# Distance Vector: link cost increases

$$\begin{aligned}
 D_y(x) &= \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} \\
 &= \min\{60+0, 1+5\} = 6
 \end{aligned}$$

node y table

	cost to		
	x	y	z
y	4	0	1
x	0	4	5
z	5	1	0

when y detects

	cost to		
	x	y	z
y	6	0	1
x			
z			

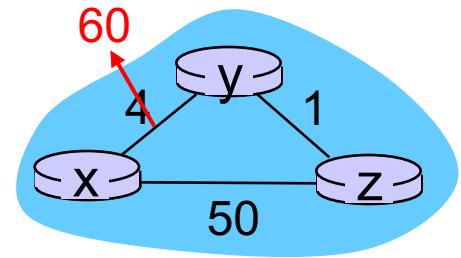
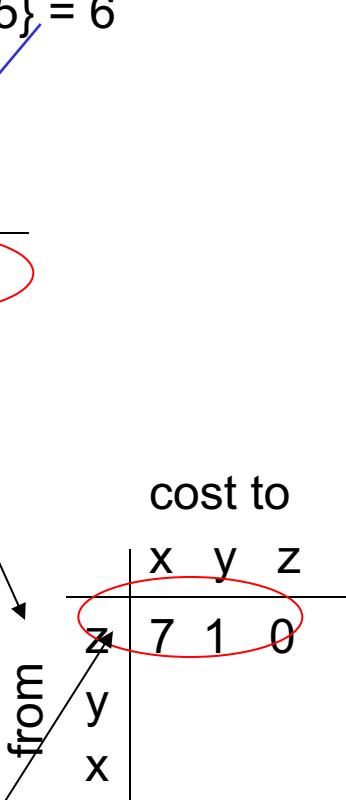
node z table

	cost to		
	x	y	z
z	5	1	0
y	4	0	1
x	0	4	5

node x table

	cost to		
	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

$$\begin{aligned}
 D_z(x) &= \min\{c(z,y) + D_y(x), c(z,x) + D_x(x)\} \\
 &= \min\{1+6, 50+0\} = 7
 \end{aligned}$$



Network Layer

# Distance Vector: link cost increases

## node y table

cost to		x	y	z
from		4	0	1
y		6	0	1
x		0	4	5
z		5	1	0

when y detects

cost to		x	y	z
from		6	0	1
y		6	0	1
x		6	0	1
z		6	0	1

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

$$= \min\{60+0, 1+7\} = 8$$

cost to

cost to		x	y	z
from		8	0	1
y		8	0	1
x		8	0	1
z		8	0	1

## node z table

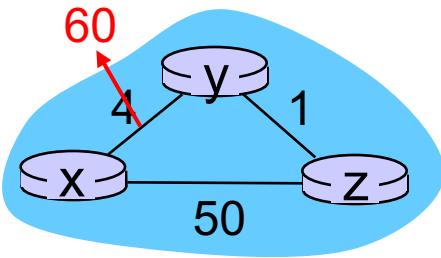
cost to		x	y	z
from		5	1	0
z		5	1	0
y		4	0	1
x		0	4	5

## node x table

cost to		x	y	z
from		0	4	5
x		0	4	5
y		4	0	1
z		5	1	0

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

$$= \min\{1+8, 50+0\} = 9$$



Network Layer

# 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”)

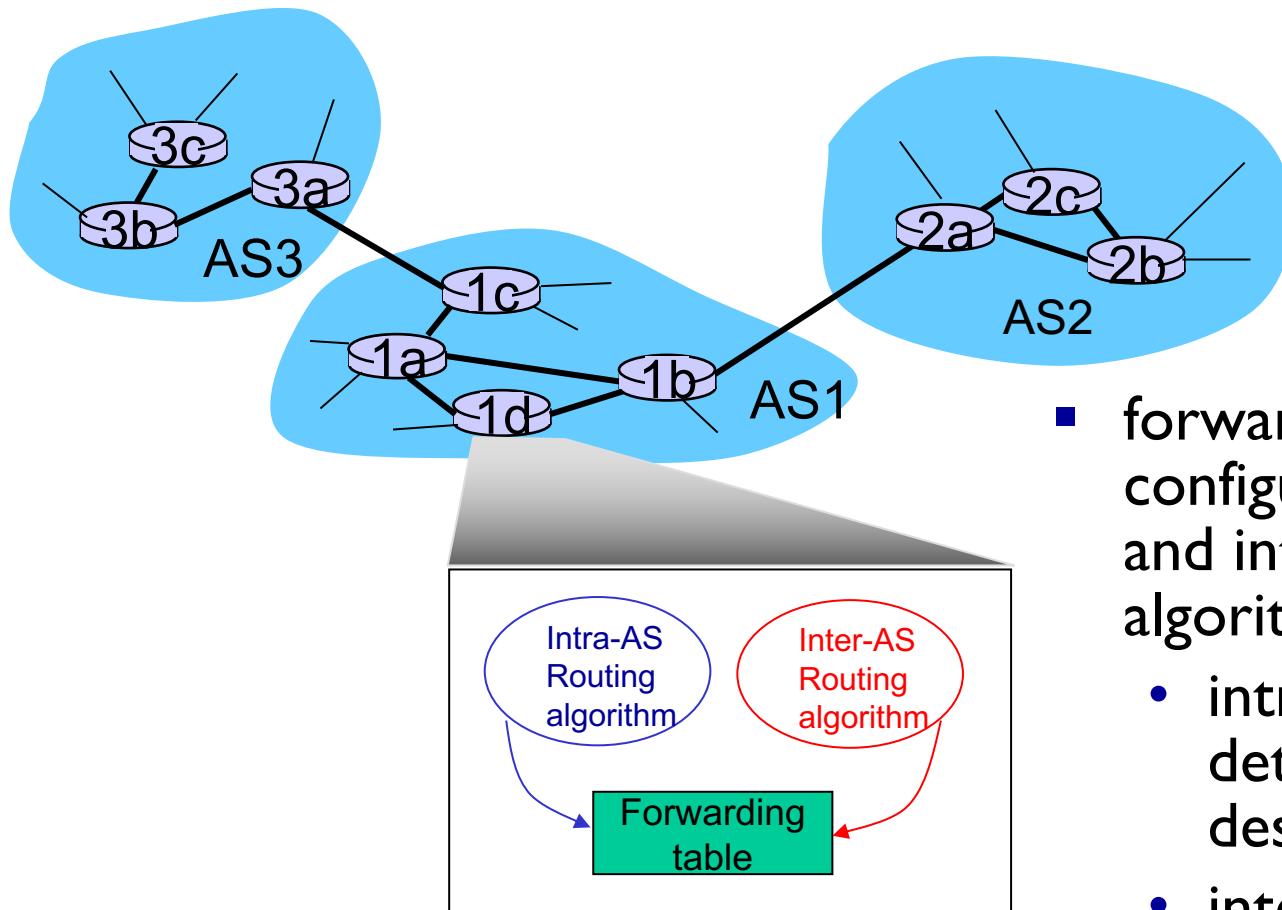
## intra-AS routing

- 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

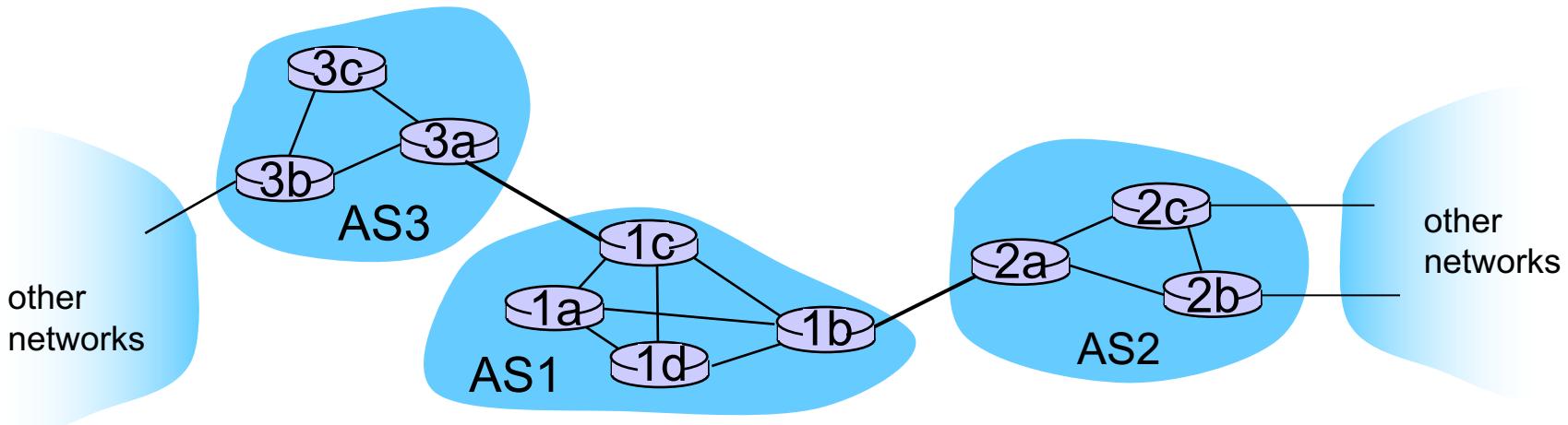
# Inter-AS tasks

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

# AS1 must:

- I. learn which dests are reachable through AS2, which through AS3
  2. propagate this reachability info to all routers in AS1

## *job of inter-AS routing!*



# Intra-AS Routing

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

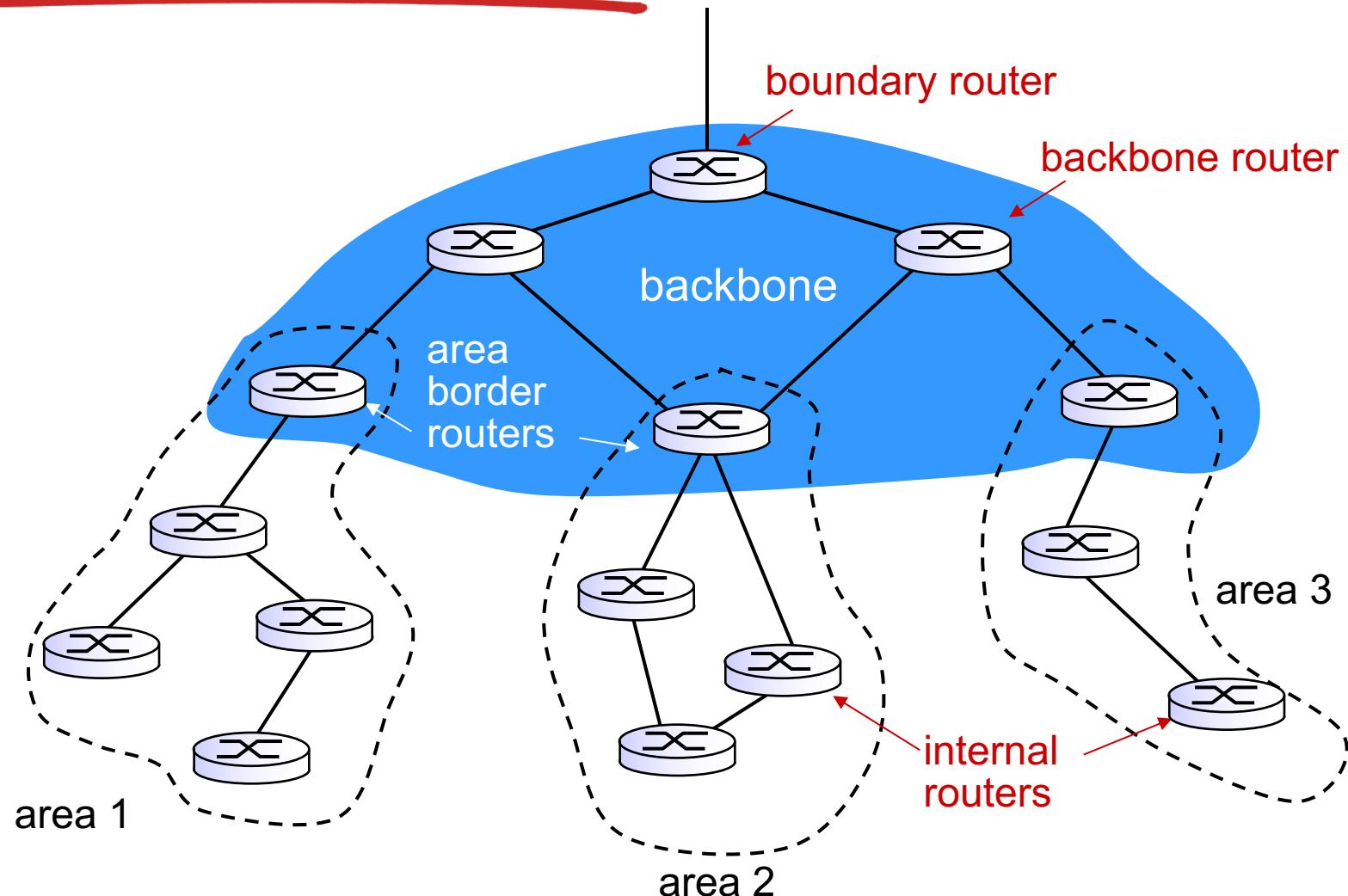
# OSPF (Open Shortest Path First)

- “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 node 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

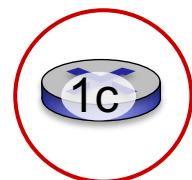
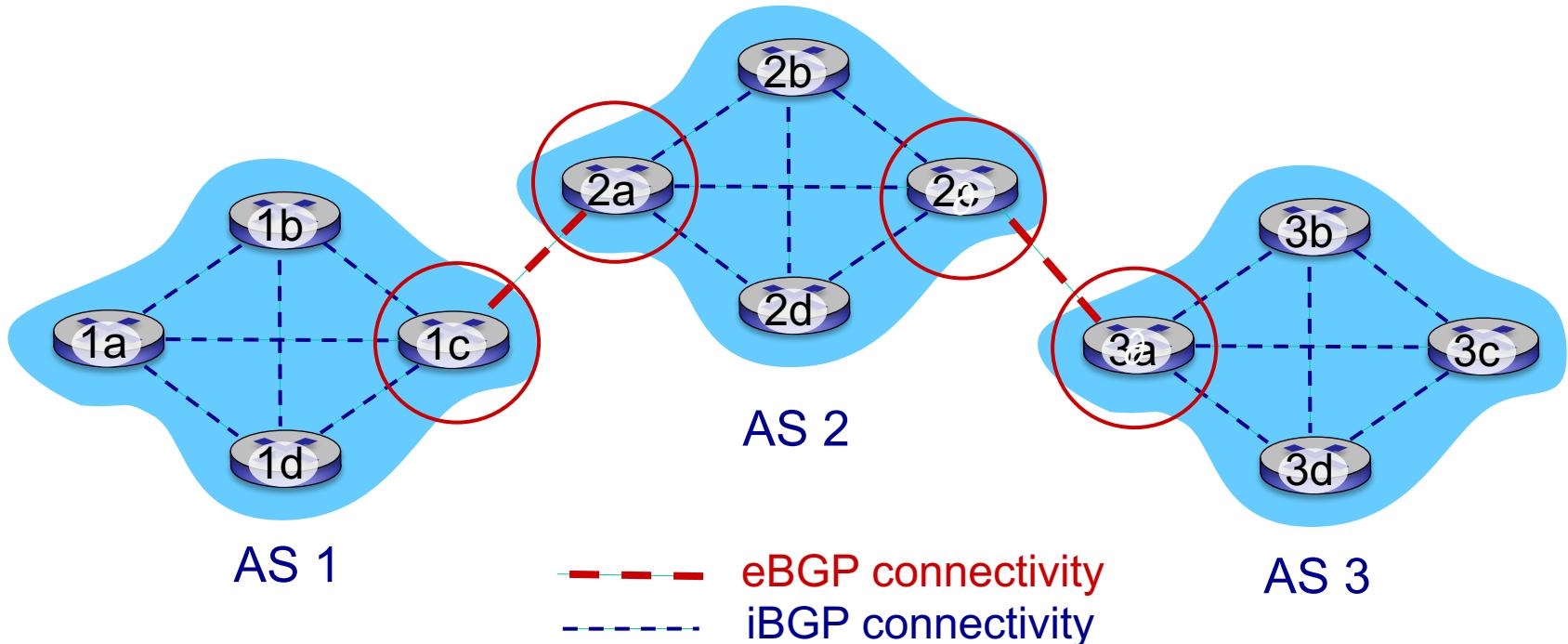
5.6 ICMP: The Internet Control Message Protocol

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*”

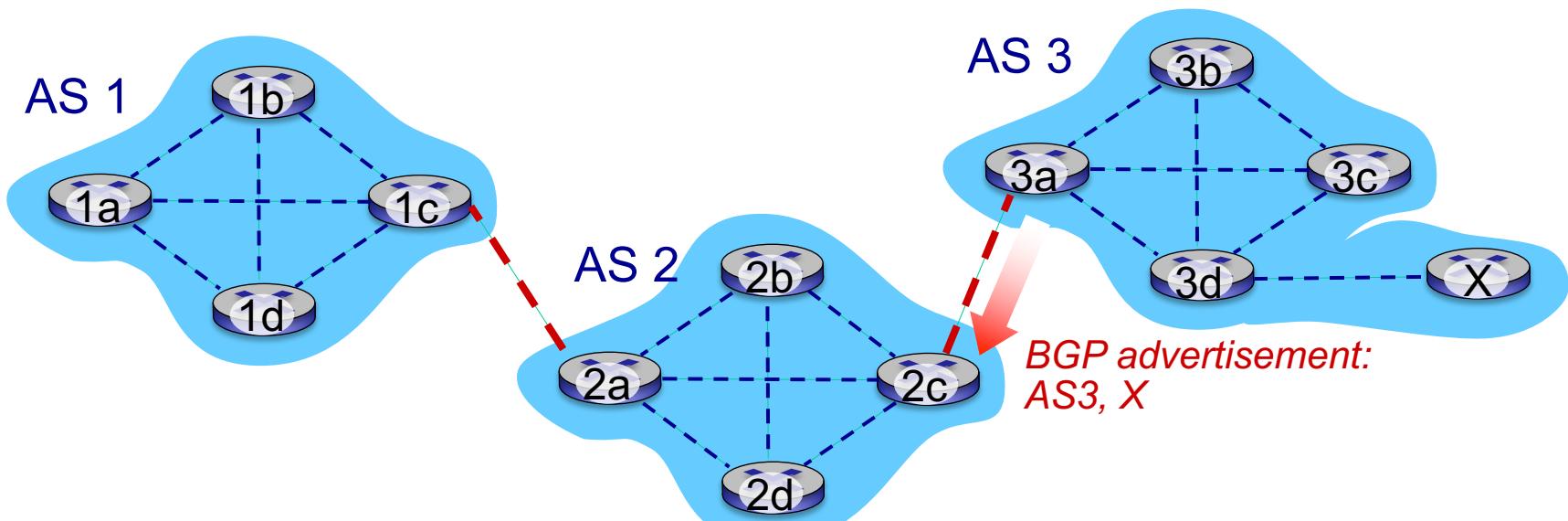
# eBGP, iBGP connections



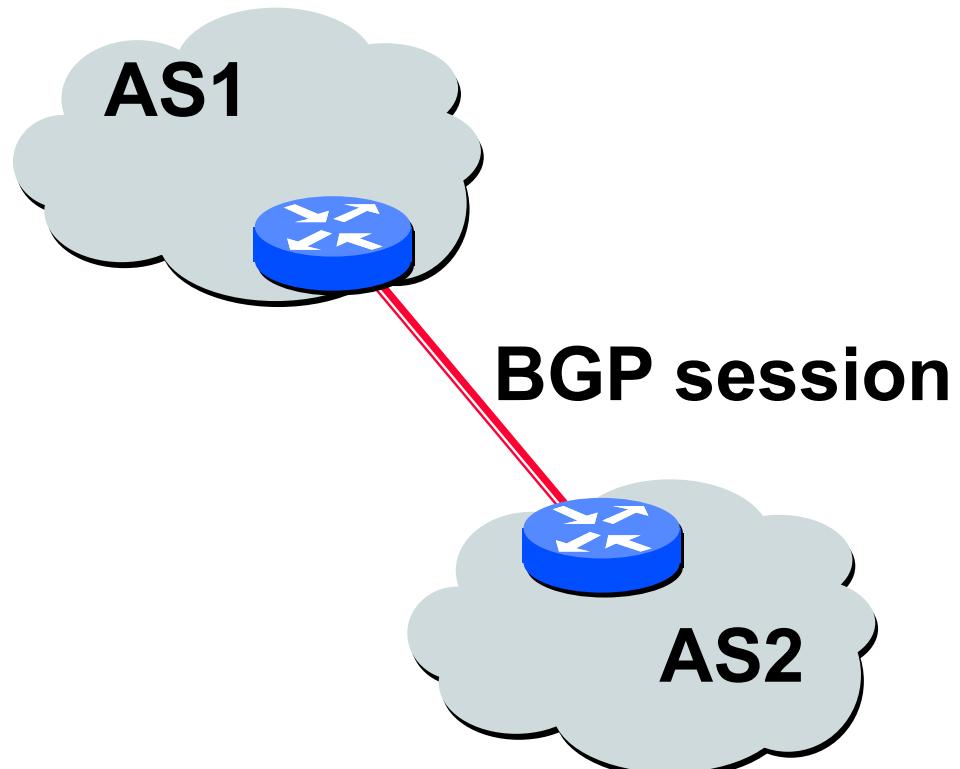
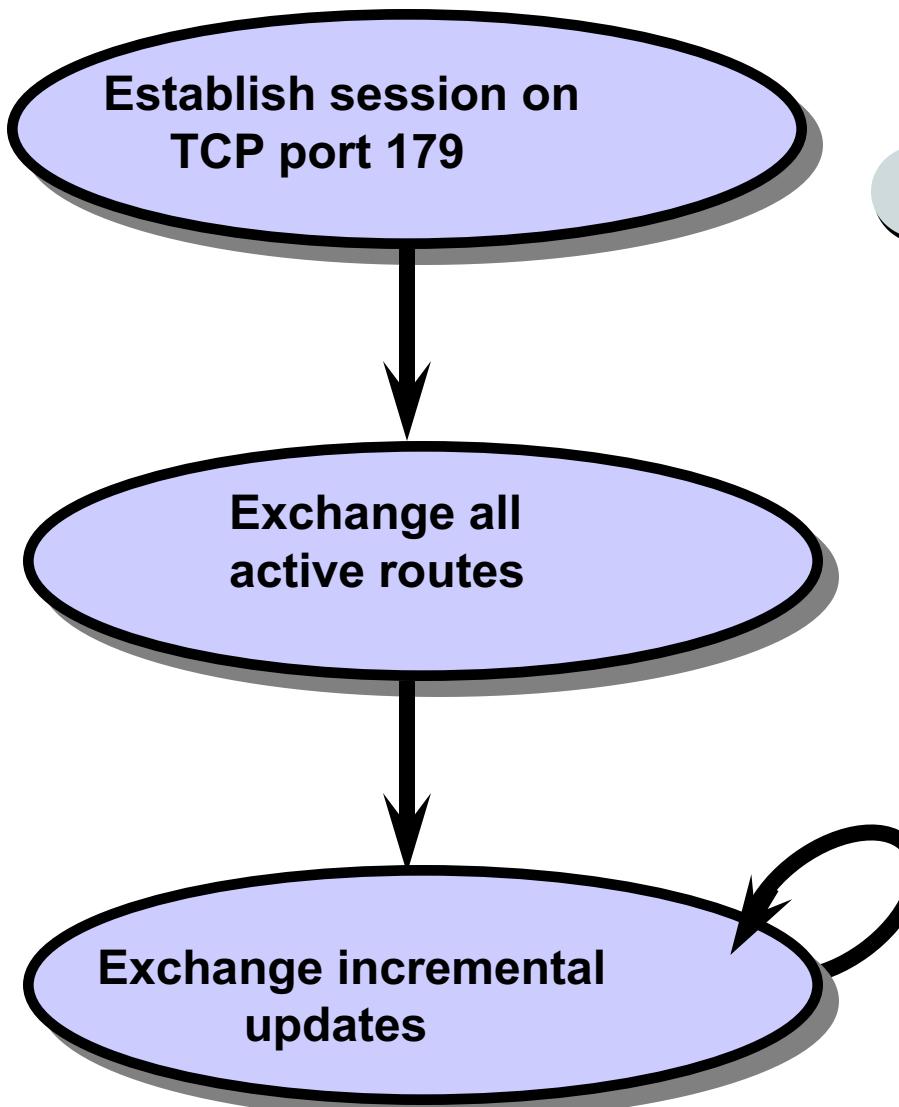
gateway routers run both eBGP and iBGP protocols

# 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



# Simplified BGP Operations



Network Layer

# AS Numbers (ASNs)

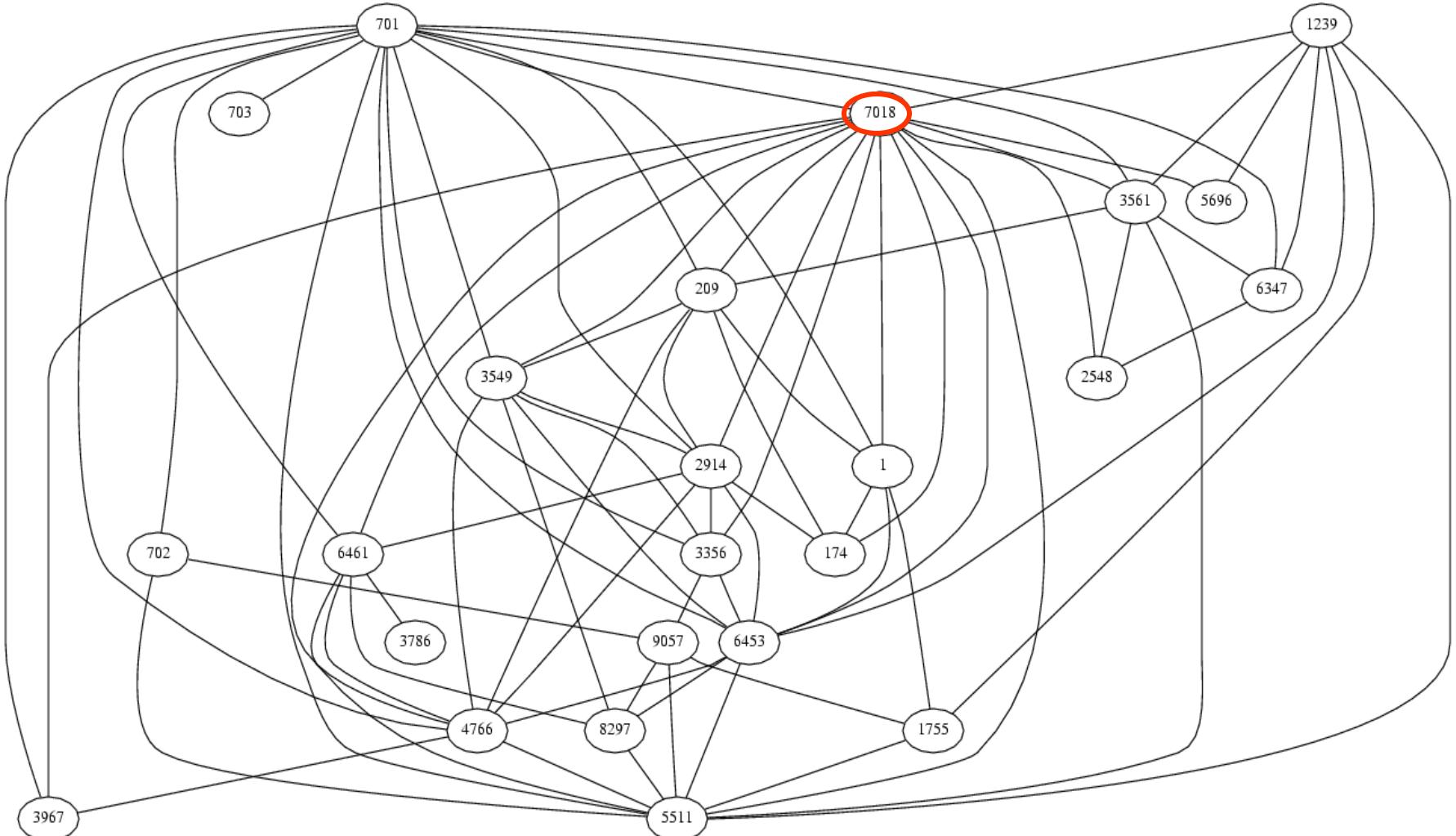
**ASNs are 16 bit values.**

**64512 through 65535 are “private”**

- **Genuity: 1**
- **MIT: 3**
- **JANET: 786**
- **UC San Diego: 7377**
- **AT&T: 7018, 6341, 5074, ...**
- **UUNET: 701, 702, 284, 12199, ...**
- **Sprint: 1239, 1240, 6211, 6242, ...**
- **...**

**ASNs represent units of routing policy**

# AS Graphs Can Be Fun

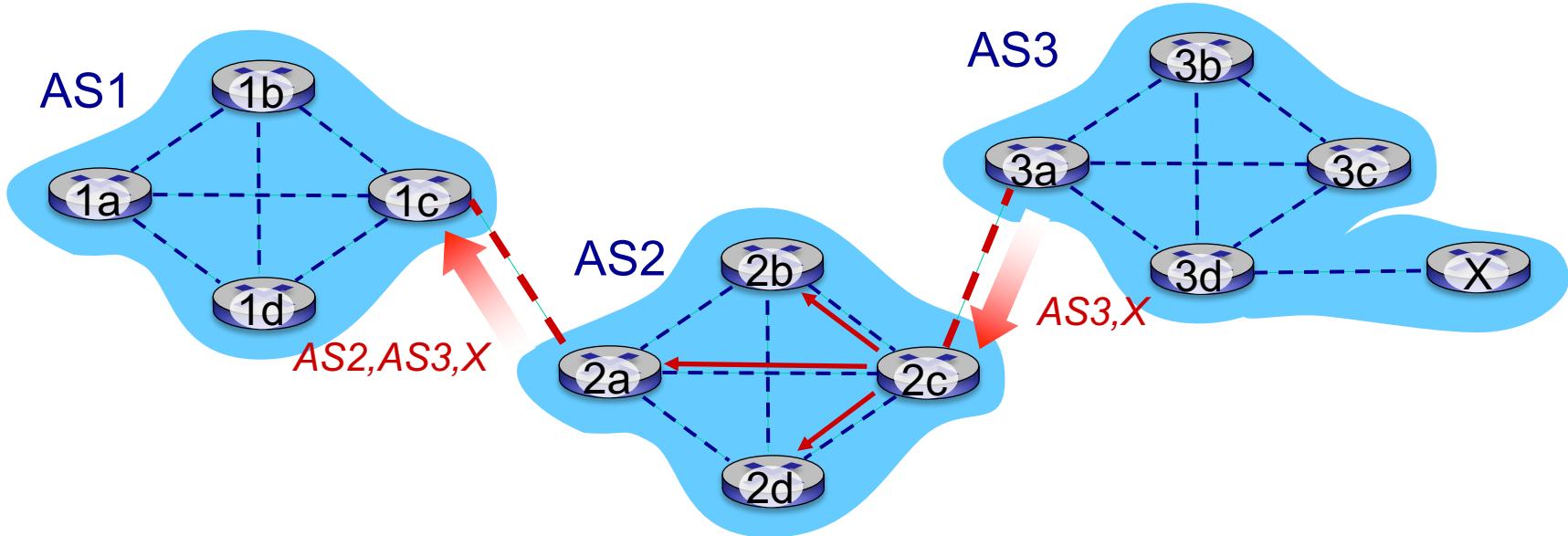


**The subgraph showing all ASes that have more than 100 neighbors in full graph of 11,158 nodes. July 6, 2001. Point of view: AT&T route-server**

# Path attributes and BGP routes

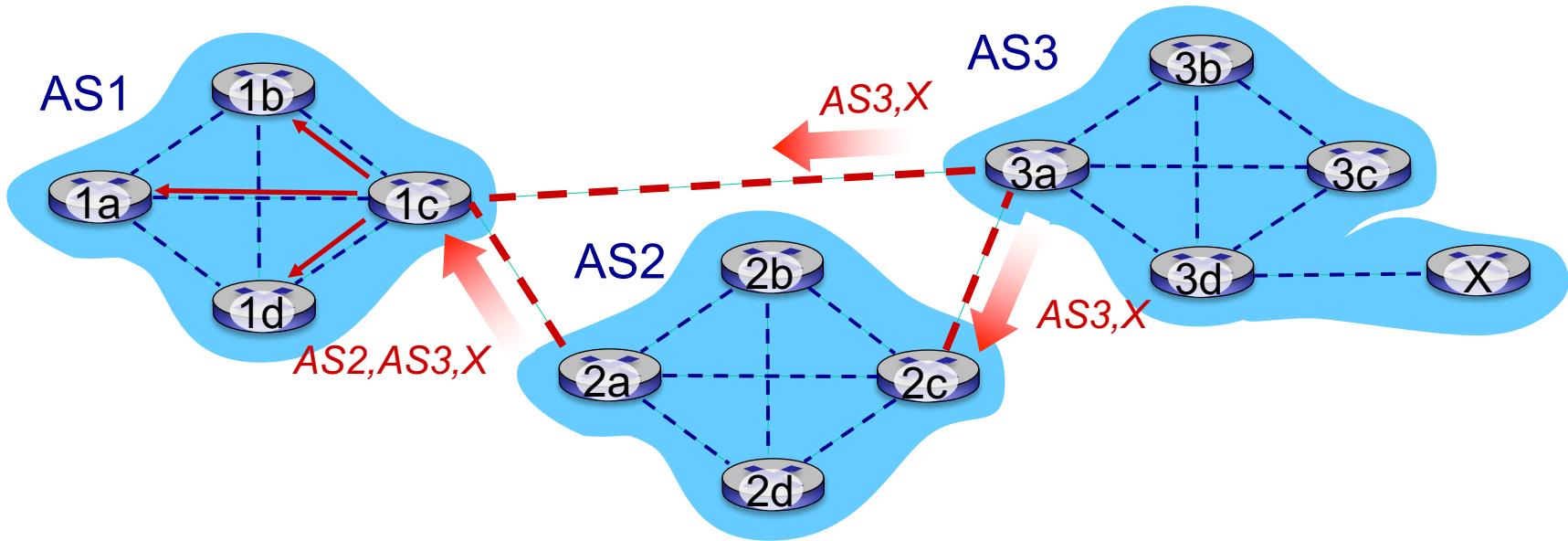
- advertised prefix includes BGP attributes
  - prefix + attributes = “route”
- two important attributes:
  - **AS-PATH**: list of ASes through which prefix advertisement has passed
  - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
- *Policy-based routing*:
  - gateway receiving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

```
show ip bgp
BGP table version is 111849680, local router ID is 203.62.248.4
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

Network          Next Hop           Metric LocPrf Weight Path
*->i192.35.25.0 134.159.0.1      50      0 16779 1 701 703 i
*->i192.35.29.0 166.49.251.25    50      0 5727 7018 14541 i
*->i192.35.35.0 134.159.0.1      50      0 16779 1 701 1744 i
*->i192.35.37.0 134.159.0.1      50      0 16779 1 3561 i
*->i192.35.39.0 134.159.0.3      50      0 16779 1 701 80 i
*->i192.35.44.0 166.49.251.25    50      0 5727 7018 1785 i
*->i192.35.48.0 203.62.248.34    55      0 16779 209 7843 225 225 225 225 i
*->i192.35.49.0 203.62.248.34    55      0 16779 209 7843 225 225 225 225 i
*->i192.35.50.0 203.62.248.34    55      0 16779 3549 714 714 714 i
*->i192.35.51.0/25 203.62.248.34  55      0 16779 3549 14744 14744 14744 14744 14744 14744 i
. . .
```

Thanks to Geoff Huston. <http://www.telstra.net/ops> on July 6, 2001

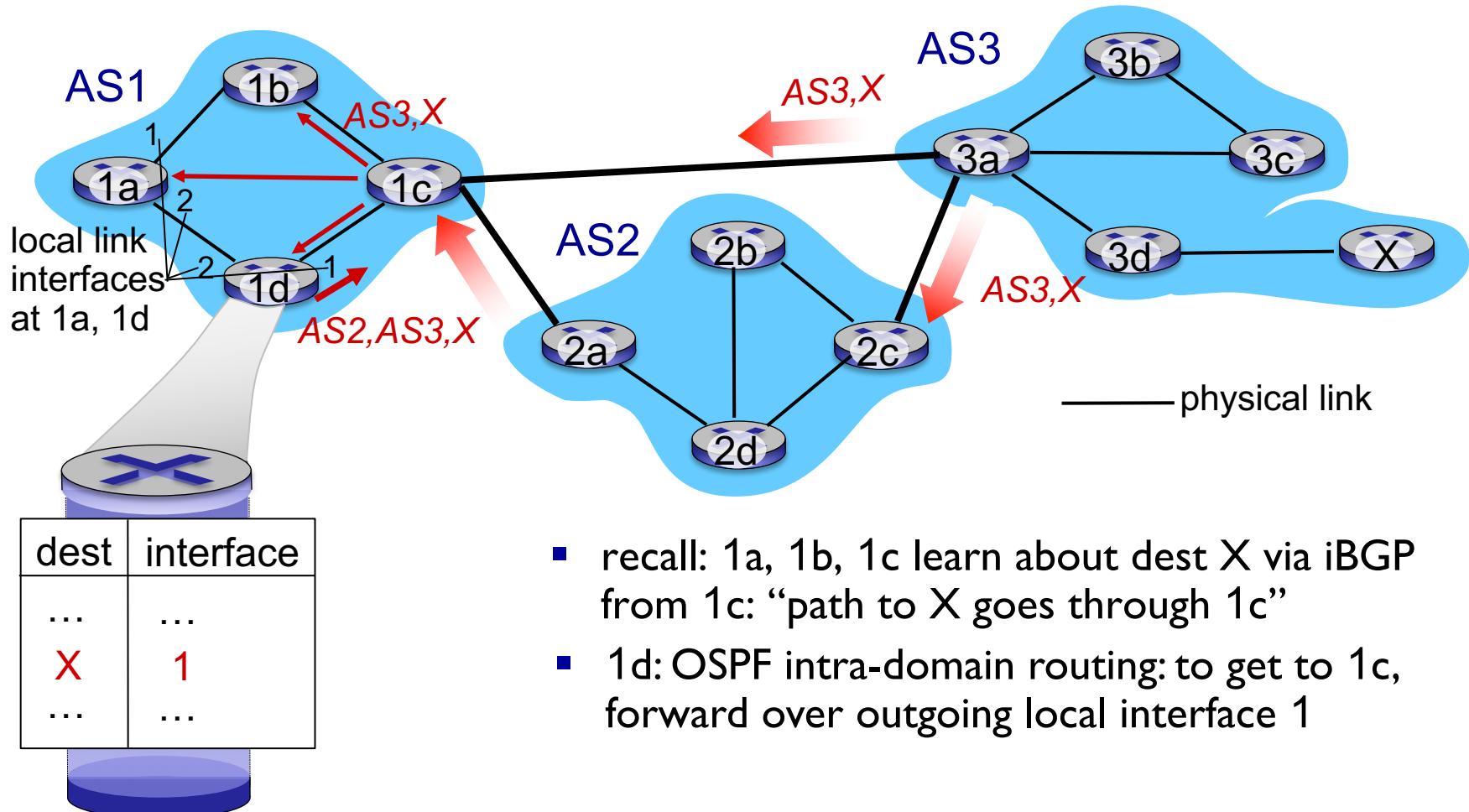
- ❖ Use “whois” queries to associate an ASN with “owner” (for example, <http://www.arin.net/whois/arinfo.html>)
- ❖ 7018 = AT&T Worldnet, 701 = Uunet, 3561 = Cable & Wireless, ...

# BGP messages

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

# BGP, OSPF, forwarding table entries

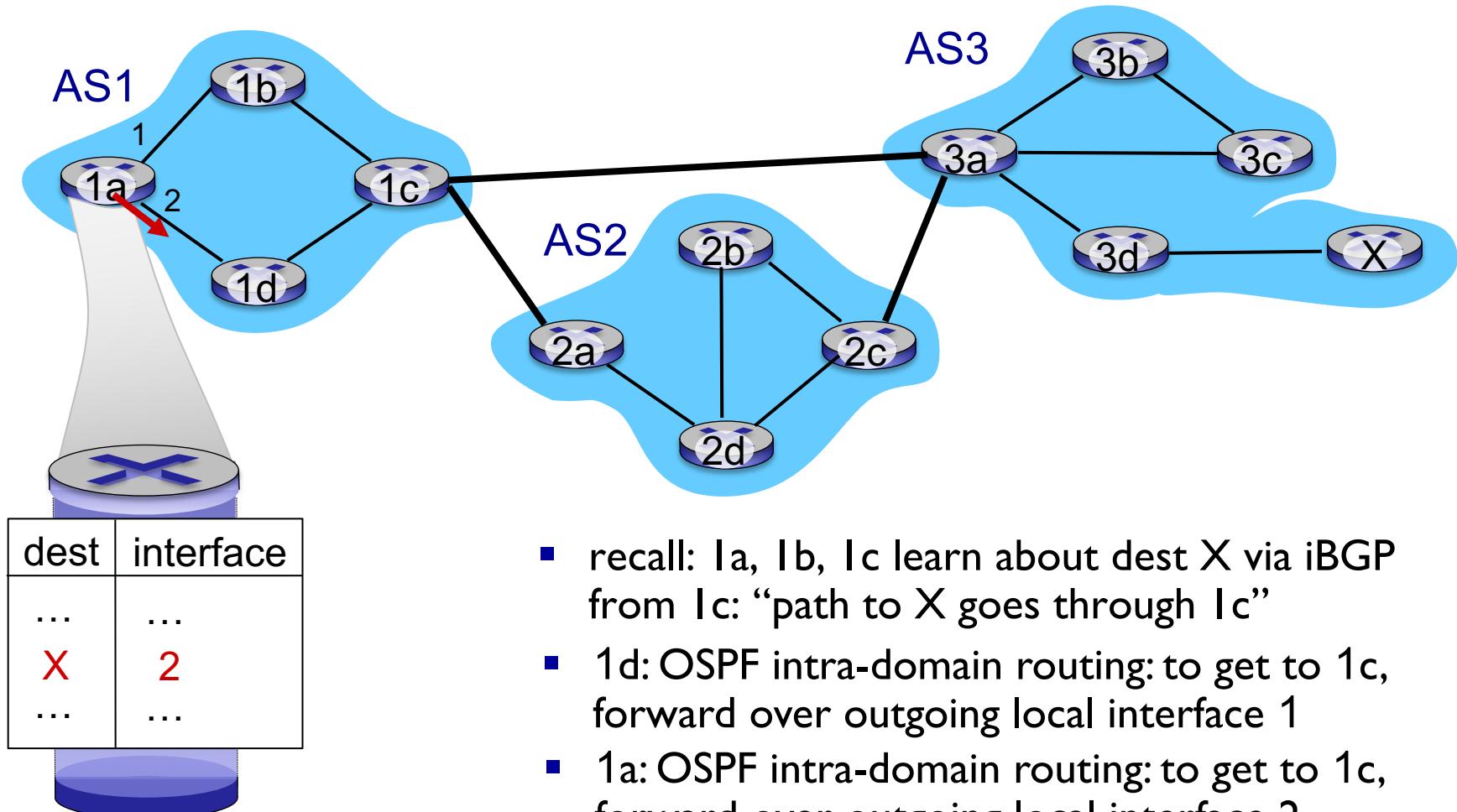
Q: how does router set forwarding table entry to distant prefix?



- 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

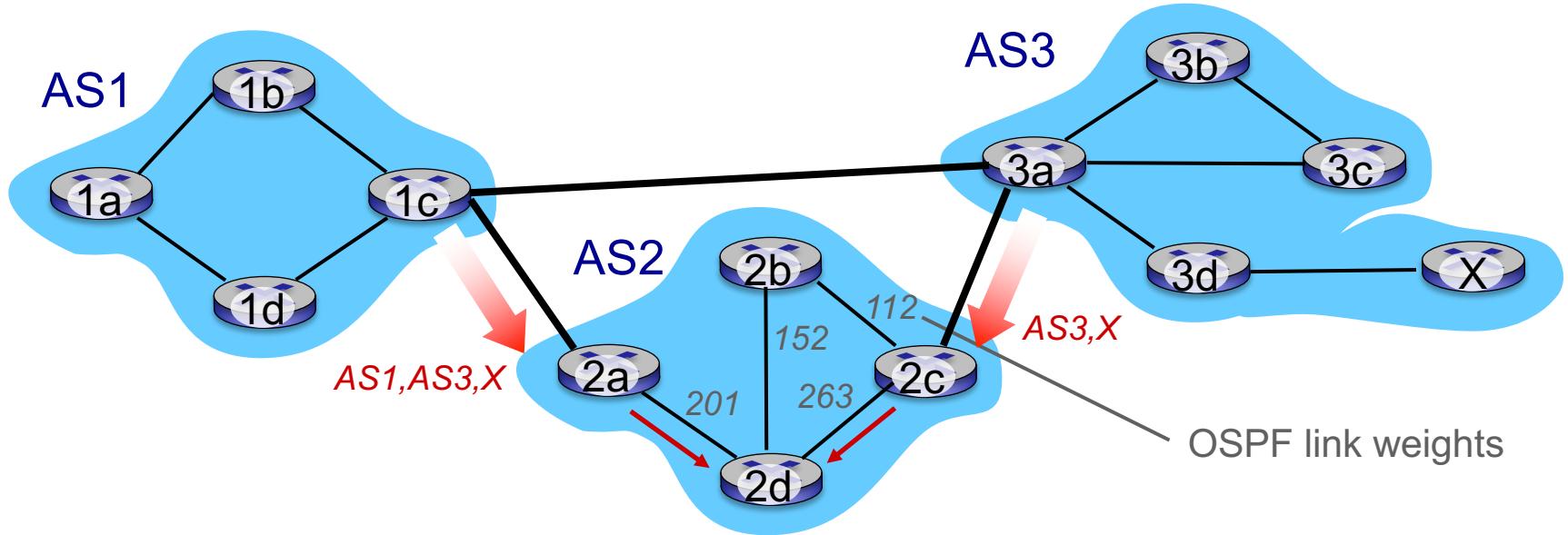
Q: how does router set forwarding table entry to distant prefix?



# BGP route selection

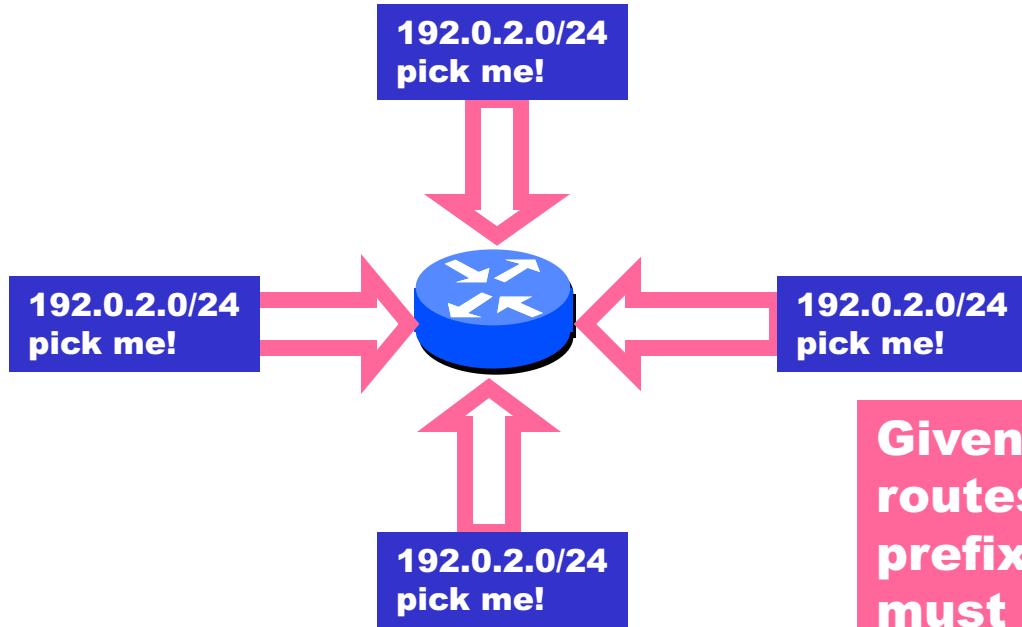
- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

# 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!

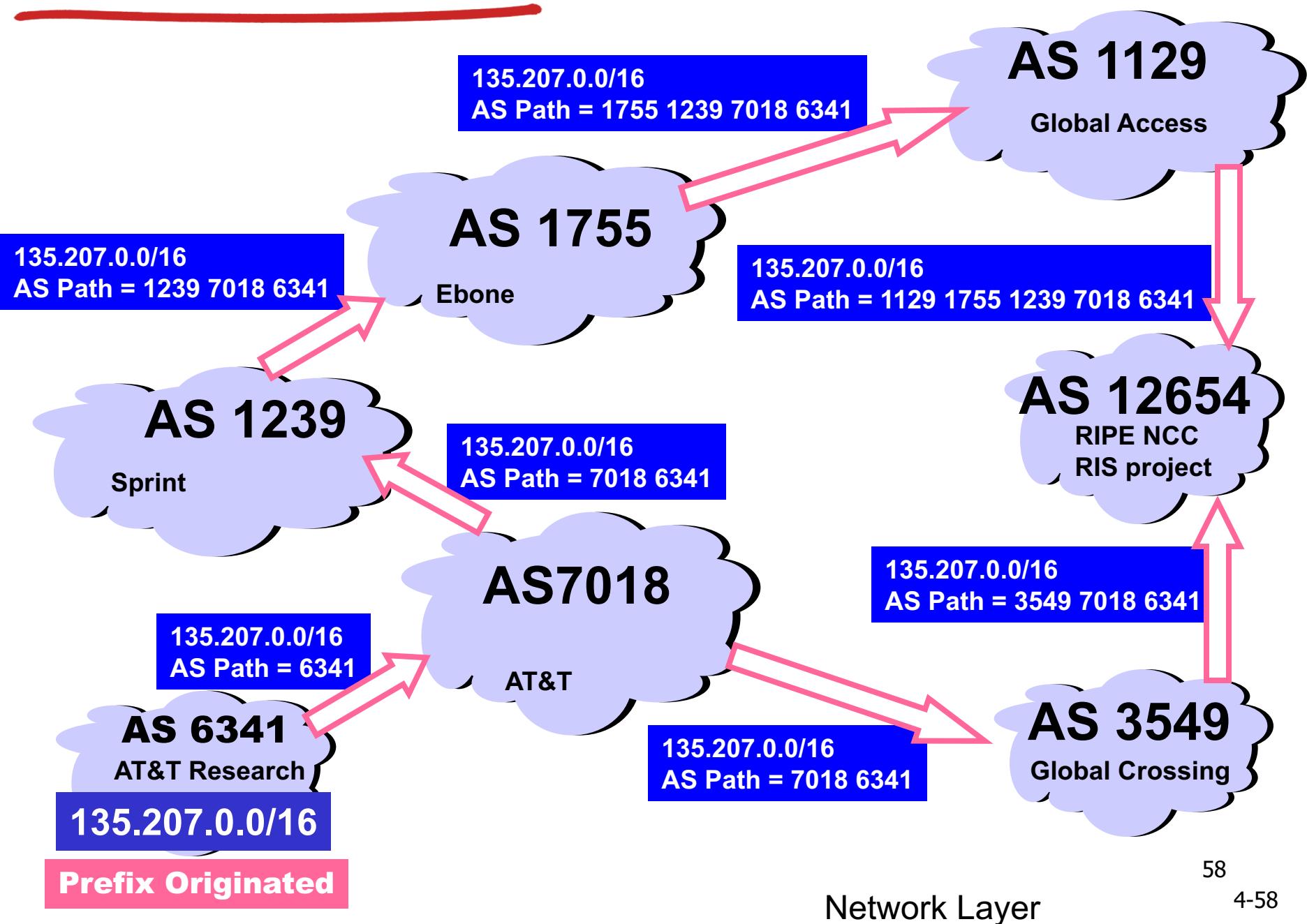
# Attributes are Used to Select Best Routes



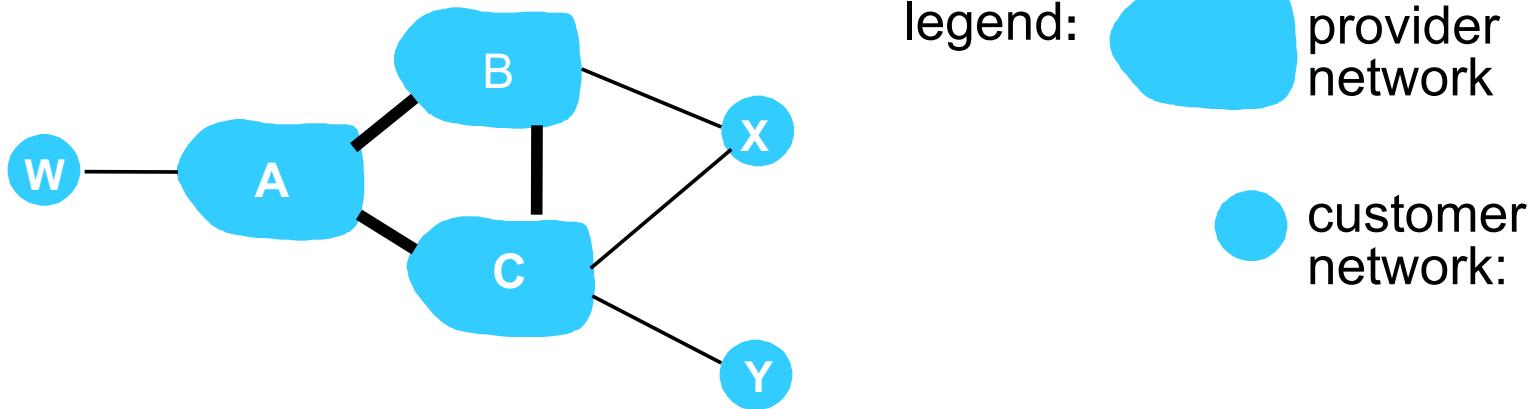
**Given multiple routes to the same prefix, a BGP speaker must pick at most one best route**

**(Note: it could reject them all!)**

# AS PATH Attribute



# BGP: achieving policy via advertisements



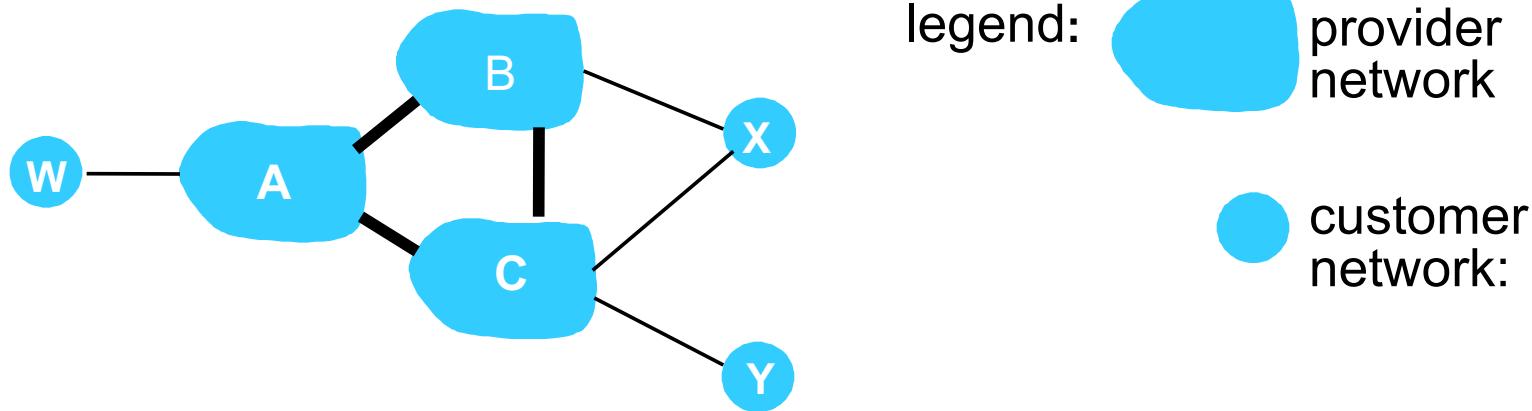
legend:

- provider network
- customer network:

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* BAw to C:
  - B gets no “revenue” for routing CBAw, since none of C, A, w are B’s customers
  - C does not learn about CBAw path
- C will route CAw (not using B) to get to w

# BGP: achieving policy via advertisements



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

# 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

# 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

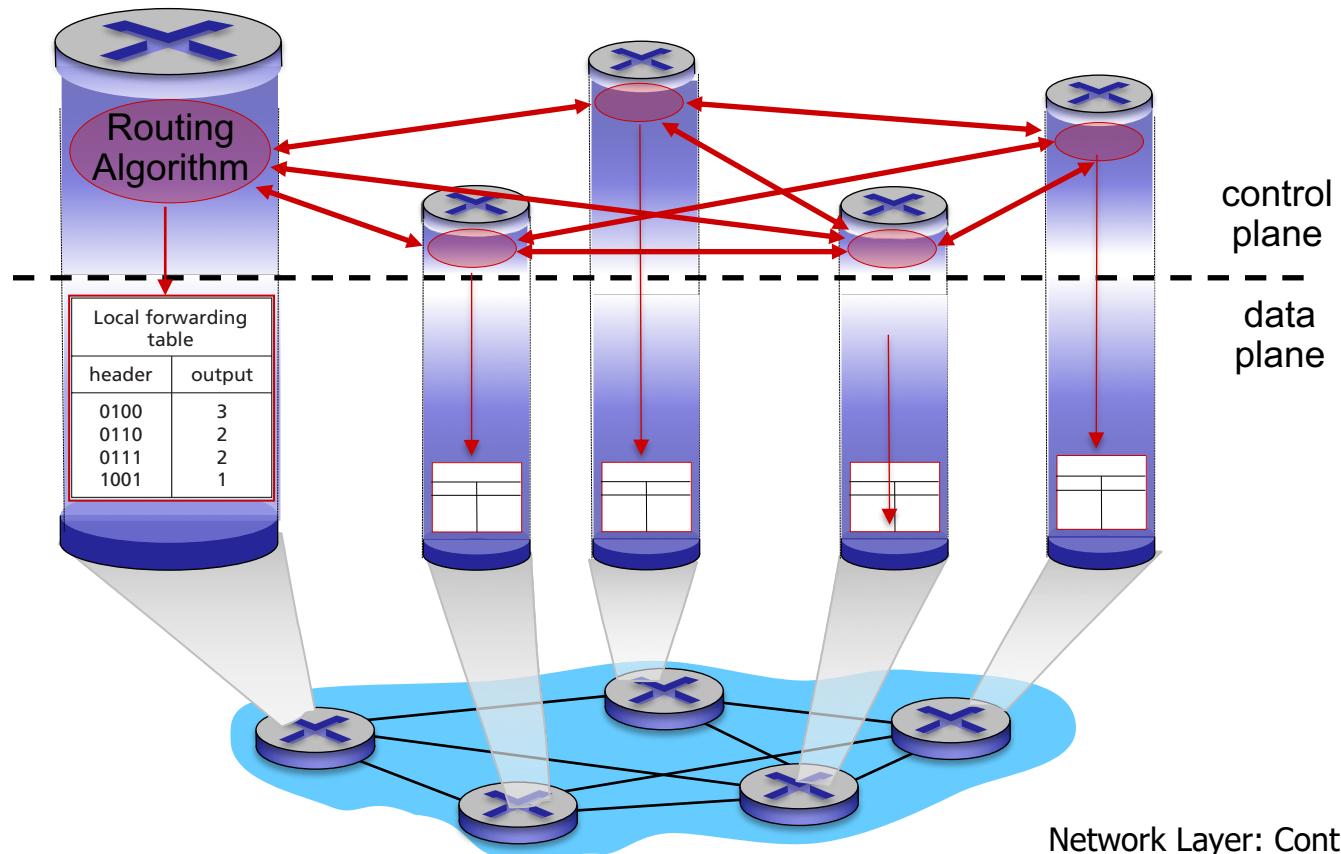
5.7 Network management and SNMP

# 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

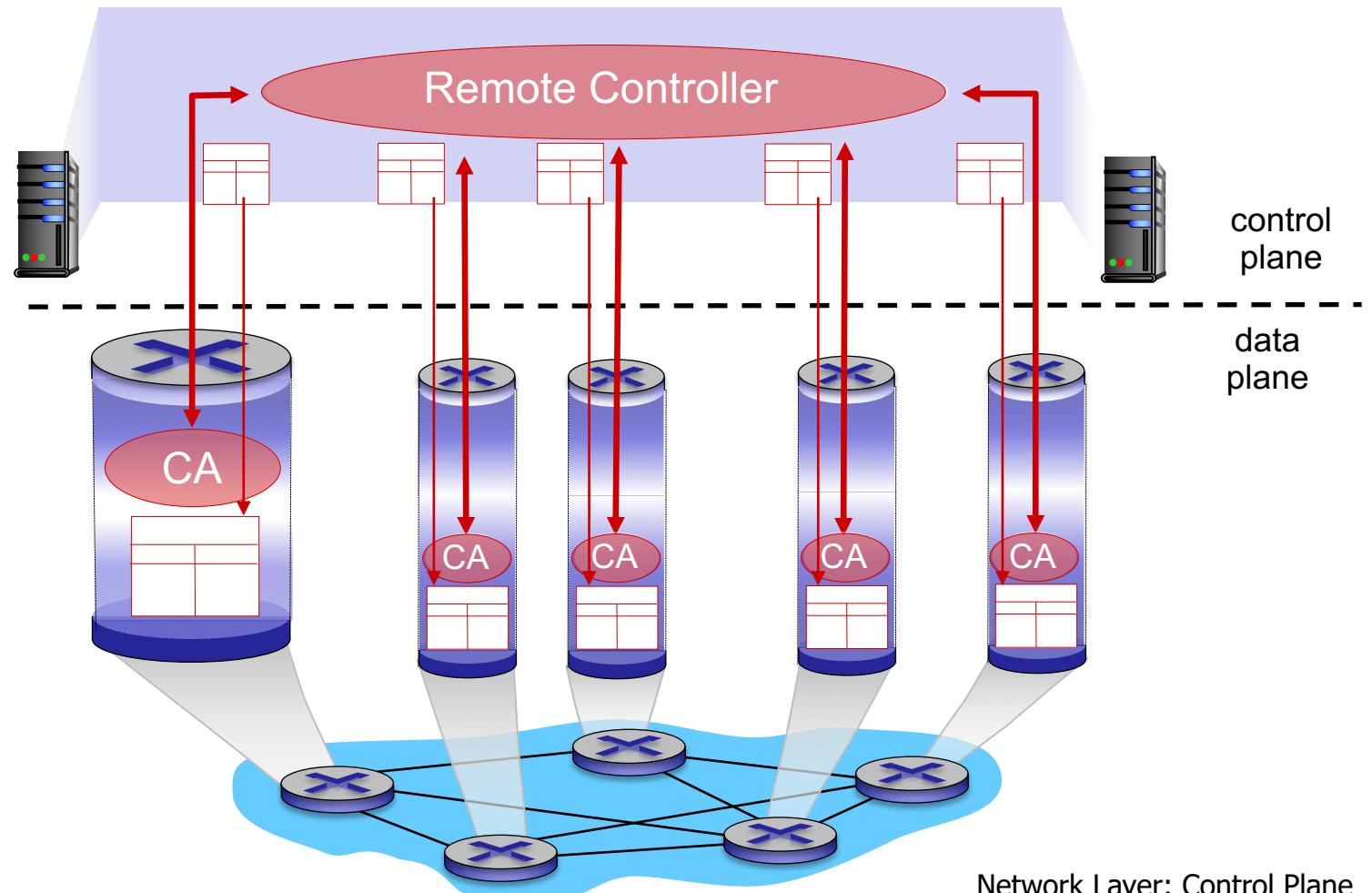
# Recall: per-router control plane

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



# Recall: logically centralized control plane

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

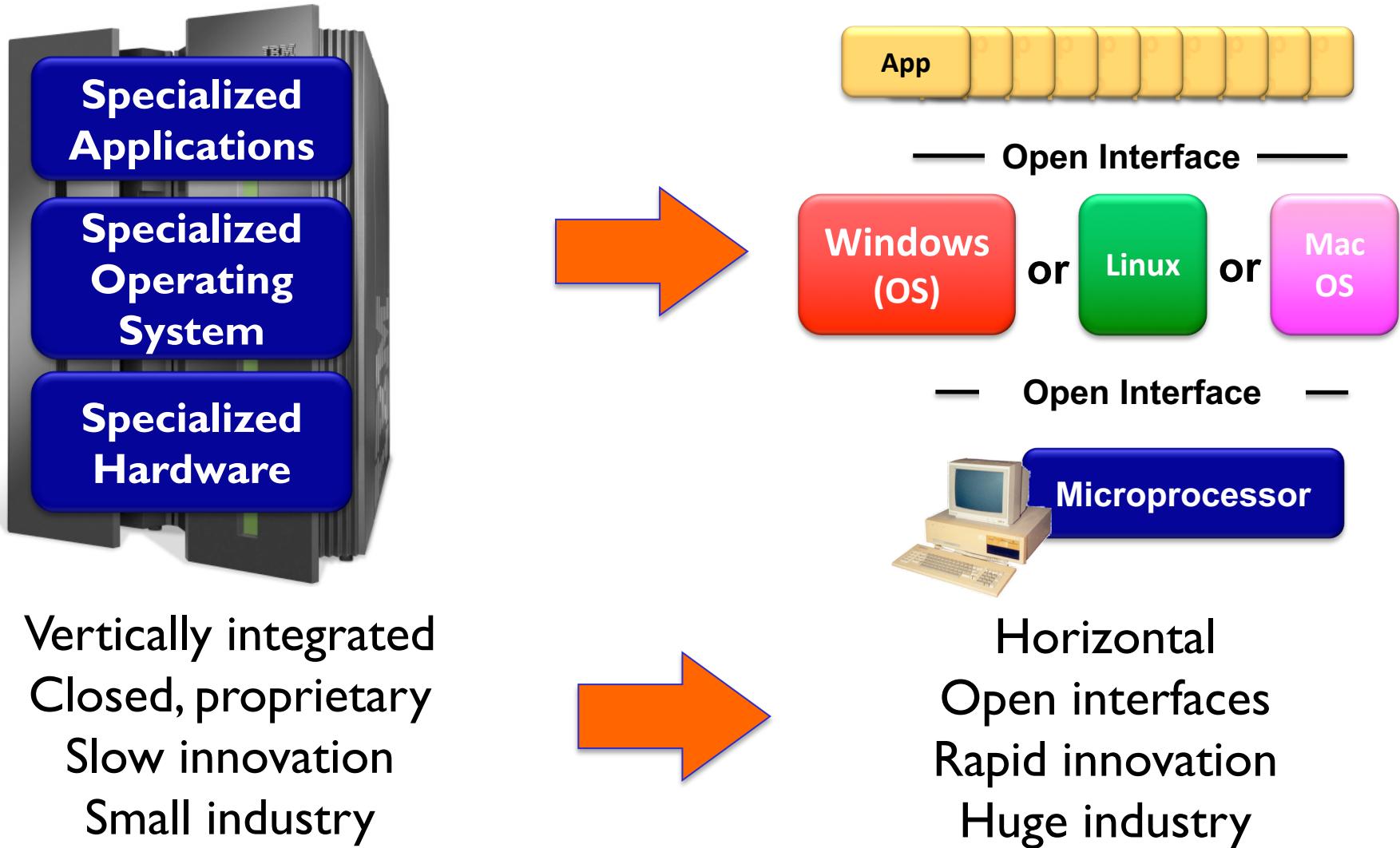


# Software defined networking (SDN)

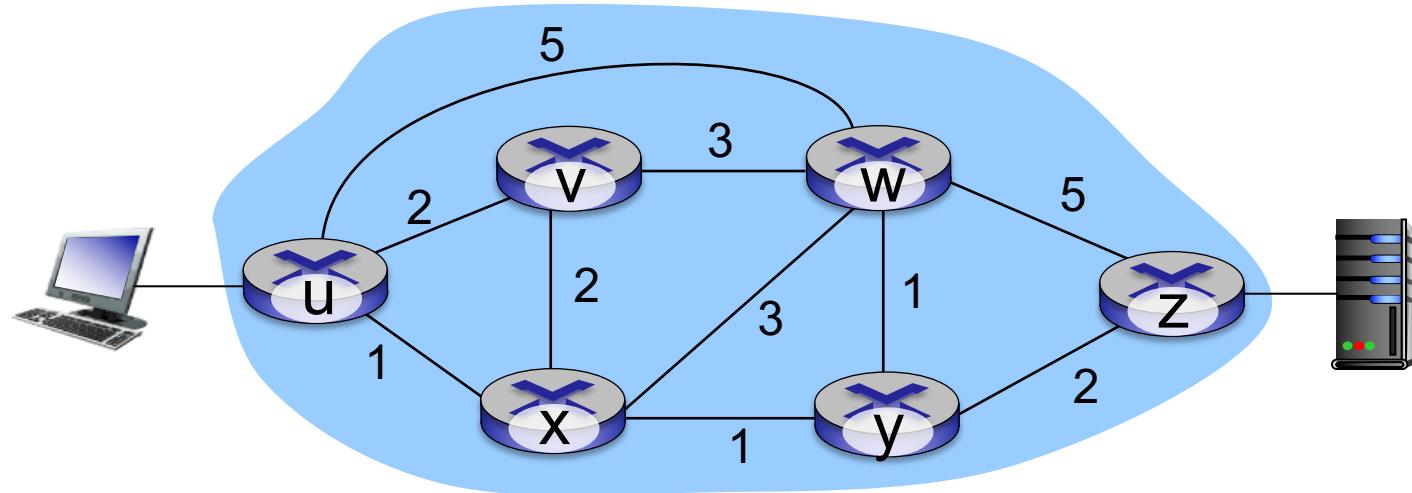
*Why* a *logically centralized* control plane?

- 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

# Analogy: mainframe to PC evolution\*



# Traffic engineering: difficult traditional routing

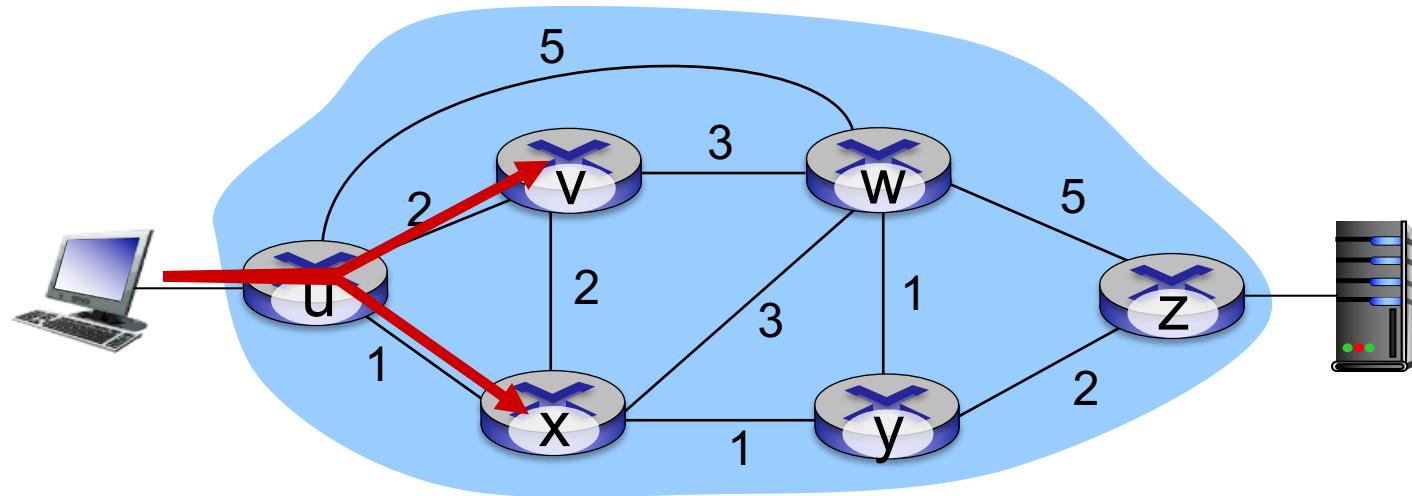


Q: what if network operator wants u-to-z traffic to flow along  $uvwz$ , x-to-z traffic to flow  $xwyz$ ?

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

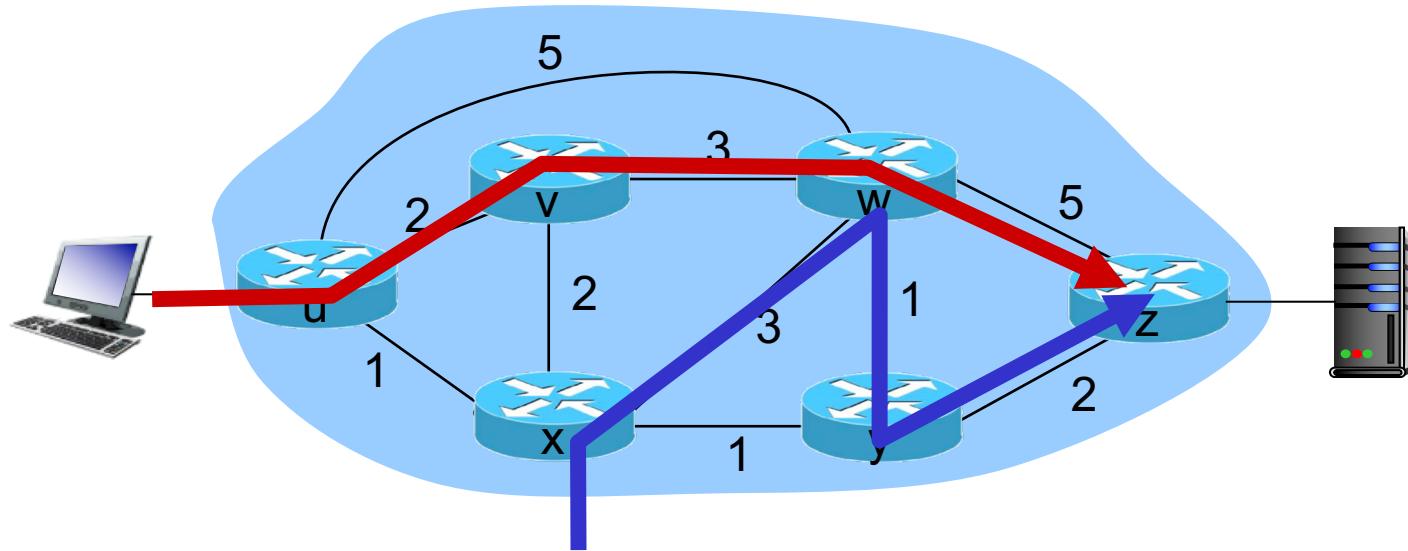
*Link weights are only control “knobs”: wrong!*

# Traffic engineering: difficult



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxzy (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

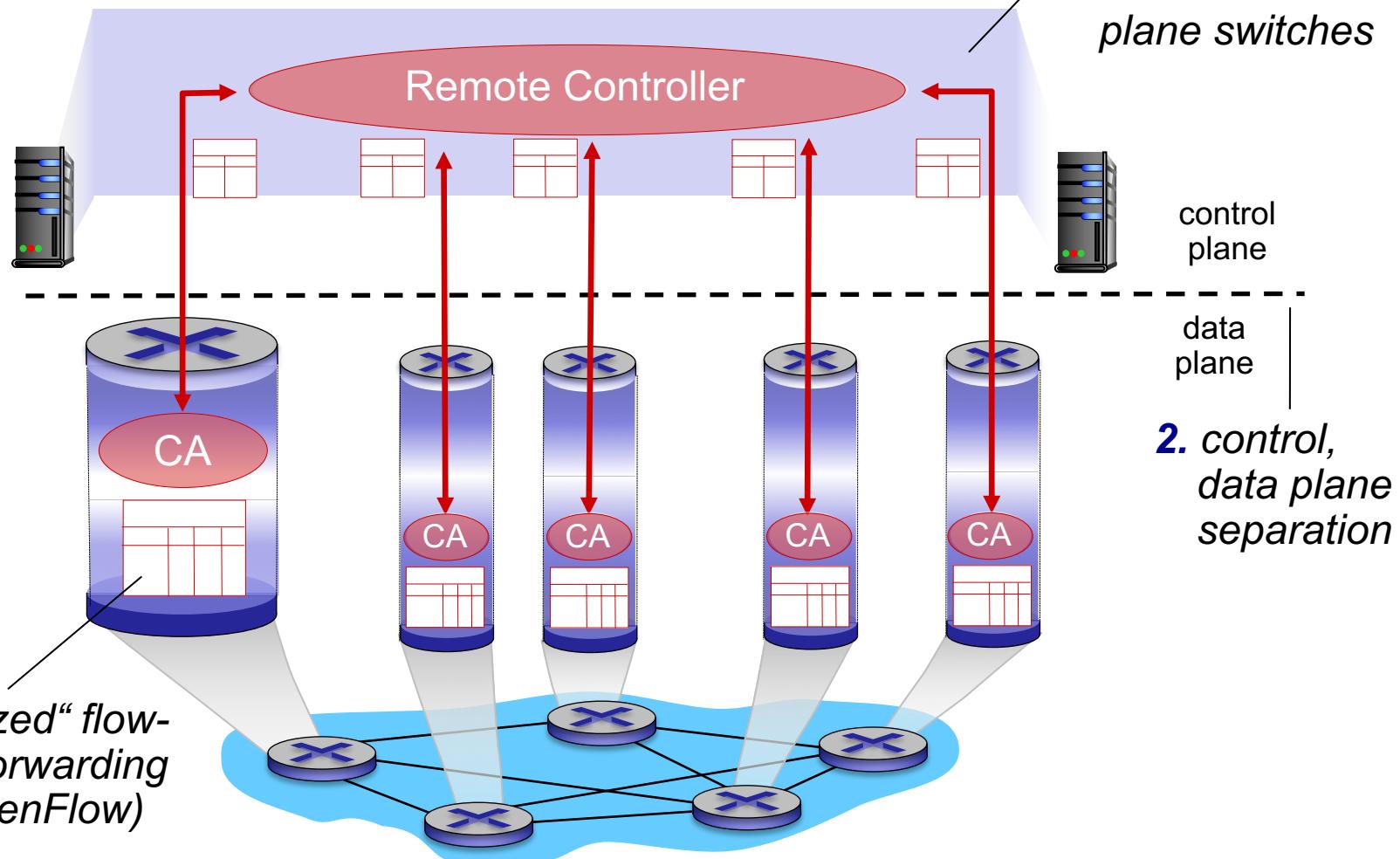
routing

access control

...

load balance

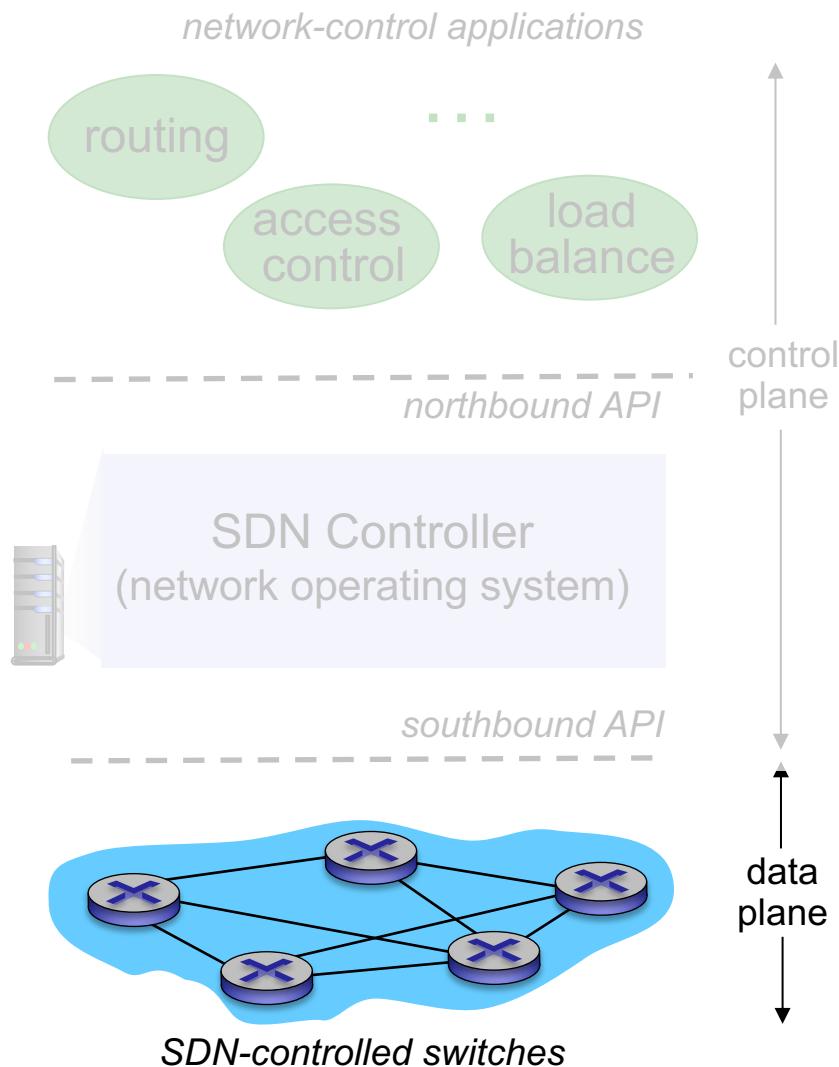
3. control plane  
functions  
external to data-  
plane switches



# SDN perspective: data plane switches

## *Data plane switches*

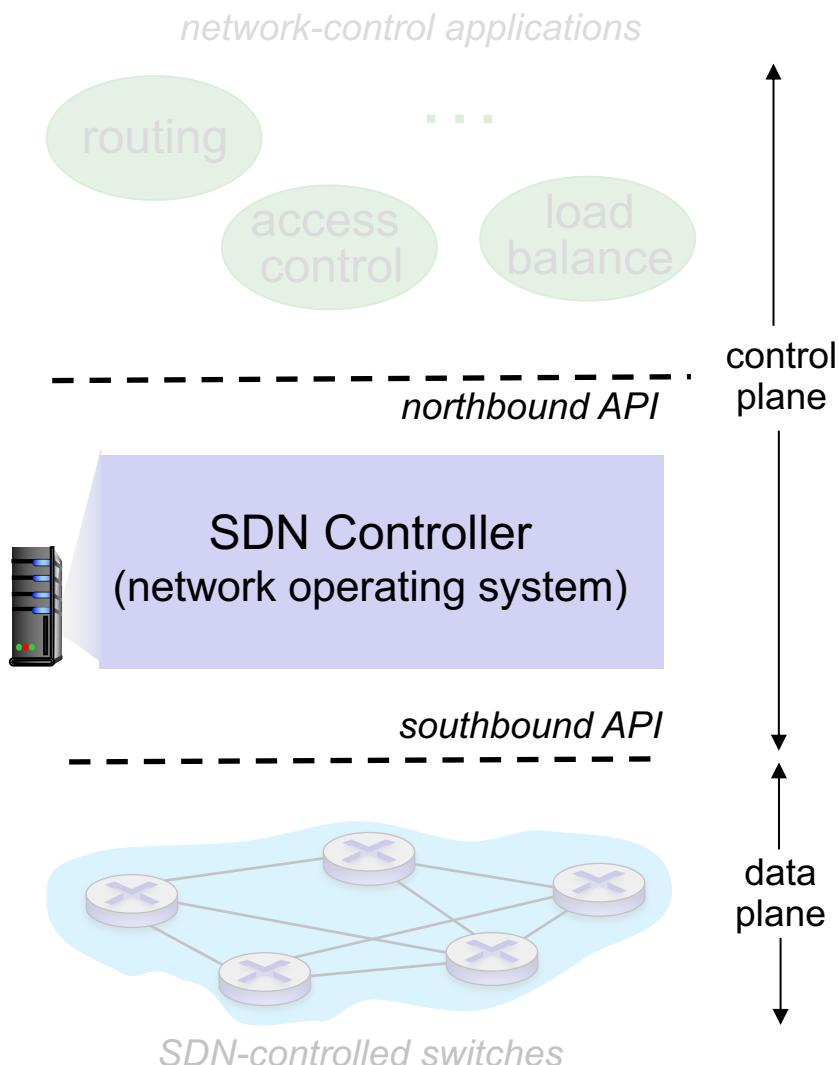
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller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



# SDN perspective: SDN controller

## *SDN controller (network OS):*

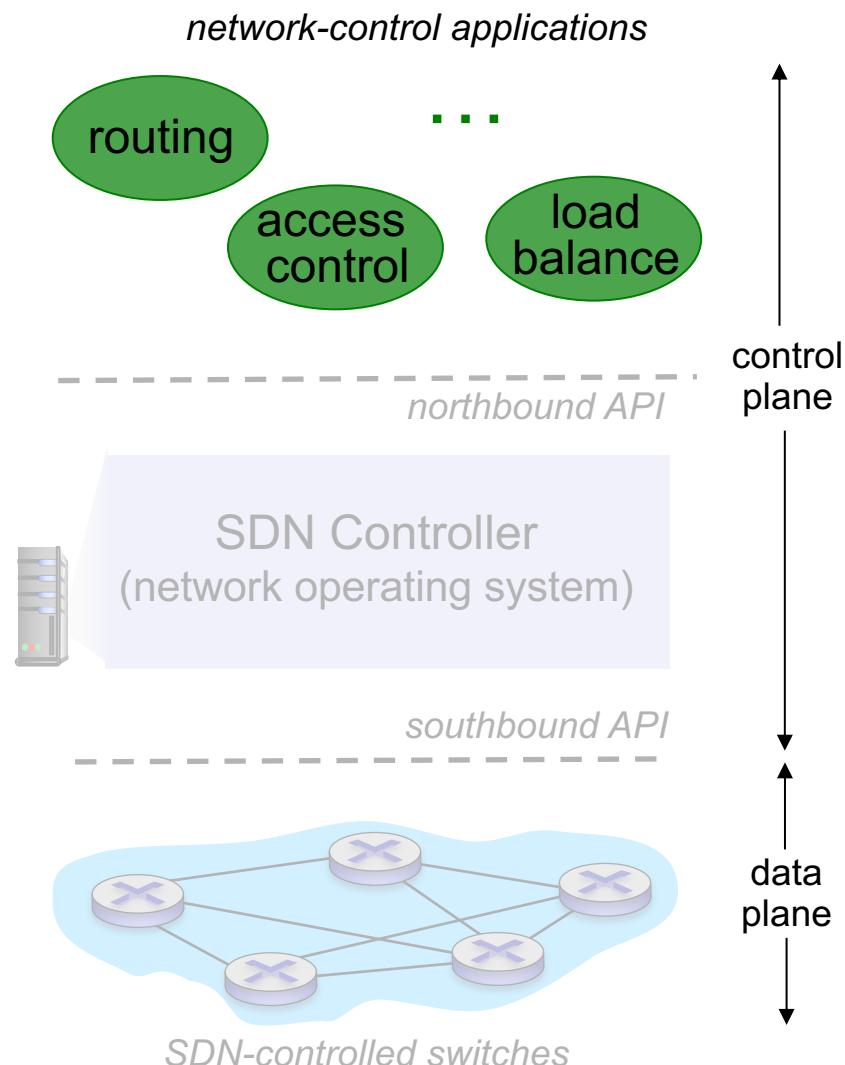
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



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

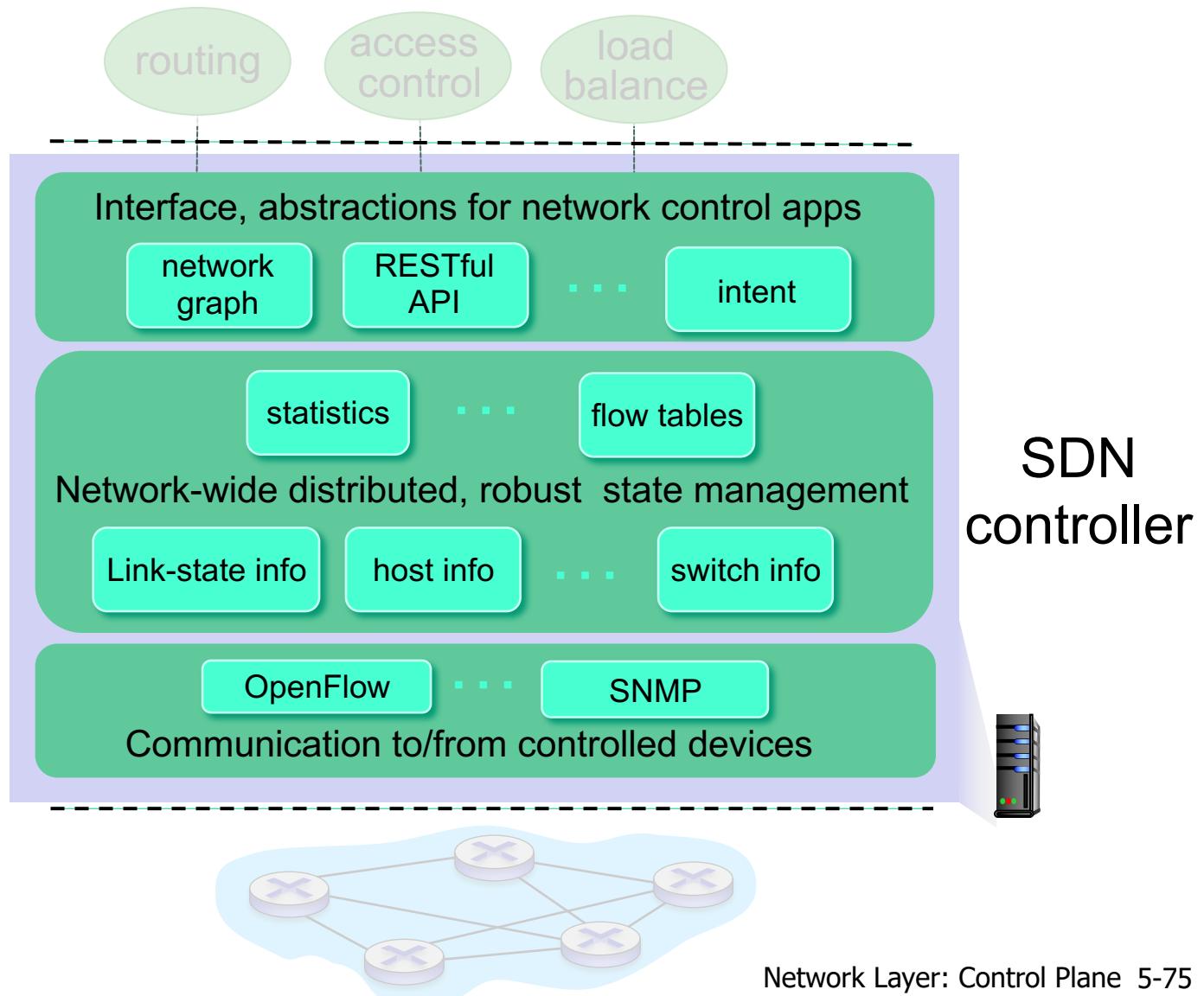


# Components of SDN controller

Interface layer to network control apps: abstractions API

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

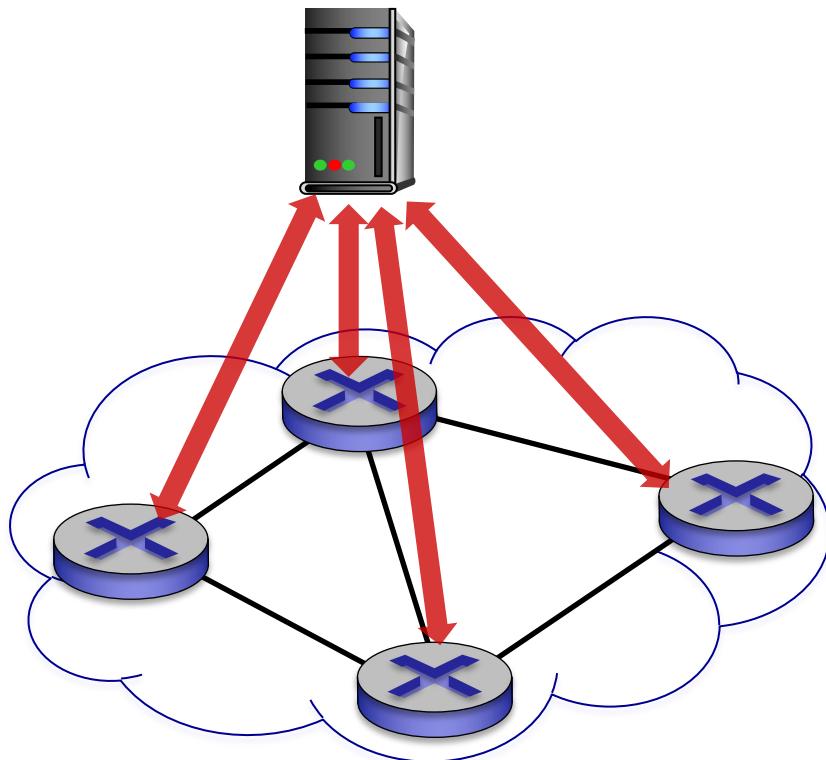
*communication layer:* communicate between SDN controller and controlled switches



SD  
controller

# OpenFlow protocol

OpenFlow Controller

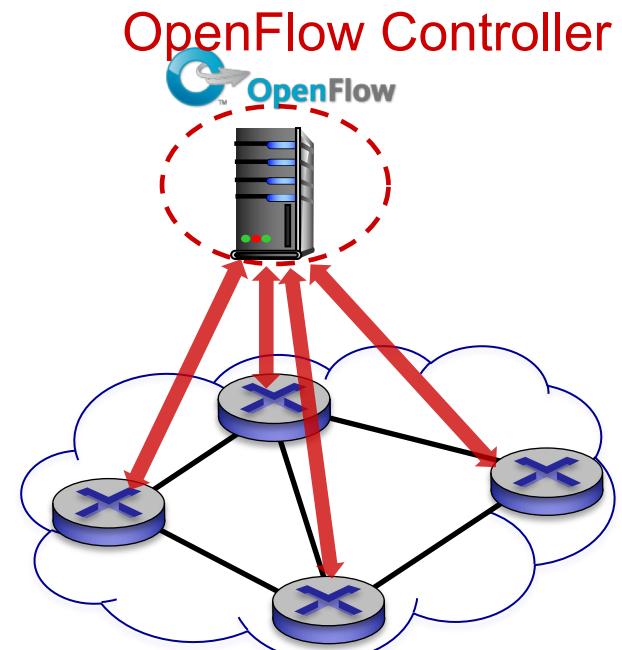


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

# OpenFlow: controller-to-switch messages

## Key controller-to-switch messages

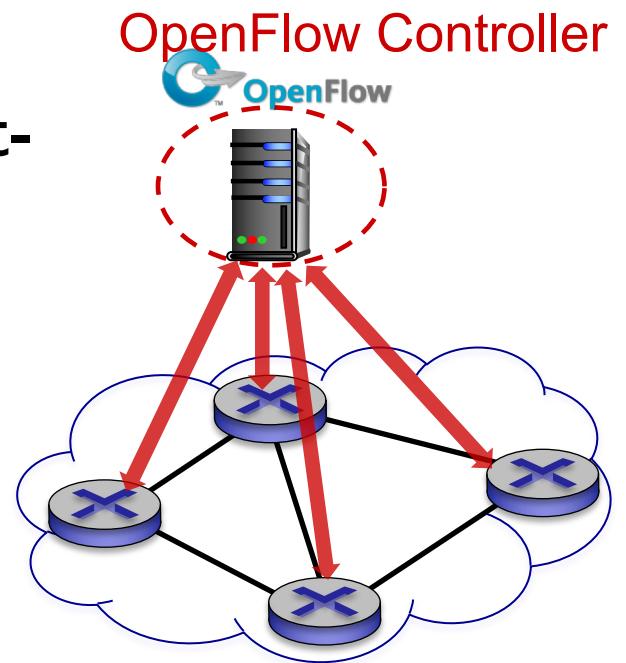
- **features**: controller queries switch features, switch replies
- **configure**: controller queries/sets switch configuration parameters
- **modify-state**: add, delete, modify flow entries in the OpenFlow tables
- **packet-out**: controller can send this packet out of specific switch port



# OpenFlow: switch-to-controller messages

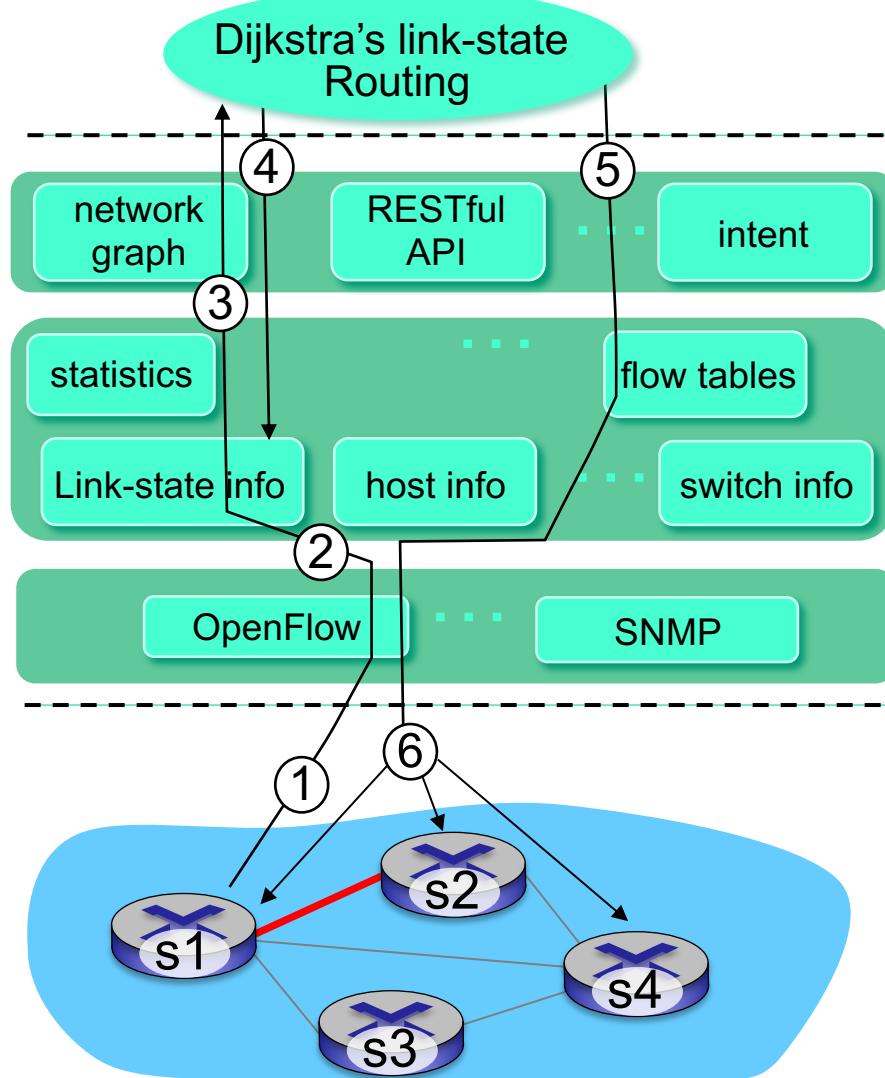
## *Key switch-to-controller messages*

- **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.



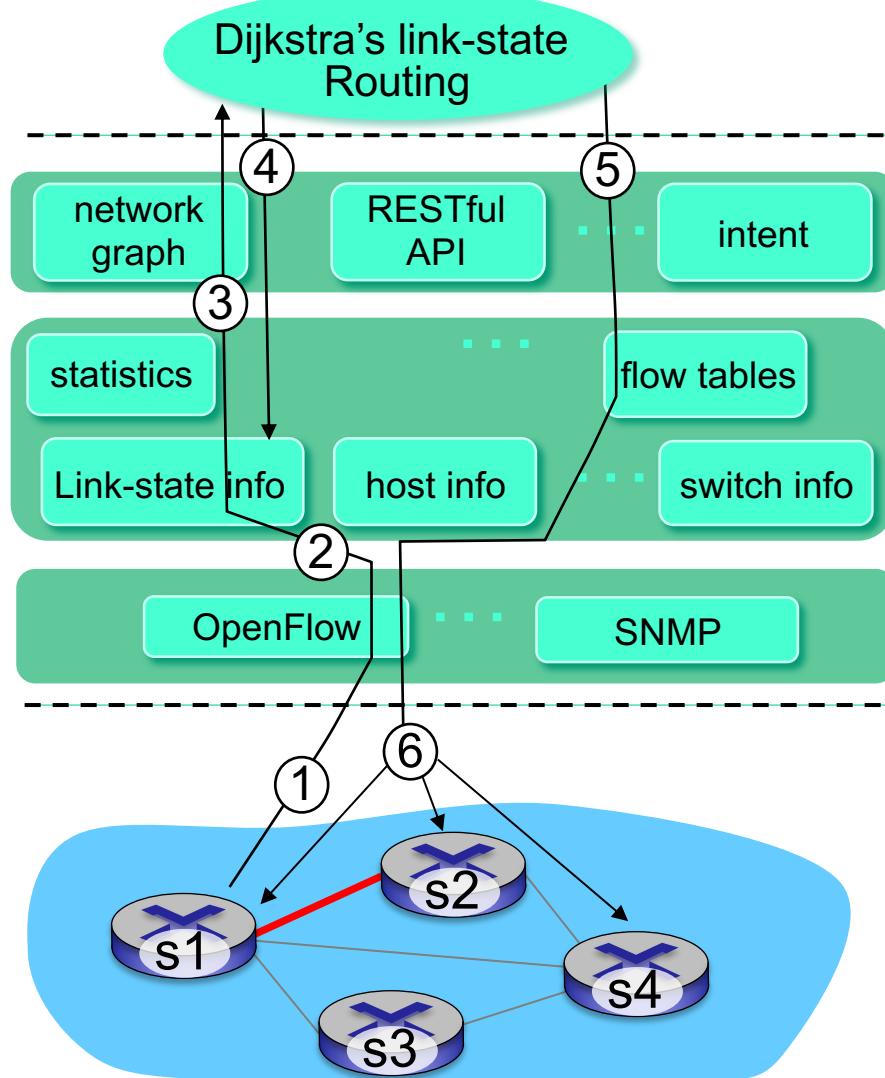
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN: control/data plane interaction example



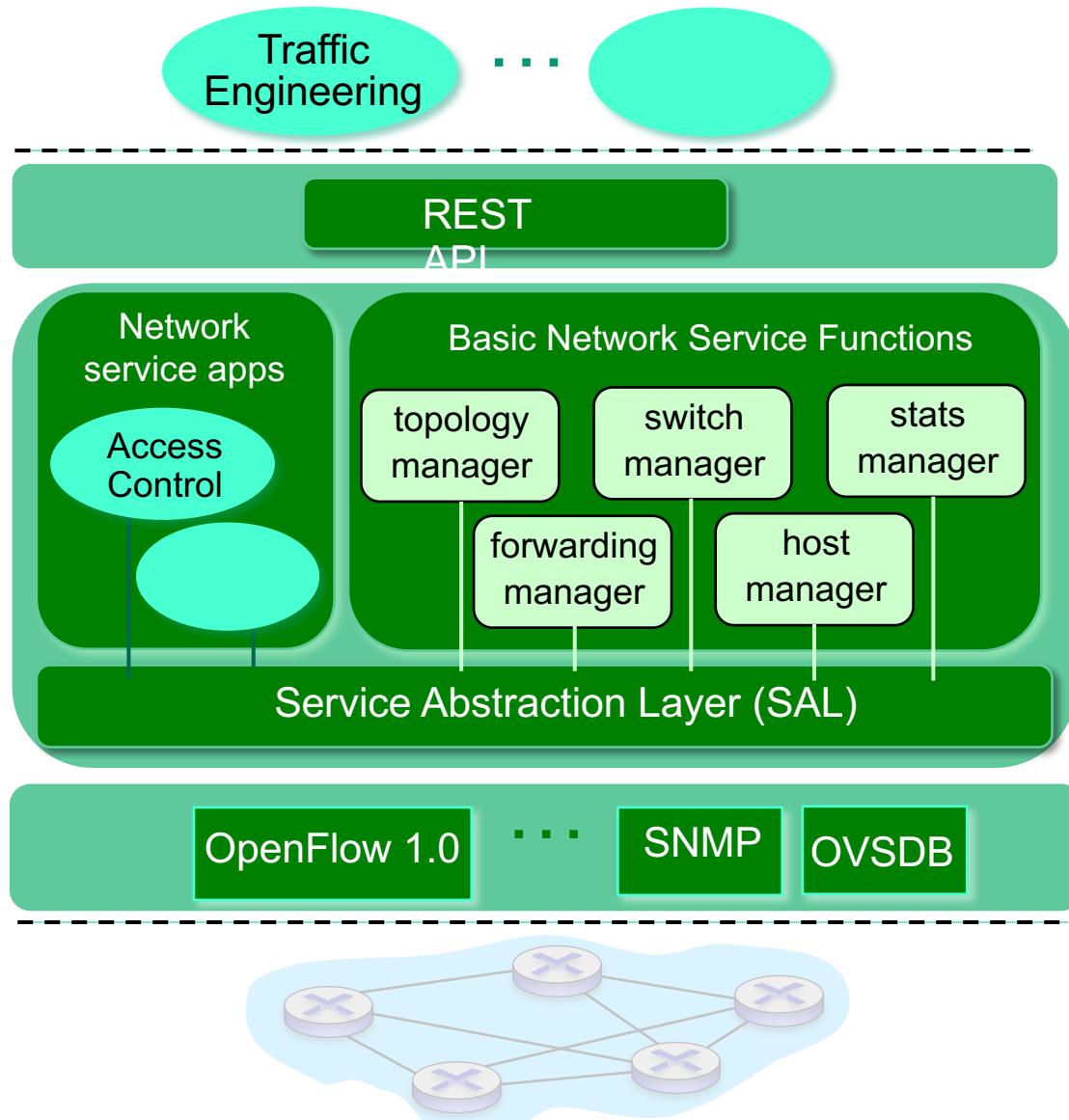
- ① SI, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's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

# SDN: control/data plane interaction example



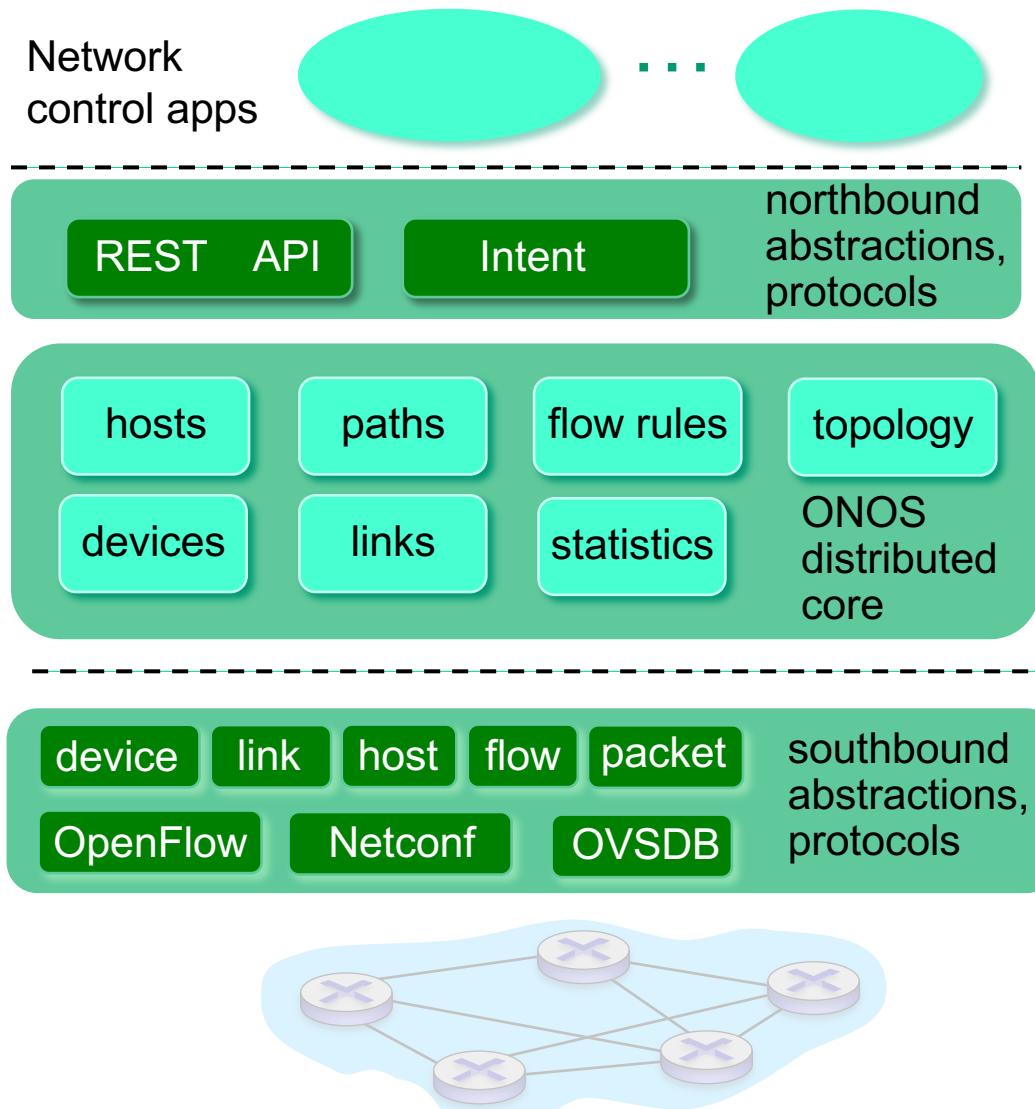
- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

# OpenDaylight (ODL) controller



- 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 scaling

# SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling

# 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

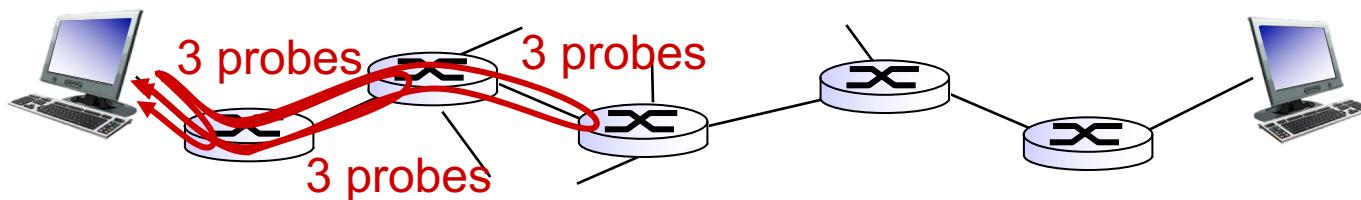
---

- 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

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 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: 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

*next stop: link layer!*