

Chapter 5

Network Layer: The Control Plane

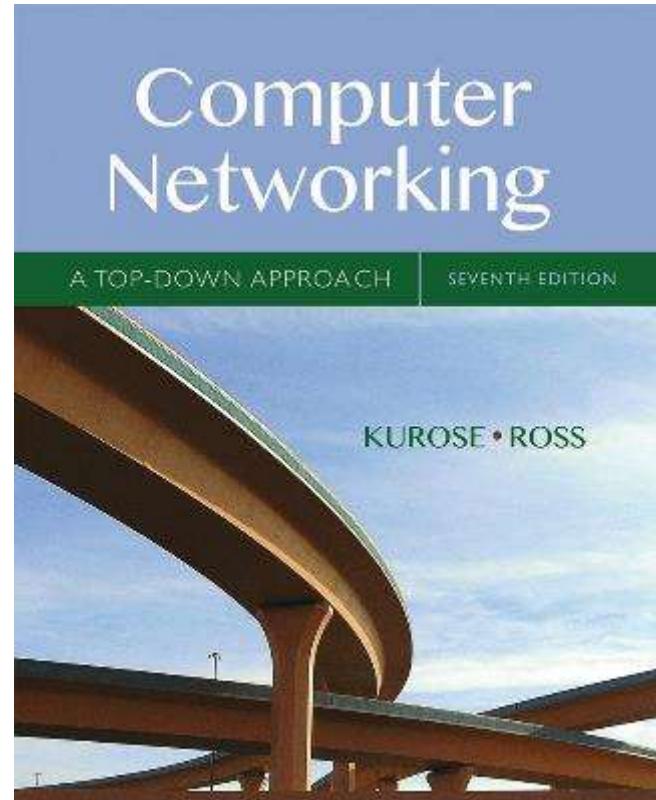
A note on the use of these Powerpoint slides:

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

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

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer
Networking: A Top
Down Approach*

7th edition

Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

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

5.7 Network management and SNMP

Network-layer functions

Recall: two network-layer functions:

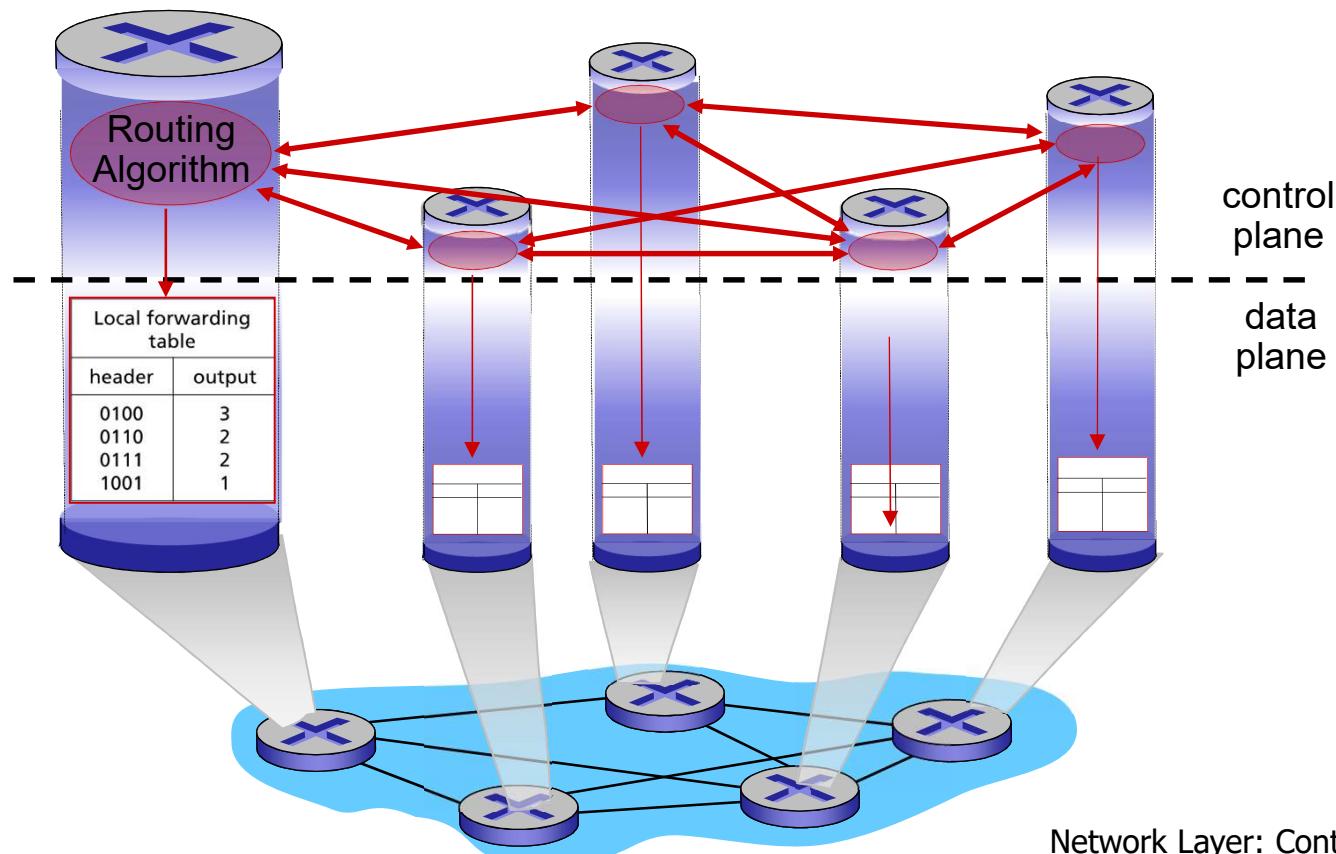
- *forwarding*: move packets from router's input to appropriate router output ***data plane***
- *routing*: determine route taken by packets from source to destination ***control plane***

Two approaches to structuring network control plane:

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

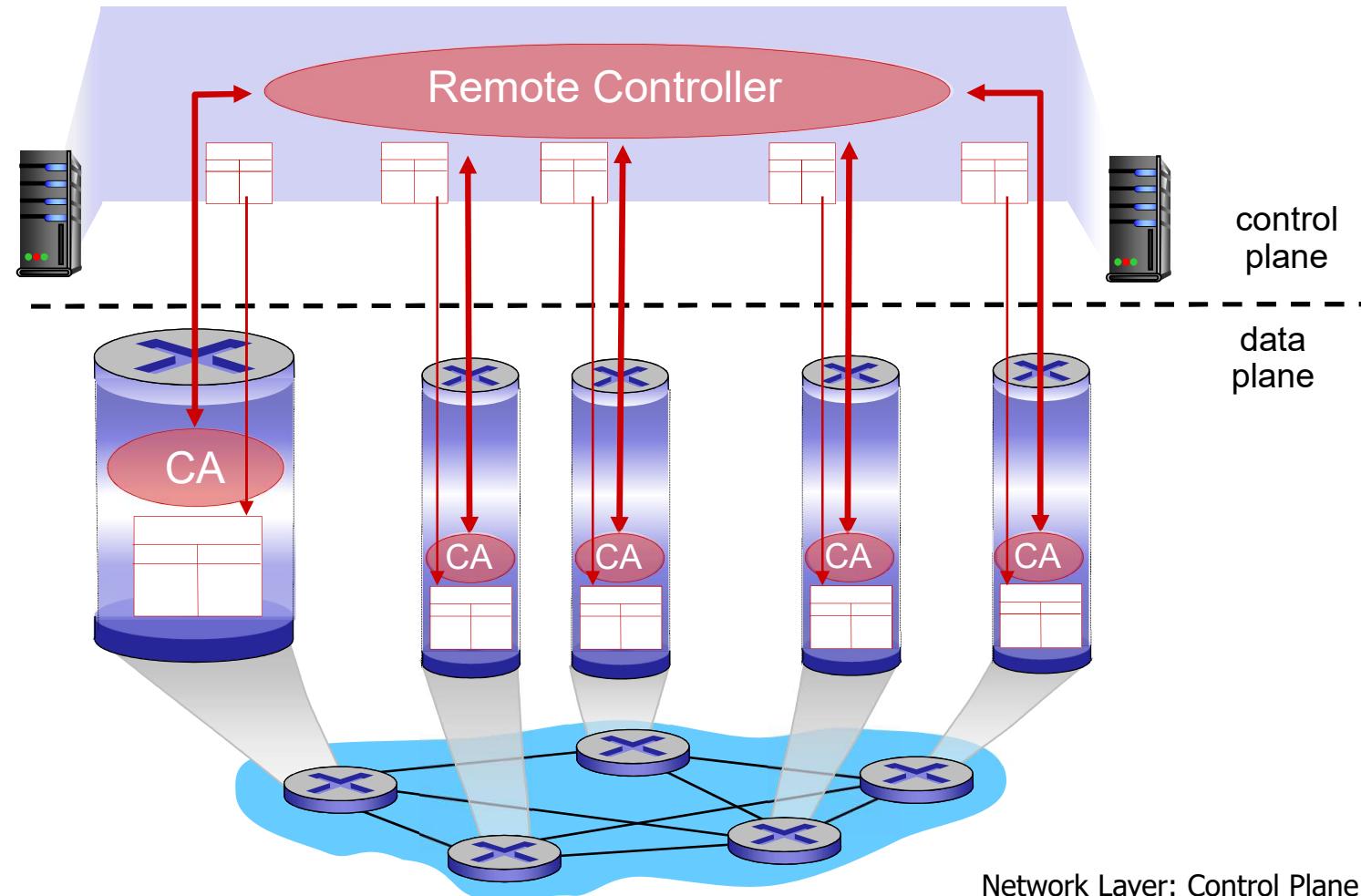
Per-router control plane

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

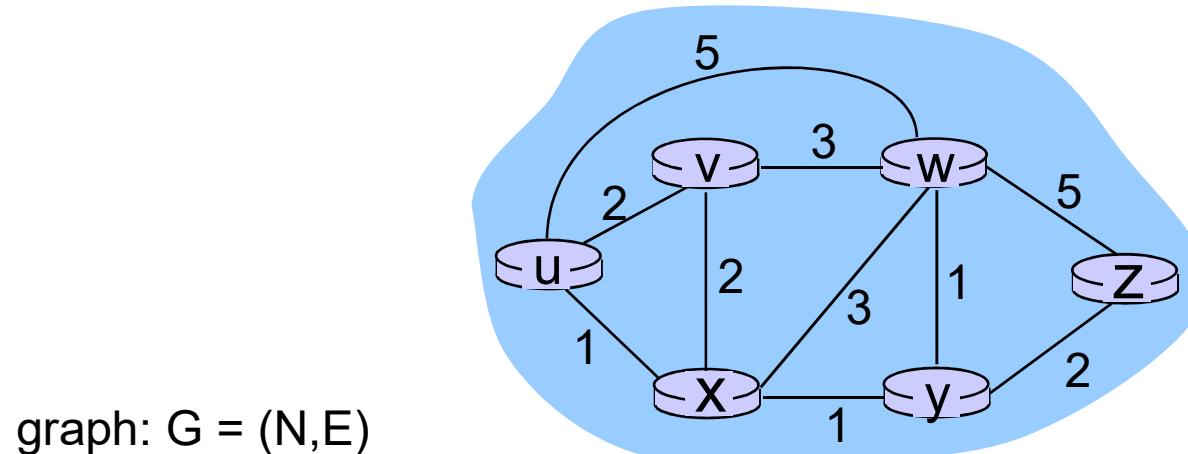
5.7 Network management and SNMP

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

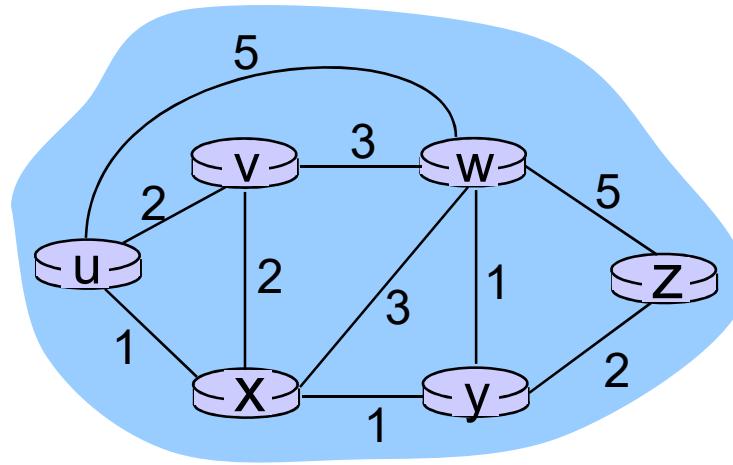


$N = \text{set of routers} = \{ u, v, w, x, y, z \}$

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

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

5.7 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

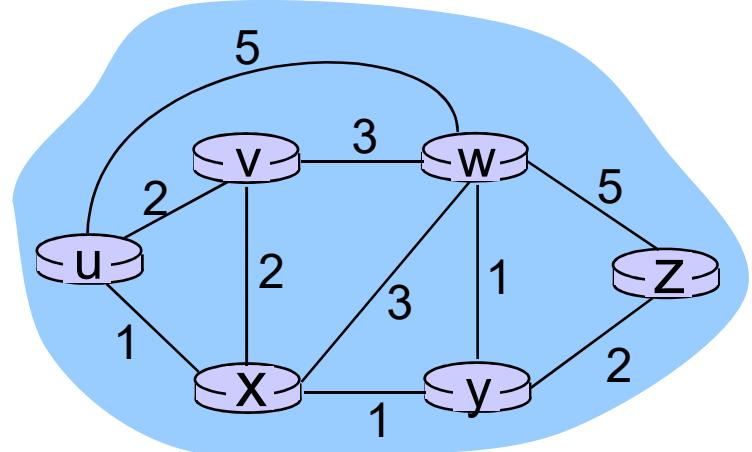
```
1 Initialization:
2    $N' = \{u\}$ 
3   for all nodes  $v$ 
4     if  $v$  adjacent to  $u$ 
5       then  $D(v) = c(u,v)$ 
6     else  $D(v) = \infty$ 
7
8 Loop
9   find  $w$  not in  $N'$  such that  $D(w)$  is a minimum
10  add  $w$  to  $N'$ 
11  update  $D(v)$  for all  $v$  adjacent to  $w$  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 | 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 | | | | | |

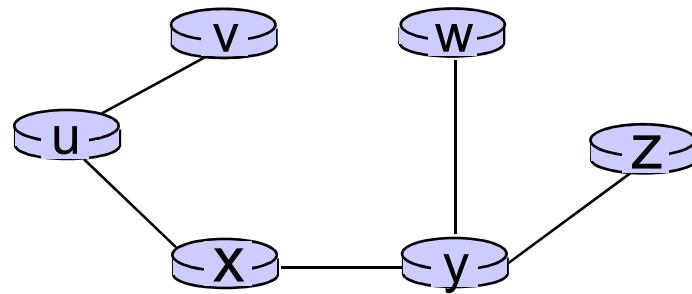
1 **Initialization:**
 2 $N' = \{u\}$
 3 for all nodes v
 4 if v adjacent to u
 5 then $D(v) = c(u,v)$
 6 else $D(v) = \infty$
 7

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 (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: example

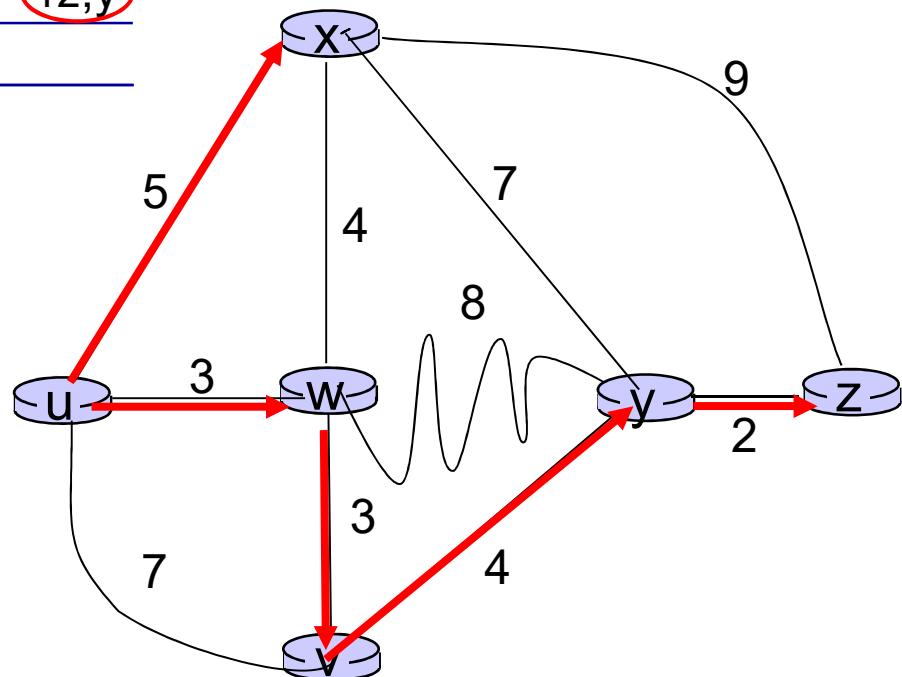
| Step | N' | D(v) | D(w) | D(x) | D(y) | D(z) |
|------|--------|------|------|------|----------|----------|
| | | p(v) | p(w) | p(x) | p(y) | p(z) |
| 0 | u | 7,u | 3,u | 5,u | ∞ | ∞ |
| 1 | uw | 6,w | | 5,u | 11,w | ∞ |
| 2 | uwx | 6,w | | | 11,w | 14,x |
| 3 | uwxv | | | | 10,v | 14,x |
| 4 | uwxvy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

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'

notes:

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



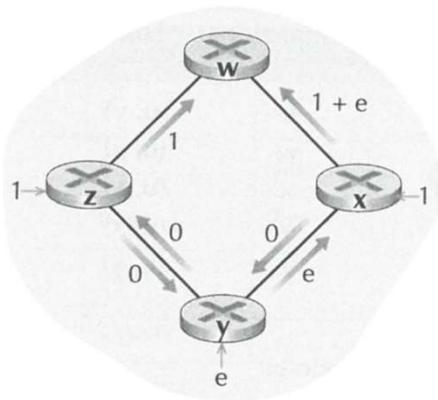
Dijkstra's algorithm, discussion

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: $O(n \log n)$

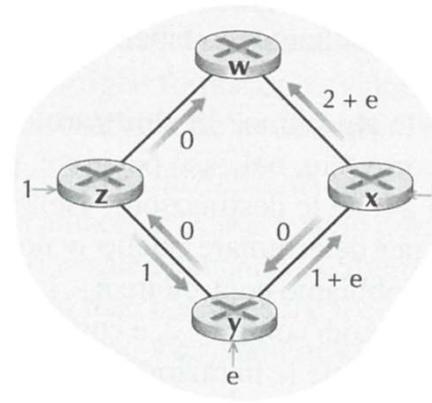
oscillations possible:

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

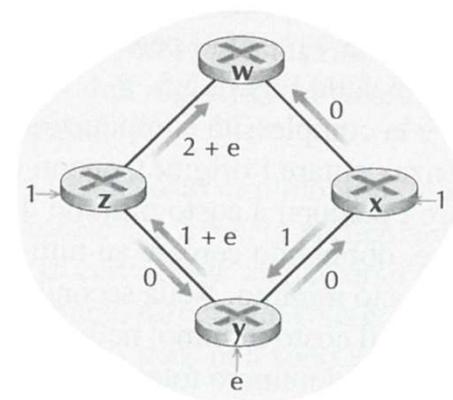


initially

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

5.7 Network management
and SNMP

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

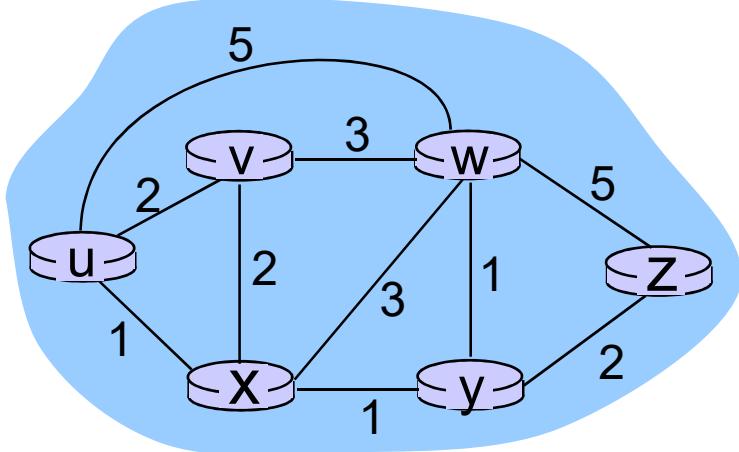
$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

then

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

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

$$\begin{aligned}
 D_x(y) &= \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \\
 &= \min\{2+0, 7+1\} = 2
 \end{aligned}$$

$$\begin{aligned}
 D_x(z) &= \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \\
 &= \min\{2+1, 7+0\} = 3
 \end{aligned}$$

node x table

| from | x | y | z |
|------|----------|----------|----------|
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

cost to

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

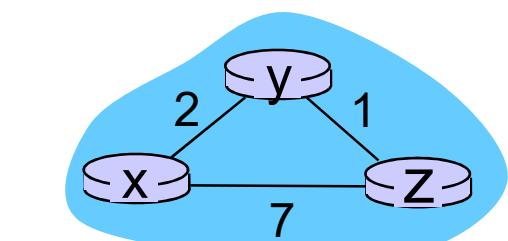
node y table

| from | x | y | z |
|------|----------|----------|----------|
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

node z table

| from | x | y | z |
|------|----------|----------|----------|
| 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 |
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

**node y
table**

| | cost to | | |
|---|----------|----------|----------|
| | x | y | z |
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

**node z
table**

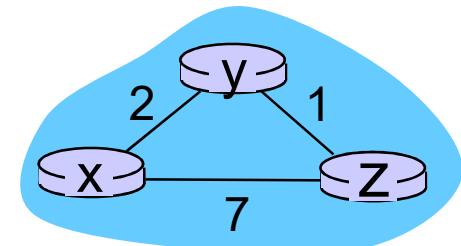
| | cost to | | |
|---|----------|----------|----------|
| | x | y | z |
| x | ∞ | ∞ | ∞ |
| y | ∞ | ∞ | ∞ |
| z | 7 | 1 | 0 |

| | cost to | | |
|---|---------|---|---|
| | x | y | z |
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

| | cost to | | |
|---|---------|---|---|
| | x | y | z |
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

| | cost to | | |
|---|---------|---|---|
| | x | y | z |
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

| | cost to | | |
|---|---------|---|---|
| | x | y | z |
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |



Distance vector: link cost changes

link cost changes:

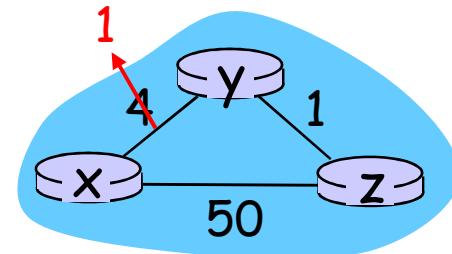
- ❖ node detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors

**“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 .

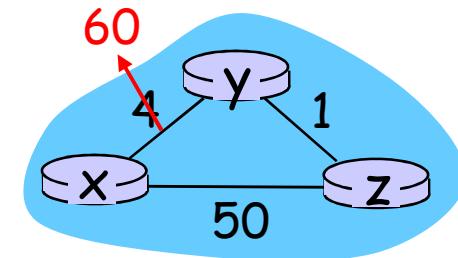


* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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

5.7 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”)

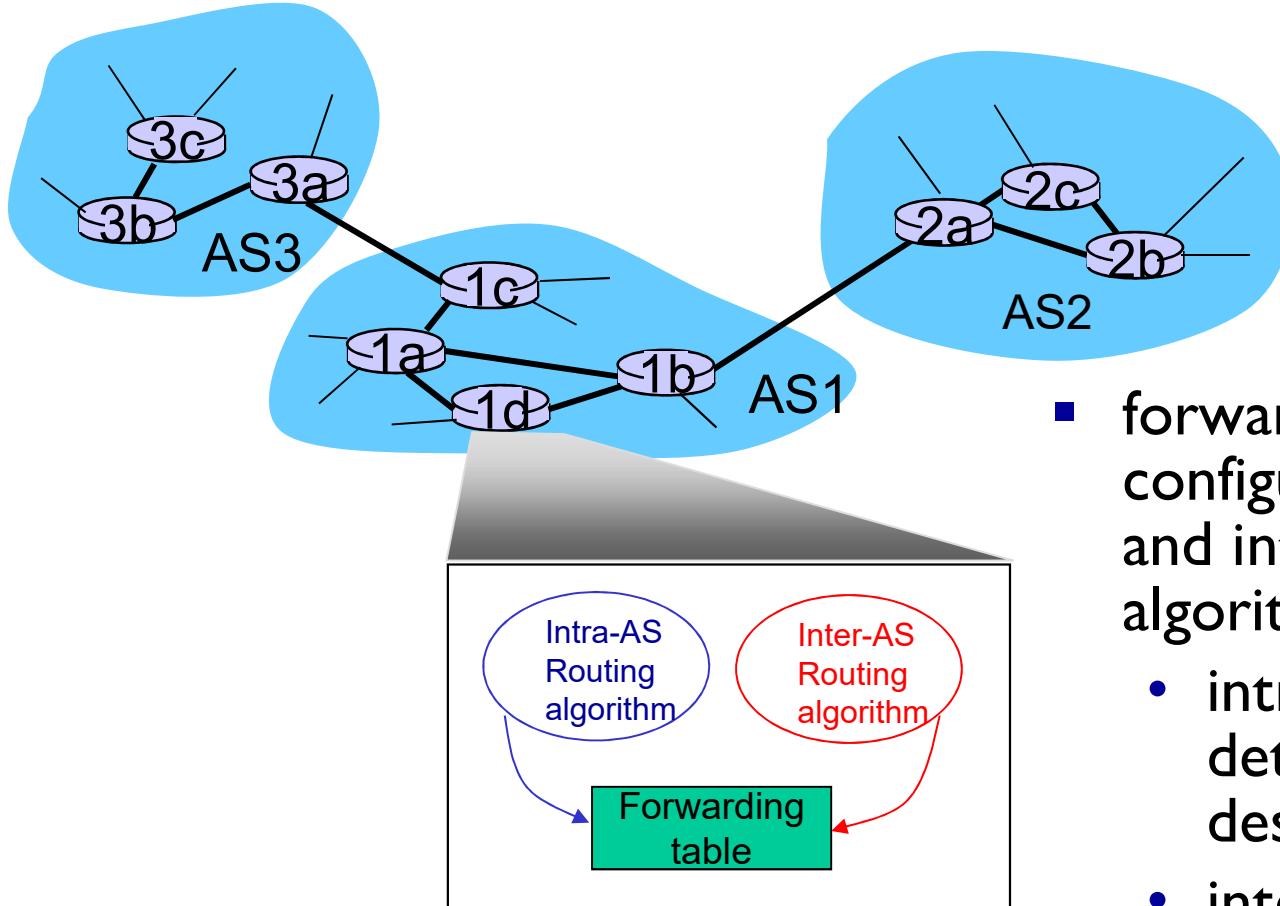
intra-AS routing

- 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

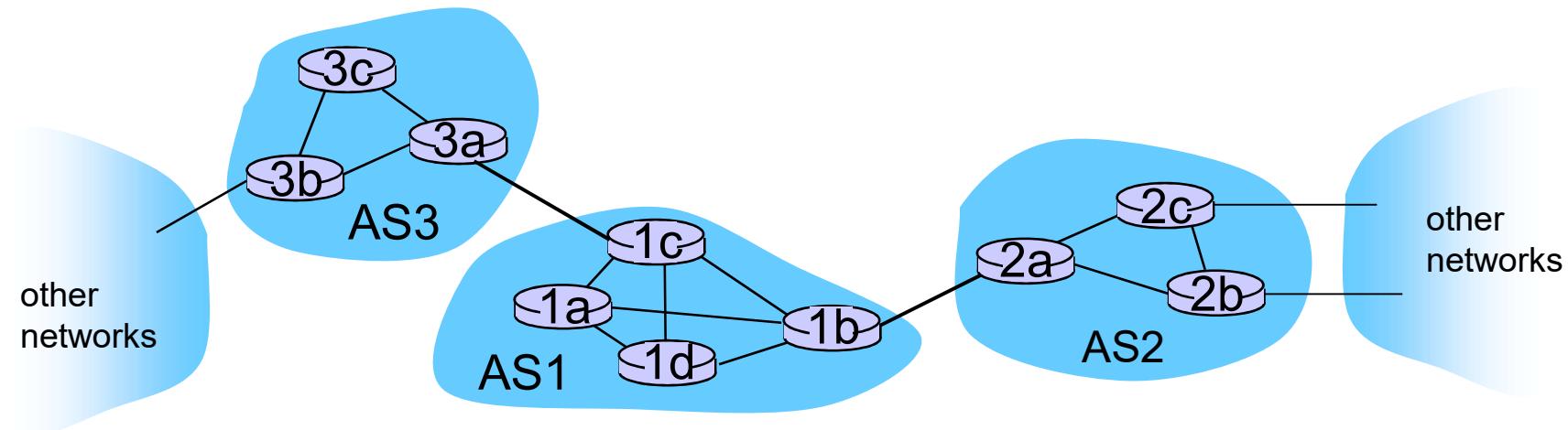
Inter-AS tasks

- 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 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

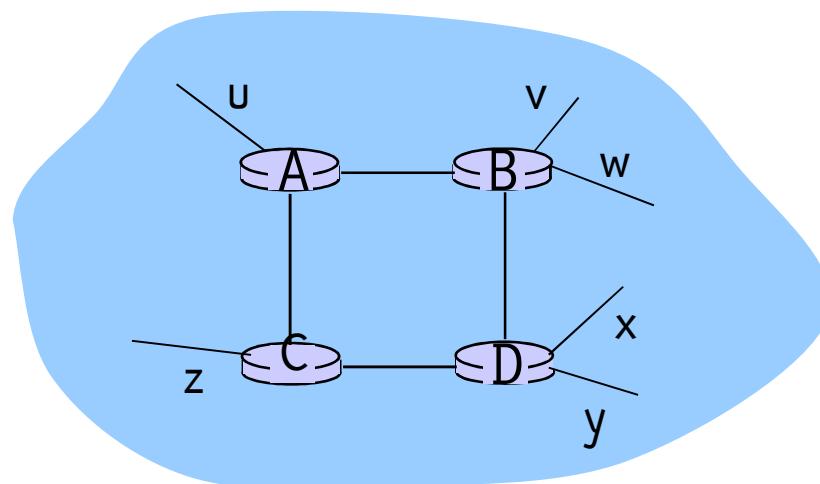
- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
 - RIP: Routing Information Protocol
 - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF)
 - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

RIP (Routing Information Protocol)

Incluso nella distribuzione BSD-UNIX dal 1982

È un distance vector algorithm basato su broadcast

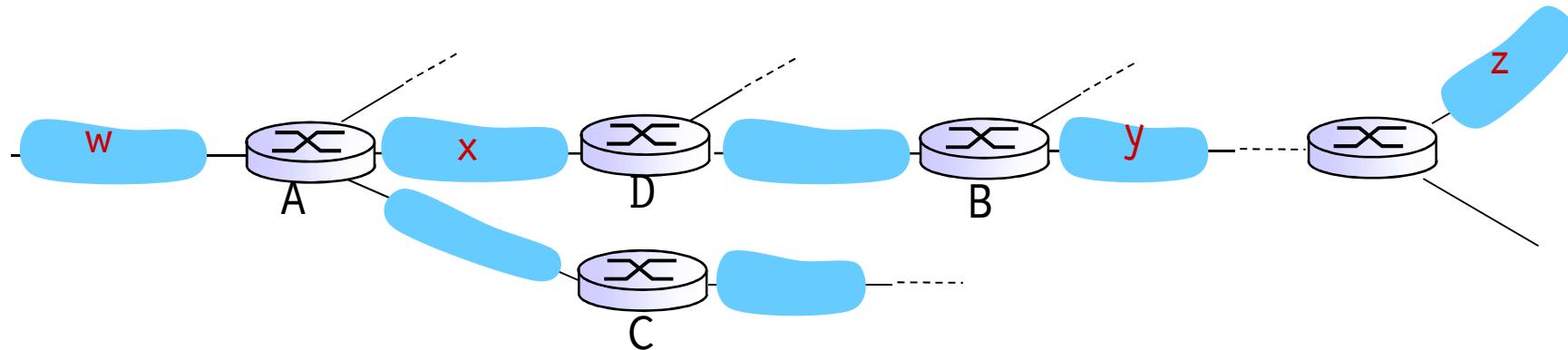
- Metrica per la distanza: # hops (max = 15 hop), ogni link a costo 1
- I DV sono scambiati fra vicini ogni 30 sec usando un **RIP response message** (detto anche **advertisement**) in broadcast
- Ogni advertisement lista fino ad un massimo di 25 **subnet** destinazione (*in termini di indirizzamento di IP*)



Da router A a **subnet** destinazione:

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

RIP: un esempio

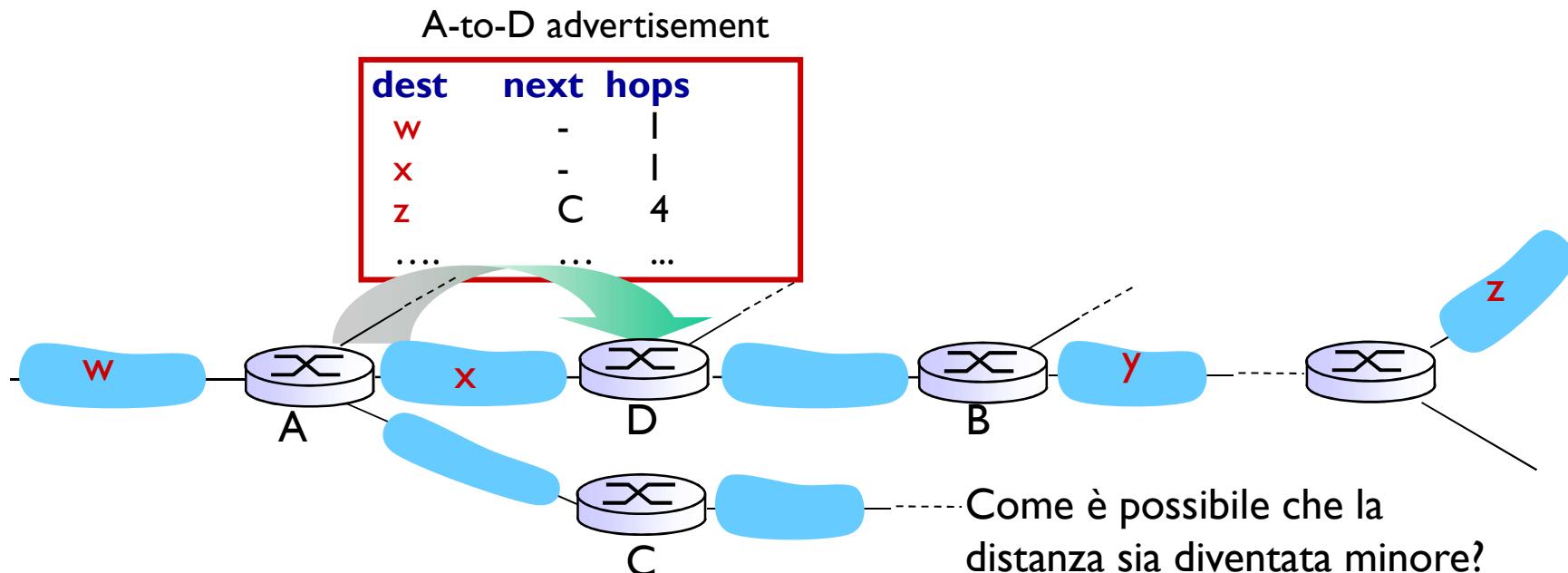


La tabella di routing di RIP nel router D

| subnet destinazione | prossimo router | # hops per dest. |
|---------------------|-----------------|------------------|
|---------------------|-----------------|------------------|

| | | |
|------|------|------|
| w | A | 2 |
| y | B | 2 |
| z | B | 7 |
| x | -- | 1 |
| | | |

RIP: un esempio



routing table in router D

| destination subnet | next router | # hops to dest |
|--------------------|-------------|----------------|
| w | A | 2 |
| y | B | 2 |
| z | B | 7 |
| x | A | 5 |
| | | |

RIP: guasto dei link, recovery

Se un RIP non riceve advertisement da un RIP vicino per 180 sec → il RIP vicino (o il link) è considerato "morto"

- Le route via questo vicino sono invalidate
- Nuovi advertisement sono inviati ai vicini
- I vicini a loro volta mandano nuovi advertisement se le loro tabelle cambiano
- L'informazione sul guasto di un link si propaga (come? ...lentamente!) all'intera rete
- *Inversione avvelenata* è usata per prevenire loop (distanza infinita = 16 hop)
- Altre euristiche sono usate per limitare i danni della lenta convergenza

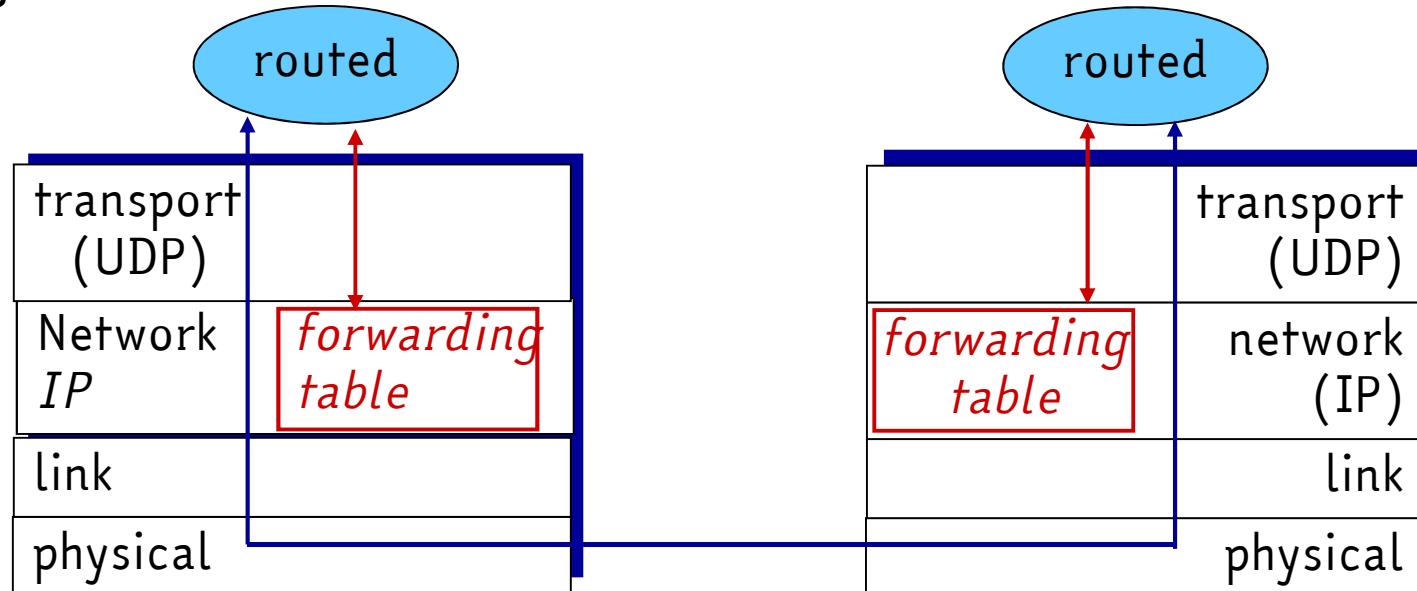
Elaborazione della tabella di RIP

Le tabelle di routing di RIP sono gestite da un processo *a livello applicativo* chiamato **route-d** (è un daemon)

Gli advertisement sono inviati in pacchetti UDP, periodicamente, per rilevare i guasti dei link/router

Route-d è a livello applicativo, sia per il Sistema Operativo che per lo stack di protocolli!

Perché bidirezionale la freccia rossa? Settare/leggere le righe e leggere lo stato dei link locali



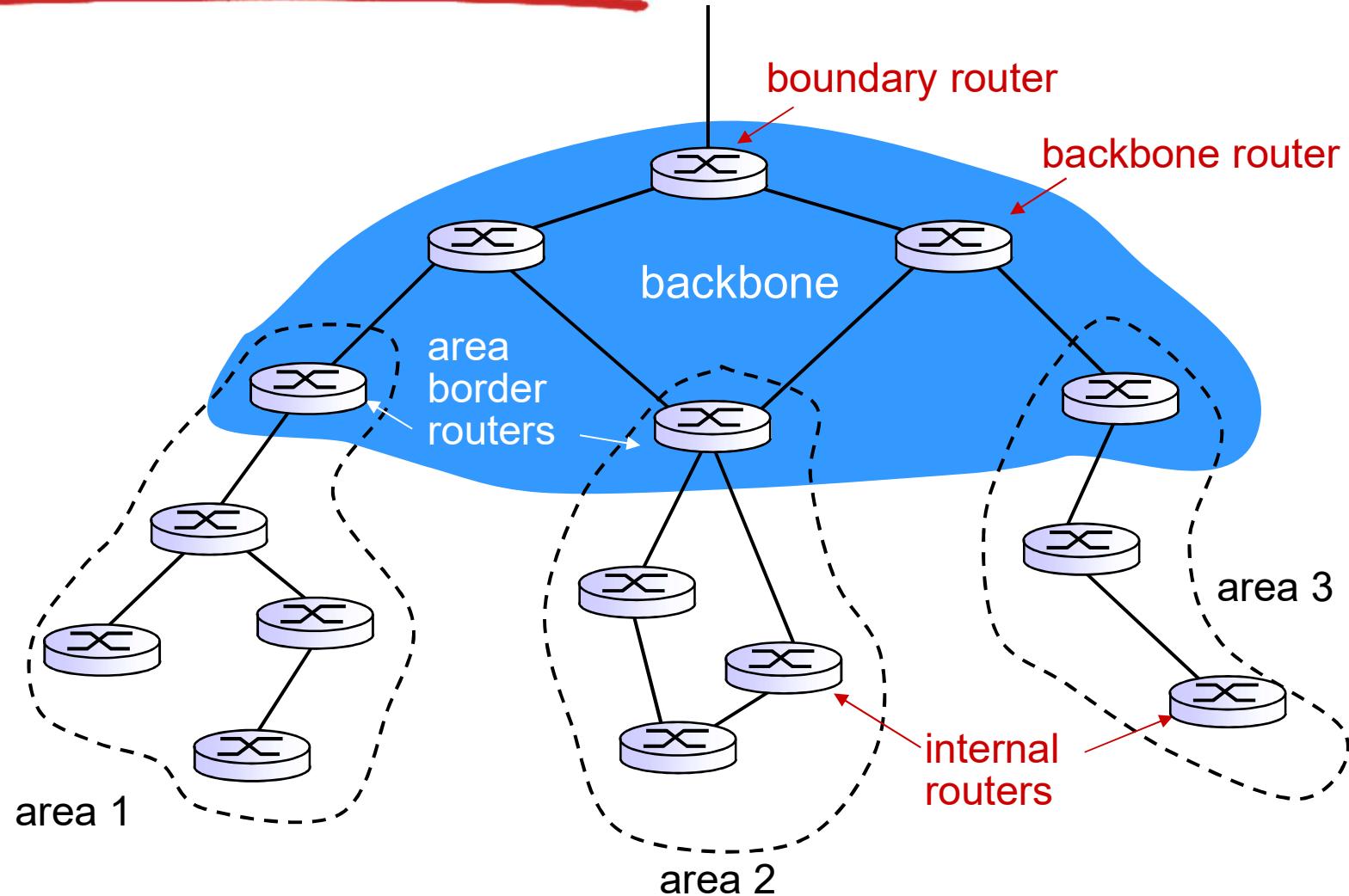
OSPF (Open Shortest Path First)

- “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.

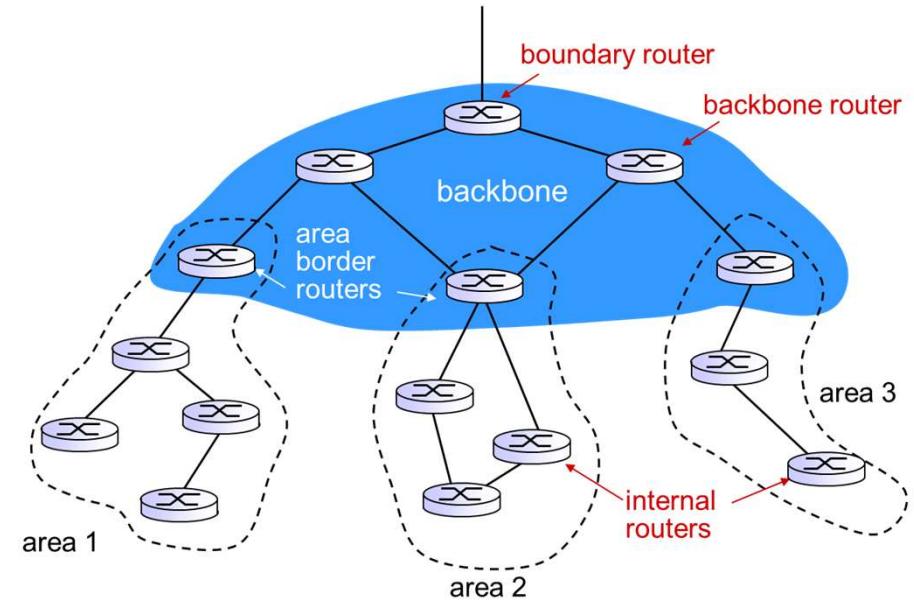
Hierarchical OSPF



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**

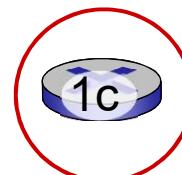
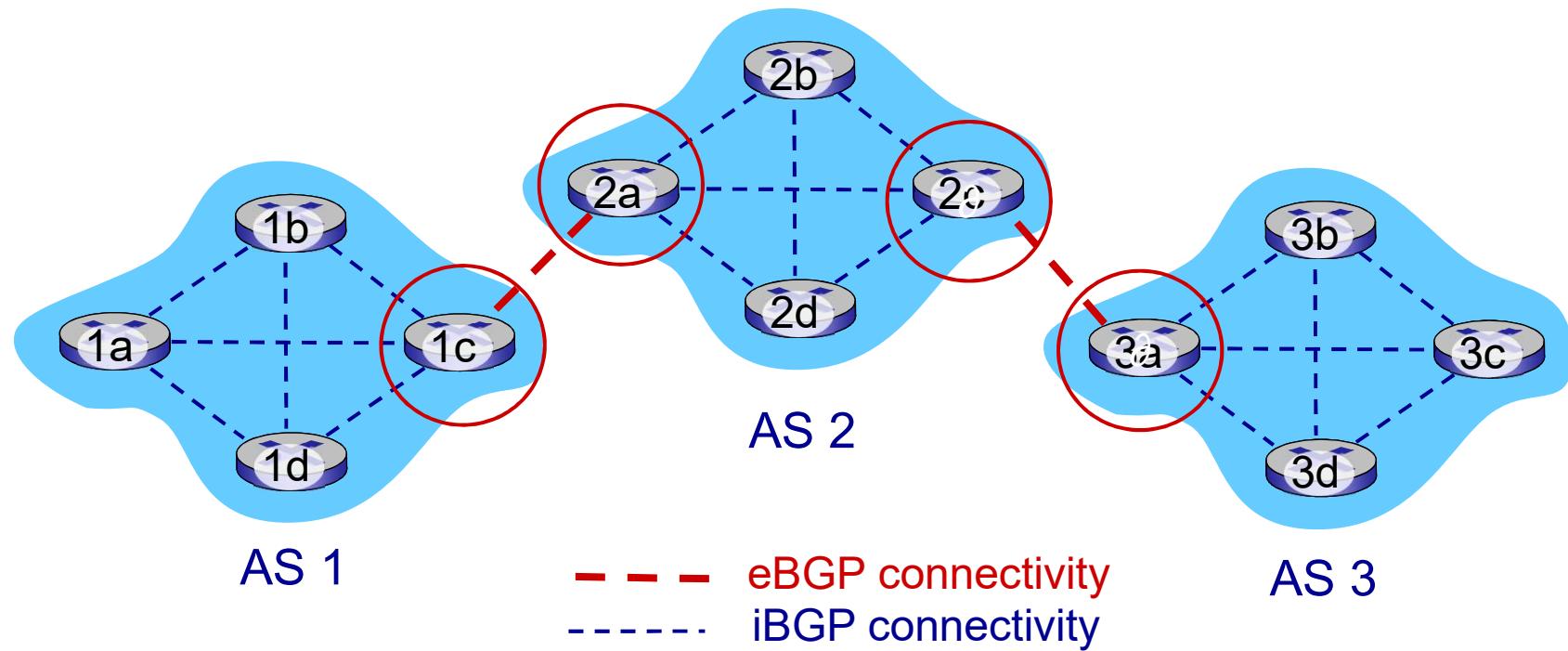
5.5 The SDN control plane

5.7 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*”

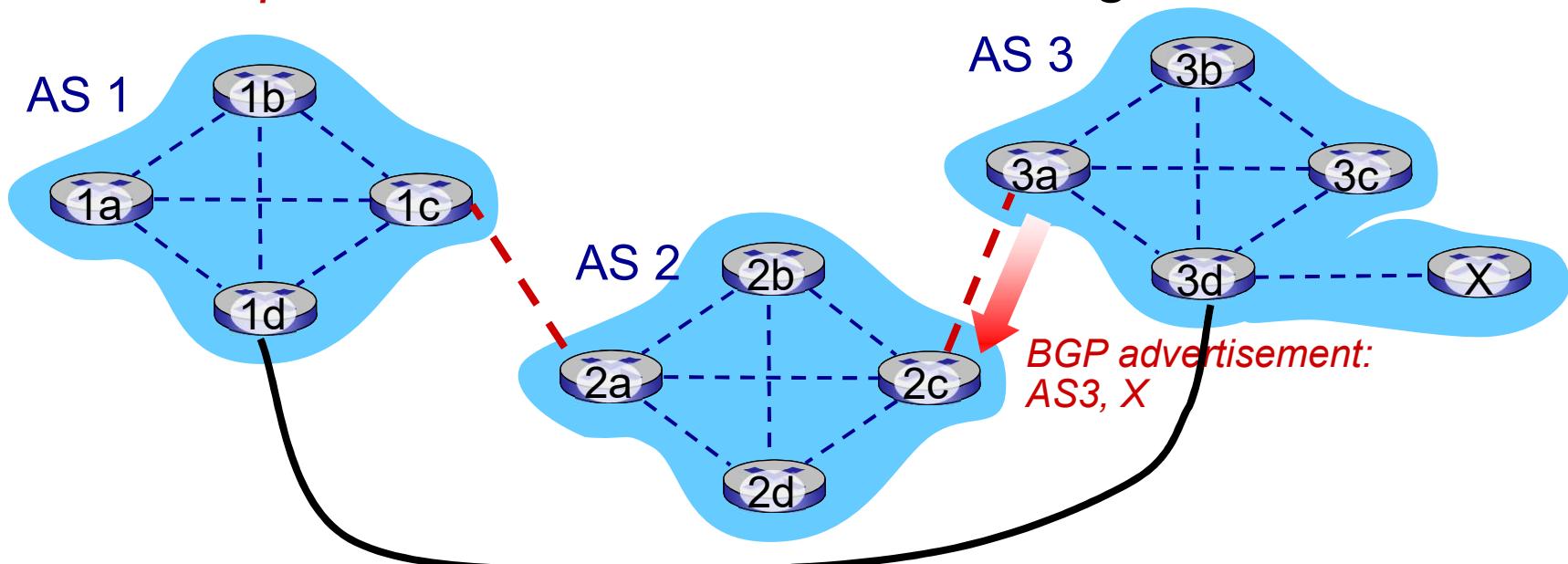
eBGP, iBGP connections



gateway routers run both eBGP and iBGP protocols

BGP basics

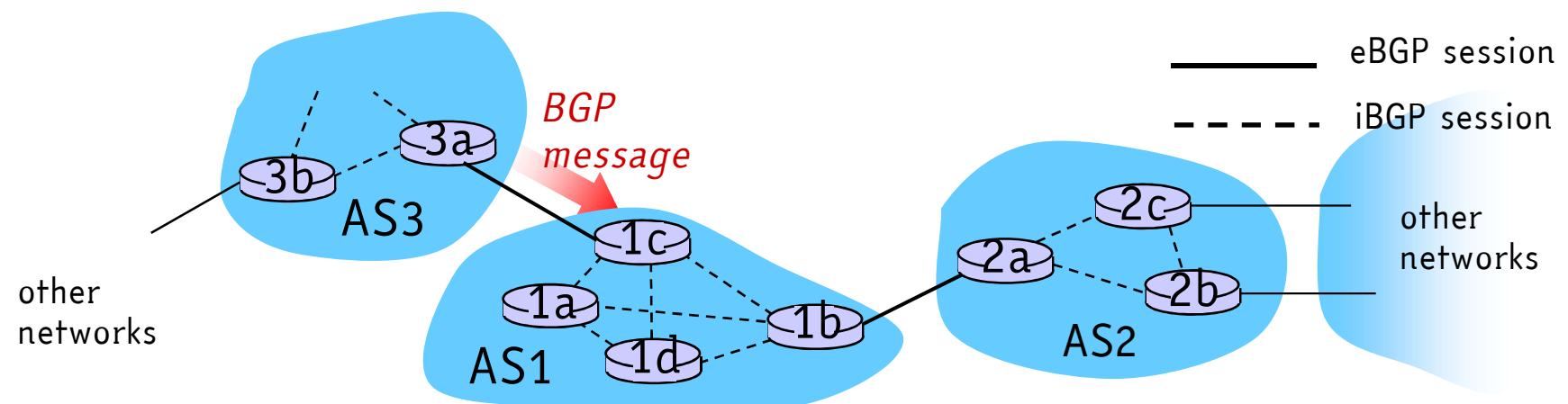
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
 - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway router 3a advertises path **AS3,X** to AS2 gateway router 2c:
 - AS3 *promises*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



BGP basics (II)

Ad es. AS2 ha: 138.16.64/24, 138.16.65/24, 138.16.66/24, e 138.16.67/24 → AS2 pubblica 138.16.64/22 a AS1

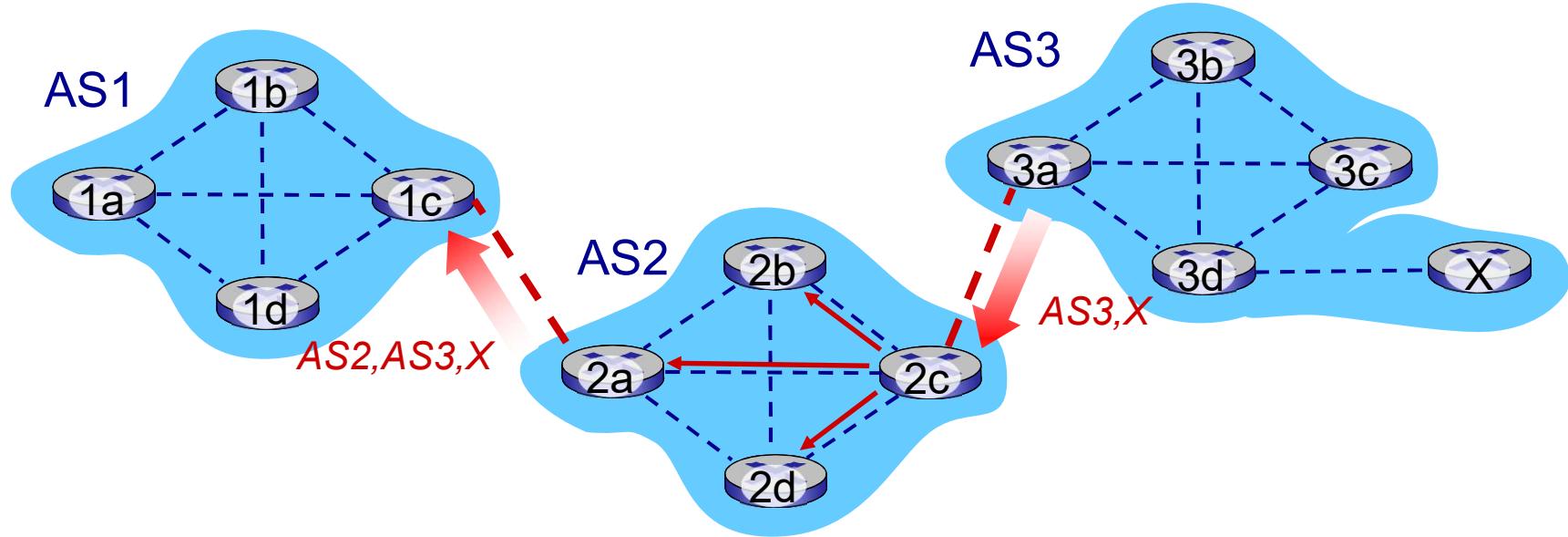
Altro es. AS2 ha 138.16.64/24, 138.16.65/24, 138.16.66/24, **ma questa volta 138.16.67/24 è in AS3**, che lo pubblica ad AS1 → AS2 può ancora pubblicare 138.16.64/22, contando sul fatto che i router (ad es. quelli di AS1) usino il longest prefix matching



Path attributes and BGP routes

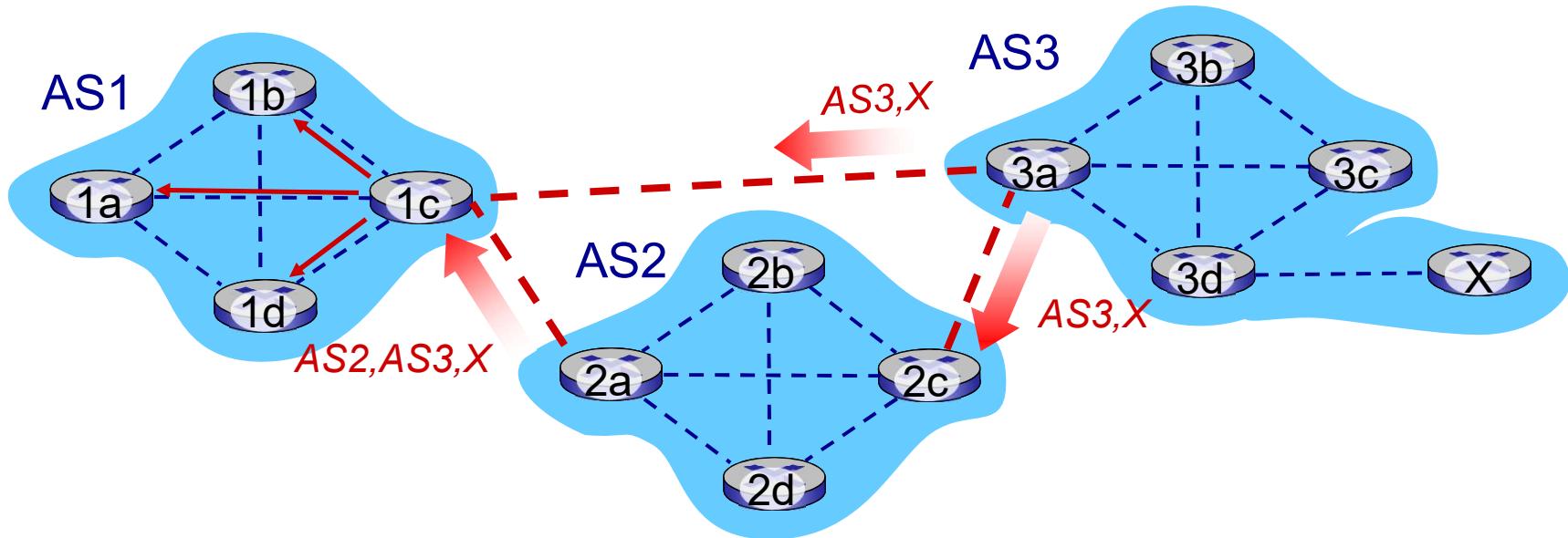
- advertised prefix includes BGP attributes
 - prefix + attributes = “route”
- two important attributes:
 - **AS-PATH**: list of ASes through which prefix advertisement has passed
 - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
- ***Policy-based routing***:
 - gateway receiving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

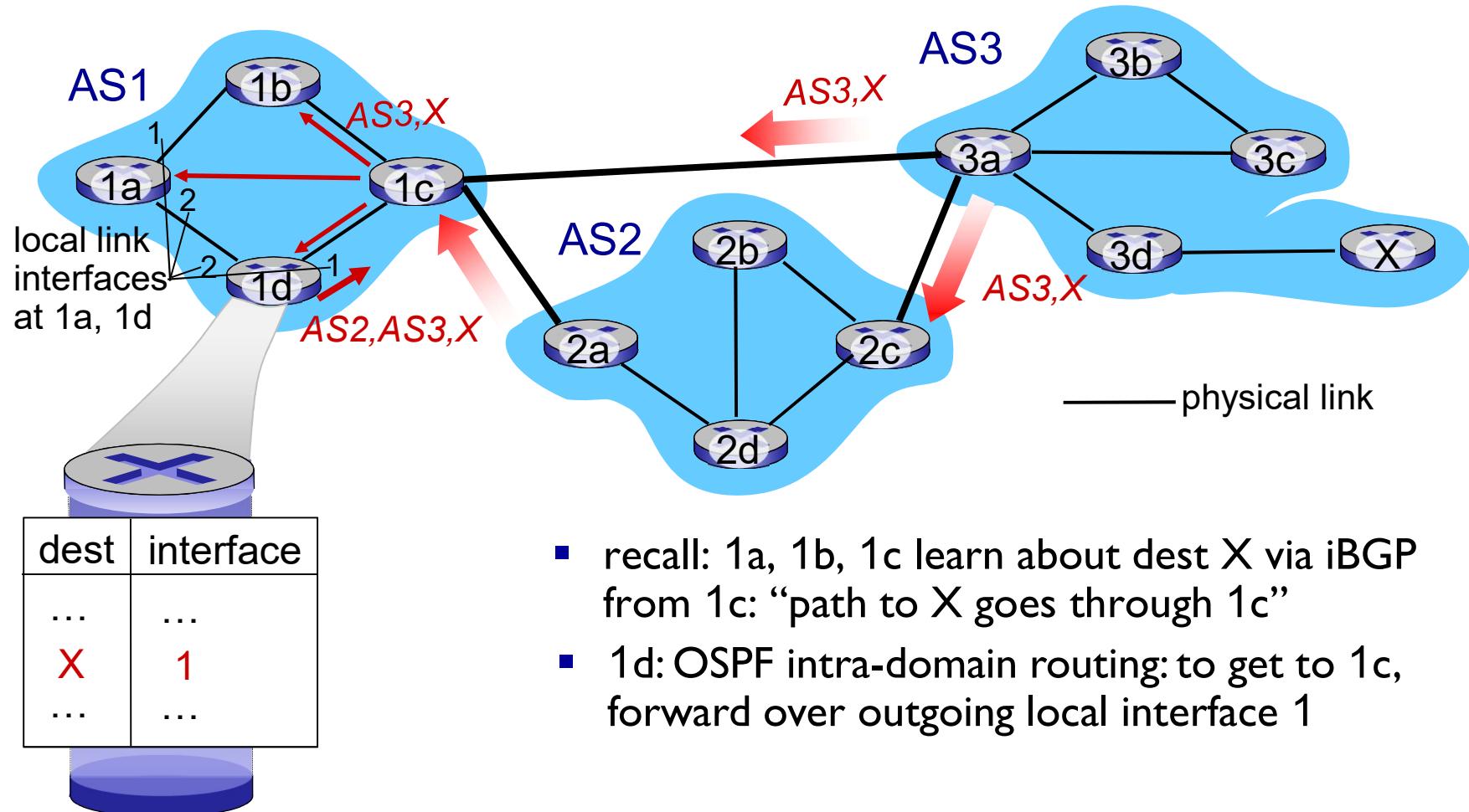
- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a
- AS1 gateway router 1c learns path **AS3,X** from 3a
- Based on policy, AS1 gateway router 1c chooses path **AS3,X, and advertises path within AS1 via iBGP**

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

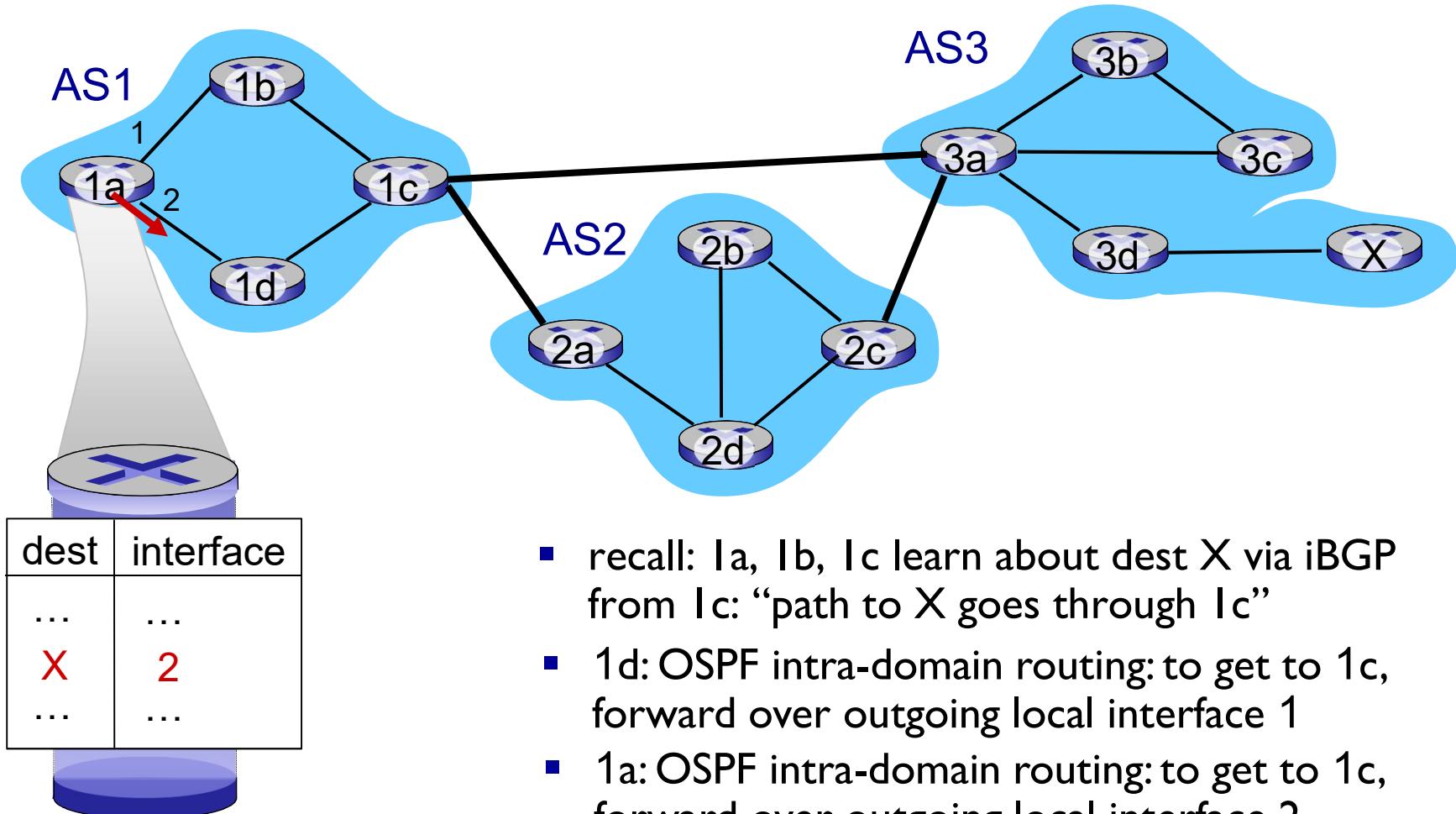
BGP, OSPF, forwarding table entries

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



BGP, OSPF, forwarding table entries

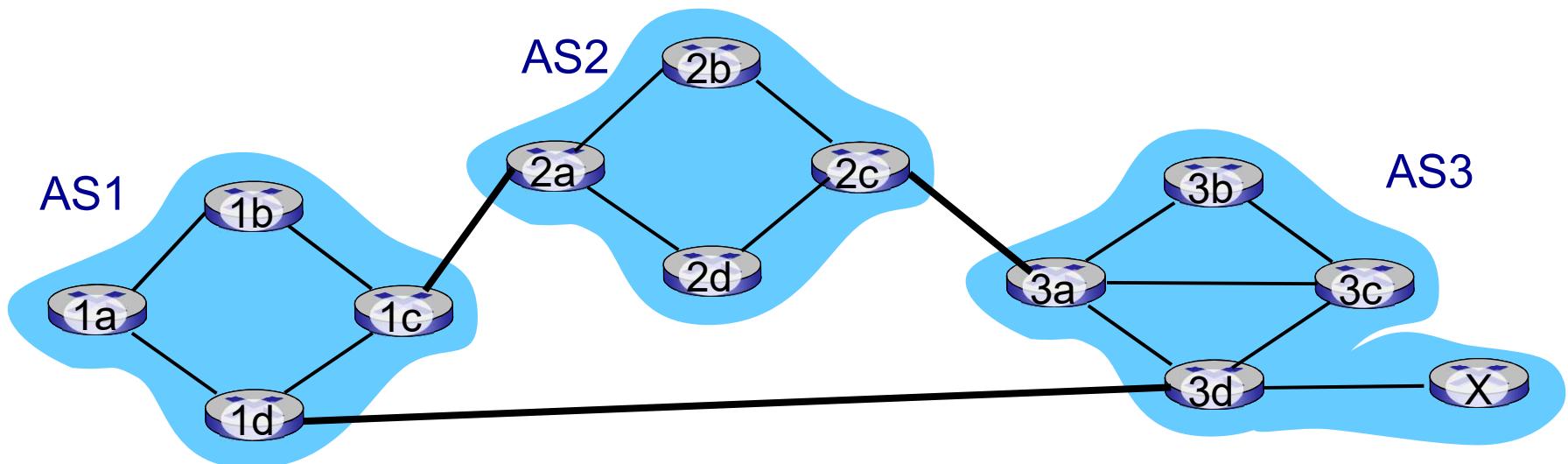
Q: how does router set forwarding table entry to distant prefix?



- 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

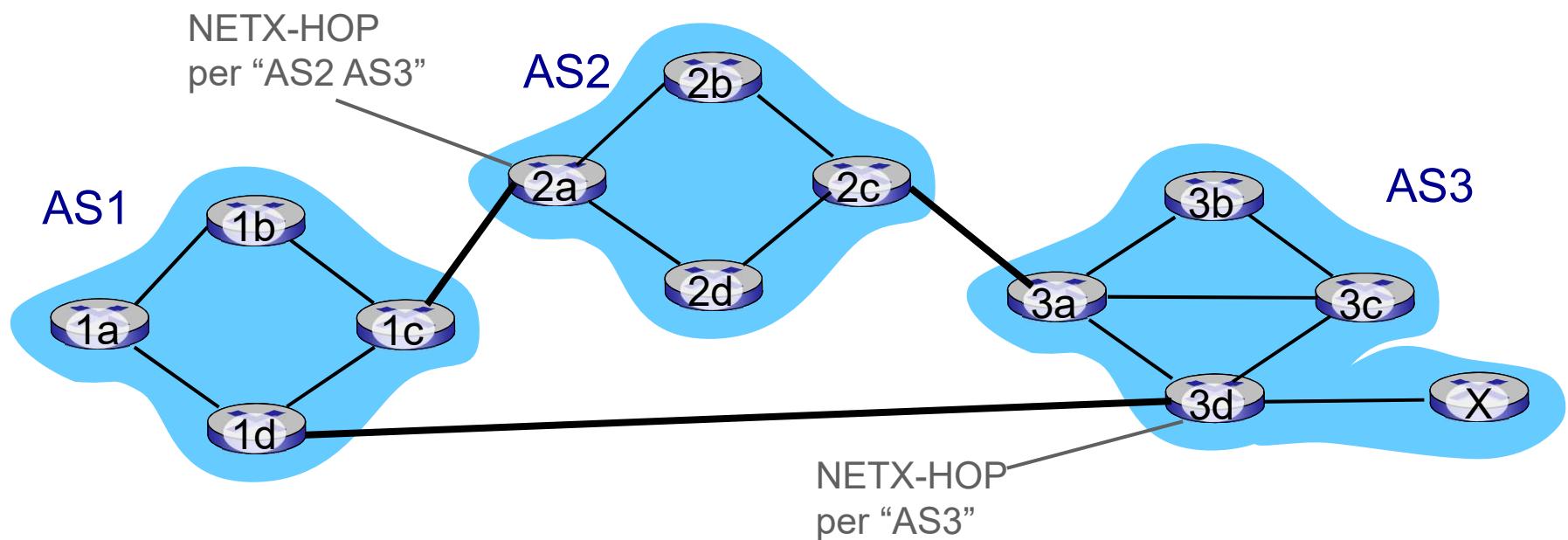
BGP route selection

- 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

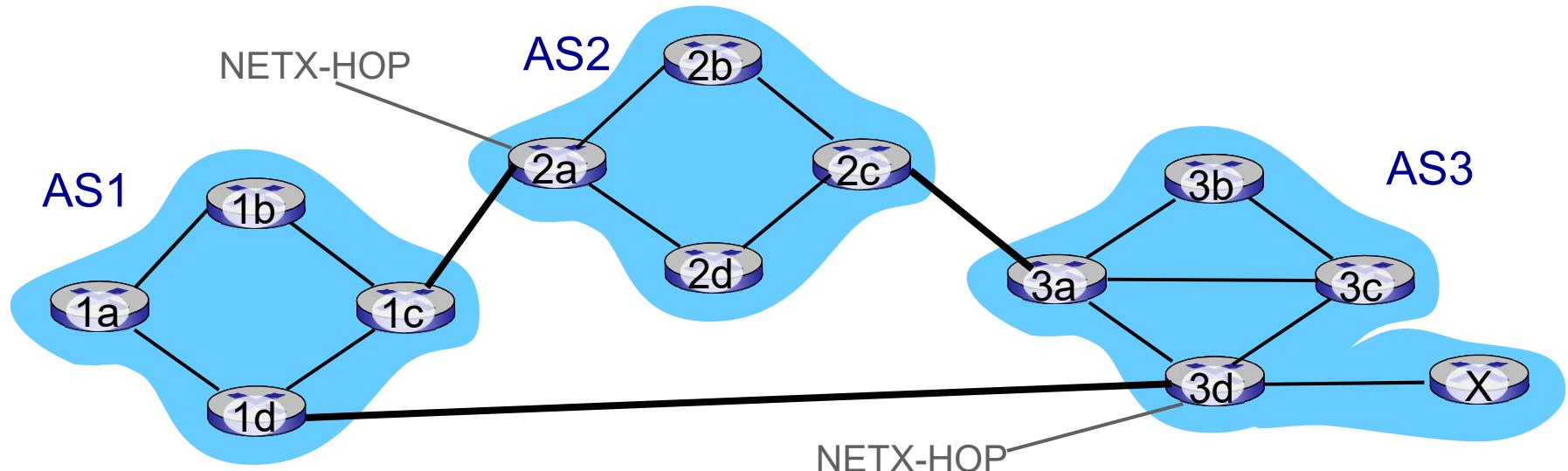


BGP route selection (II)

Attributo NEXT-HOP: closest NEXT-HOP router
(hot potato routing)

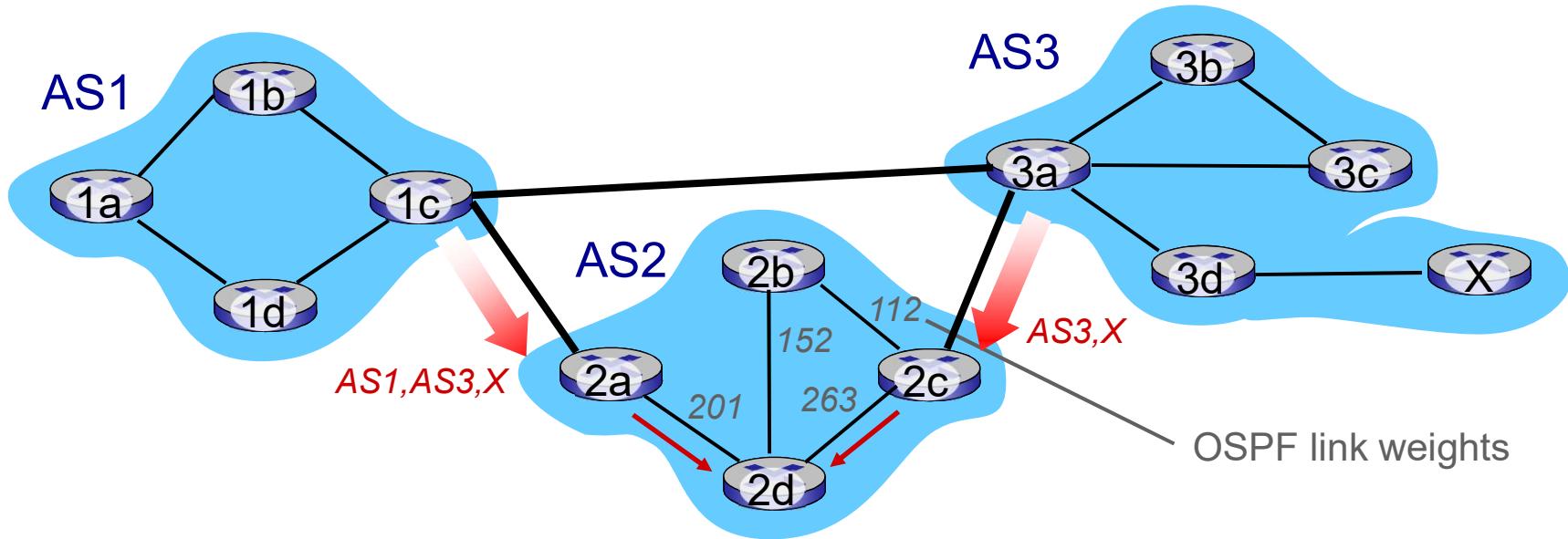


Hot Potato Routing



- 1b impara due percorsi verso il prefisso X: uno via 2a (NEXT-HOP) e l'altro via 3d (NEXT-HOP)
 - Costo definito come numero di hop
 - Costo (intra-AS) $1b-2a=2$, costo $1b-3d=3$
- *hot potato routing*: sceglie il router che ha il minor costo (intra-domain) e quindi 1b per andare verso X sceglie la strada via 2a (anche se il numero di AS attraversati verso x è maggiore). **Nella scelta i costi inter-domain non vengono considerati!**

Hot Potato Routing (II)



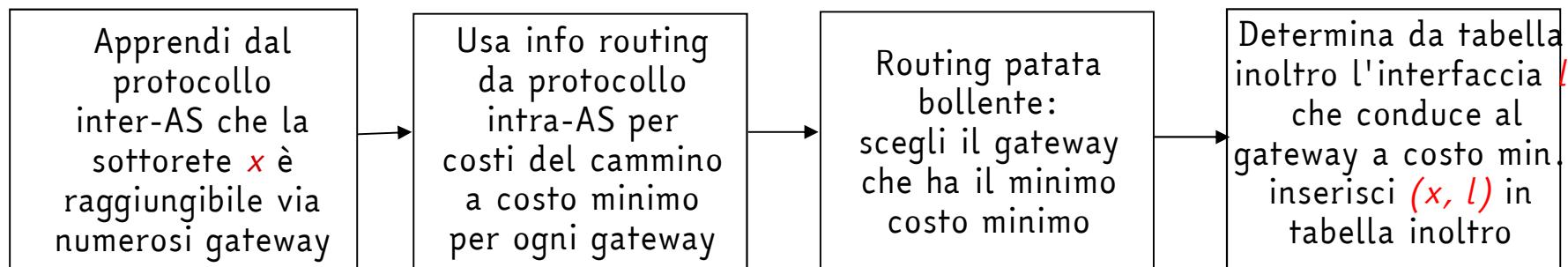
- 2d learns (via iBGP) it can route to X via 2a or 2c
- *hot potato routing*: choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

Hot Potato Routing (III)

Ora supponiamo che AS1 apprenda dal protocollo inter-AS che la sottorete **x** sia raggiungibile sia attraverso AS3 che attraverso AS2.

Per configurare la tabella di inoltro, il router 1d deve determinare a quale gateway inoltrare i datagrammi per la destinazione **x**

- Anche questo è compito del protocollo di instradamento inter-AS!
- *Intradamento a patata bollente*: invia il datagramma al più vicino dei due router



BGP route selection - contd

In realtà BGP usa un algoritmo più complesso del routing hot potato

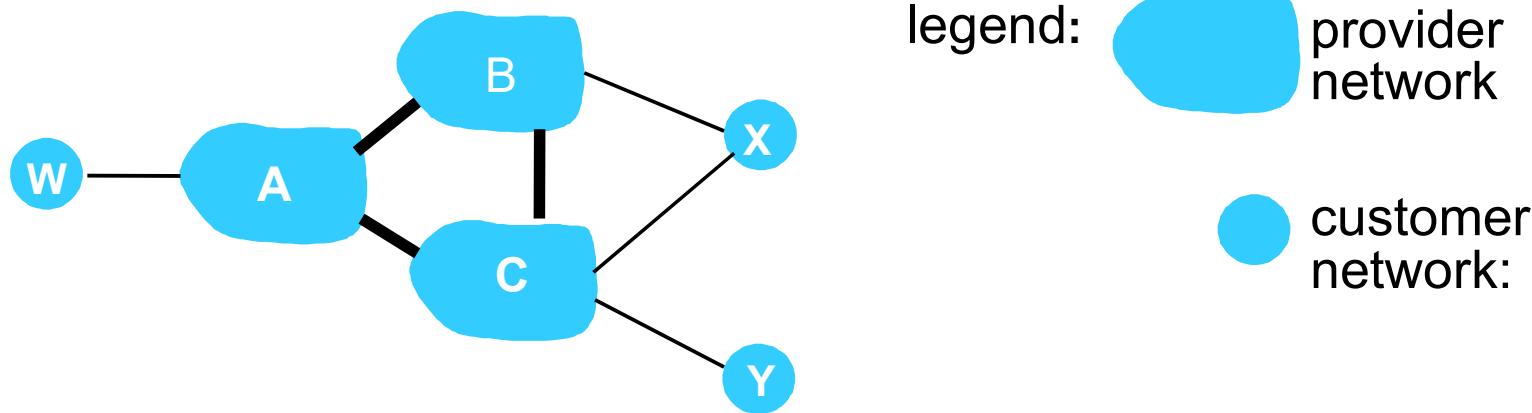
Alle rotte viene assegnato un valore di preferenza locale (scelta dell'amministratore dell'AS)

Tra le rotte con lo stesso valore di preferenza locale si seleziona quella con l'AS PATH più breve

Tra le rotte con lo stesso valore di preferenza e la stessa lunghezza di AS PATH si seleziona quella con il NEXT-HOP più vicino (hot potato routing)

Tabelle di instradamento BGP tratte da un router di ISP di livello I sono disponibili all'indirizzo <http://www.routeviews.org>

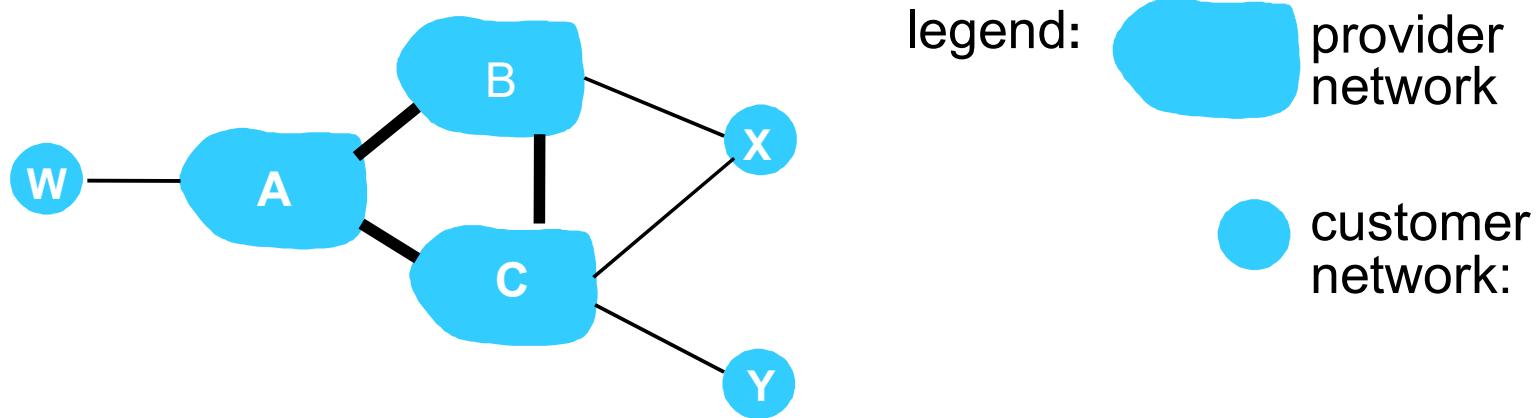
BGP: achieving policy via advertisements



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* 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

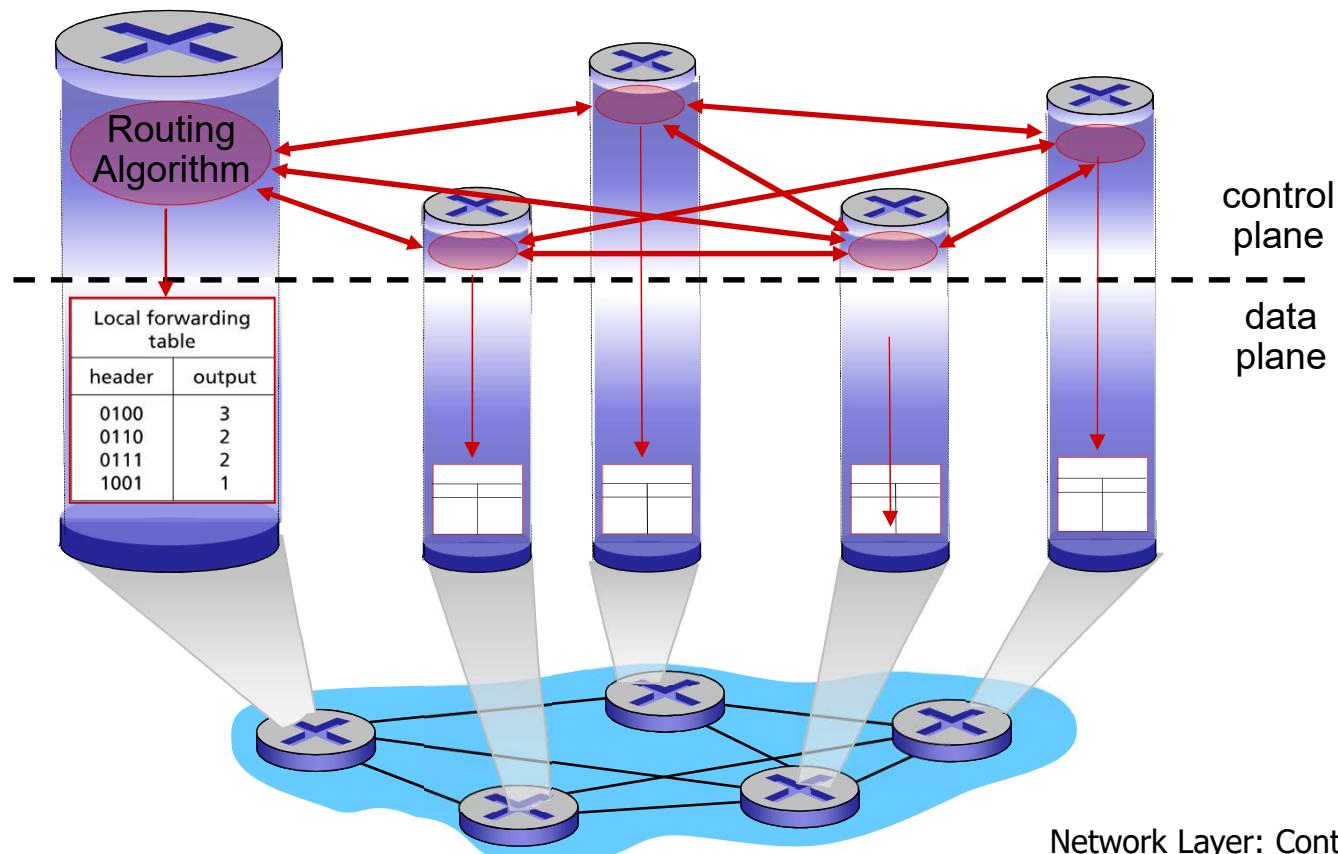
5.7 Network management and SNMP

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

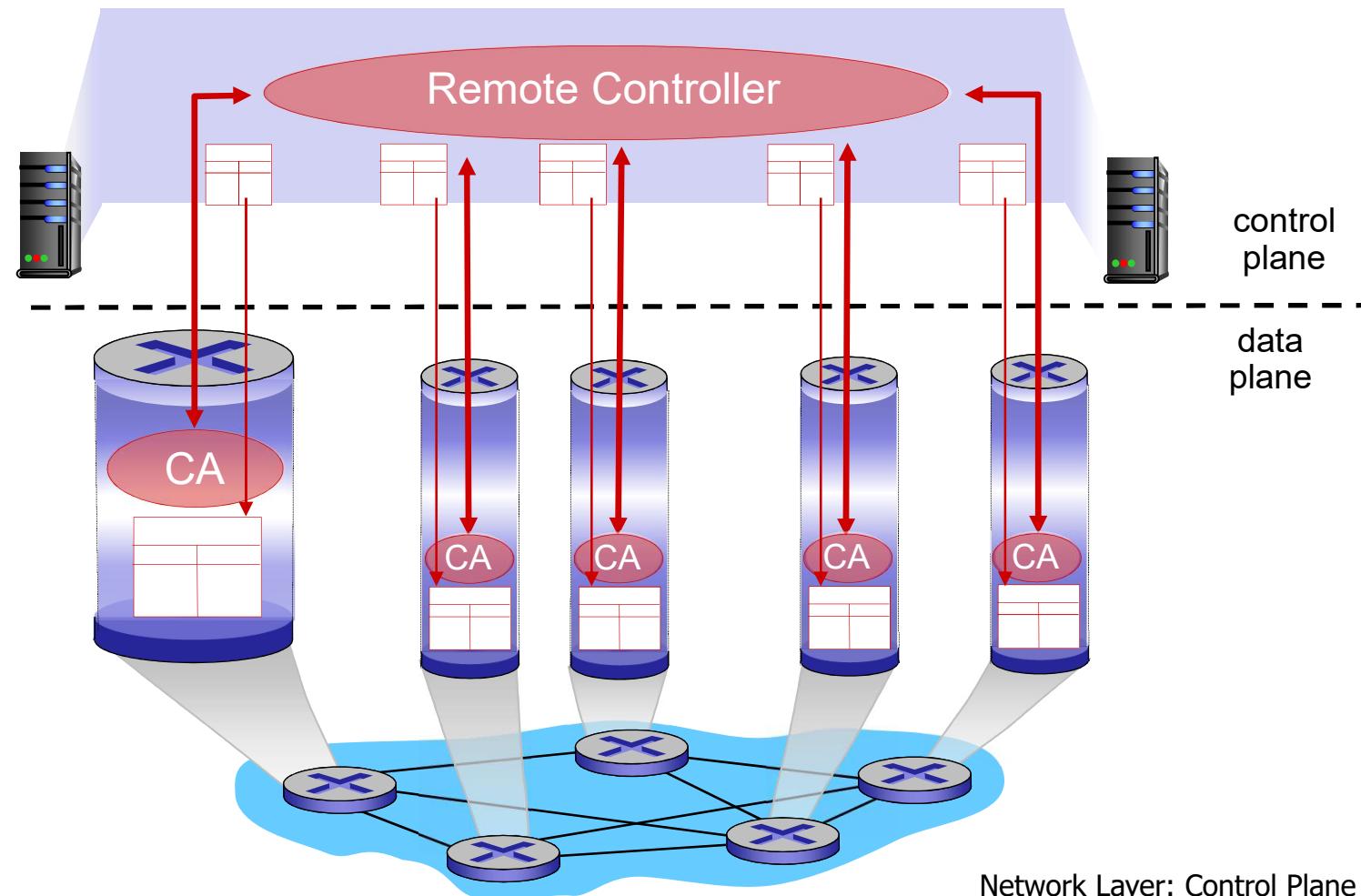
Recall: per-router control plane

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)

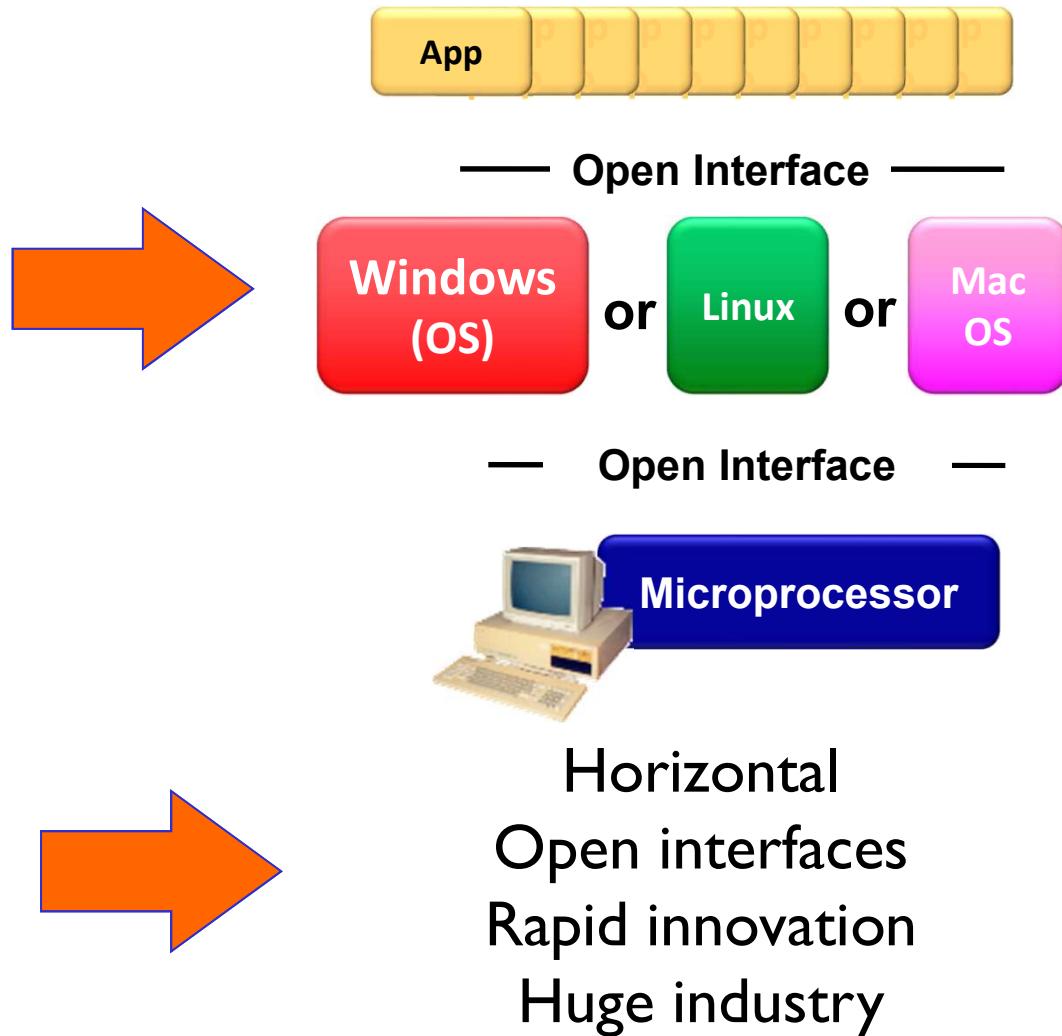
*Why a **logically centralized** control plane?*

- 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

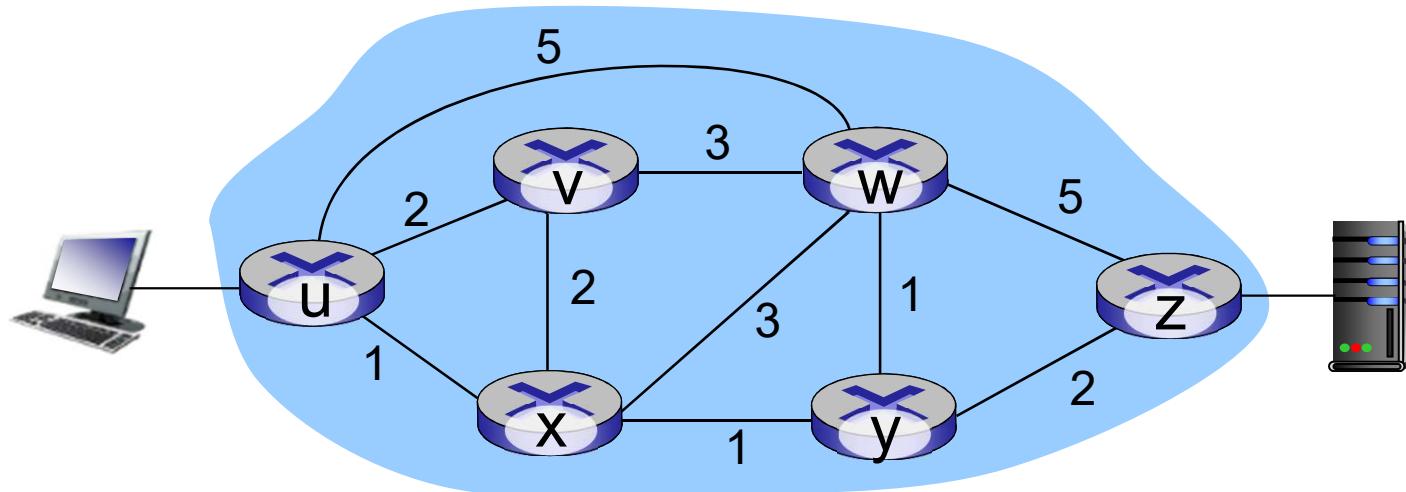
Analogy: mainframe to PC evolution*



Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Traffic engineering: difficult traditional routing

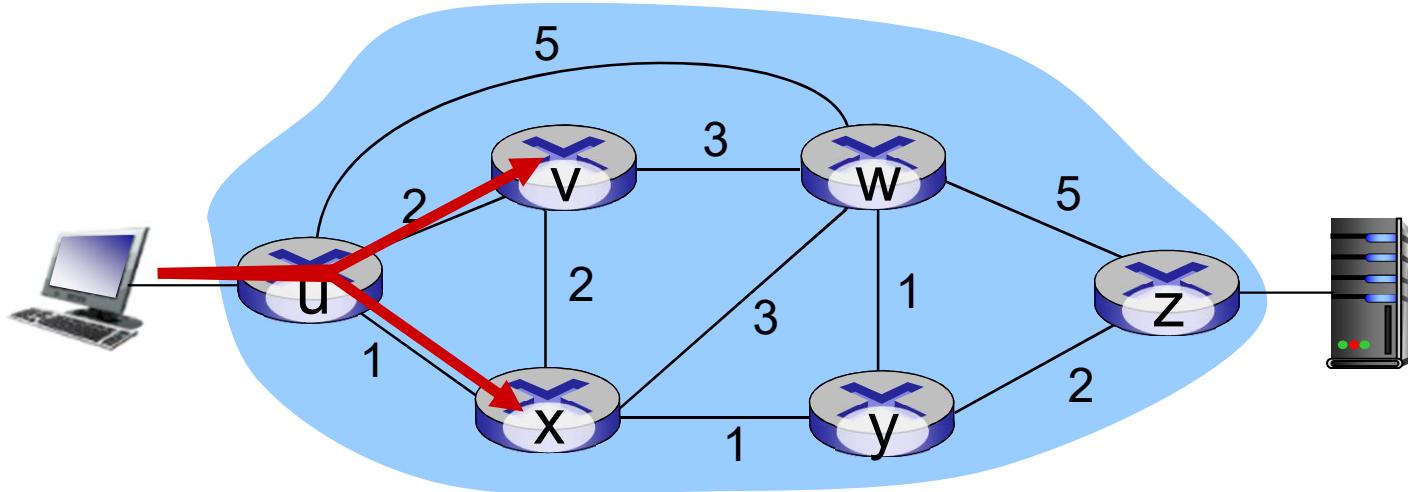


Q: what if network operator wants u-to-z traffic to flow along $uvwz$, x-to-z traffic to flow $xwyz$?

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

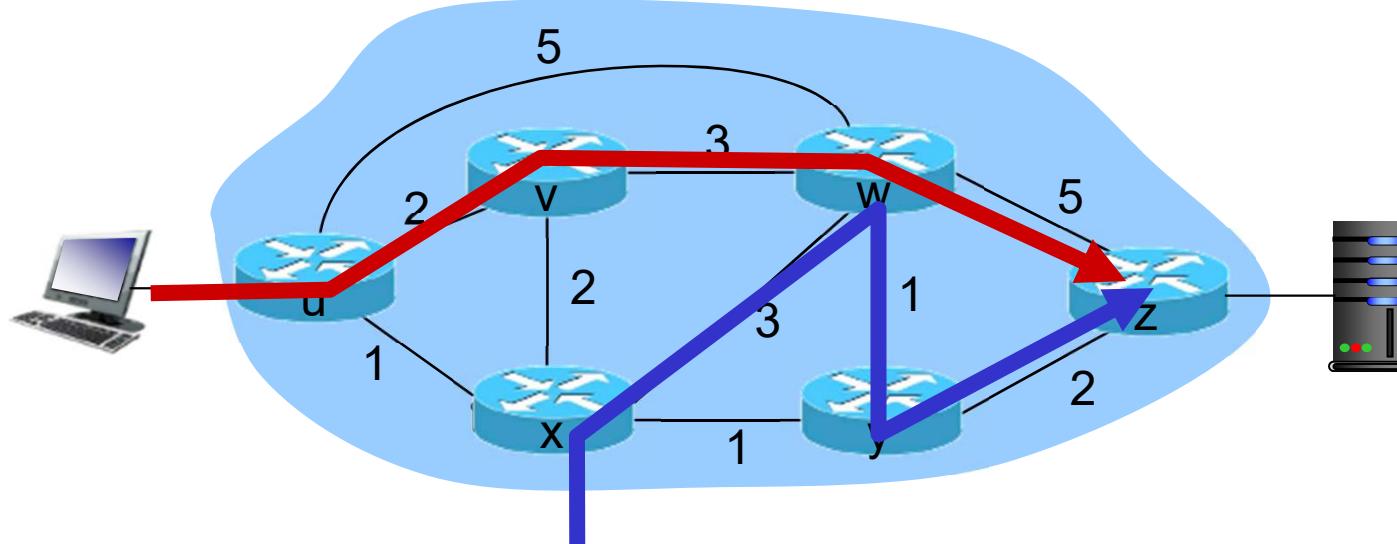
Link weights are only control “knobs”: wrong!

Traffic engineering: difficult



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

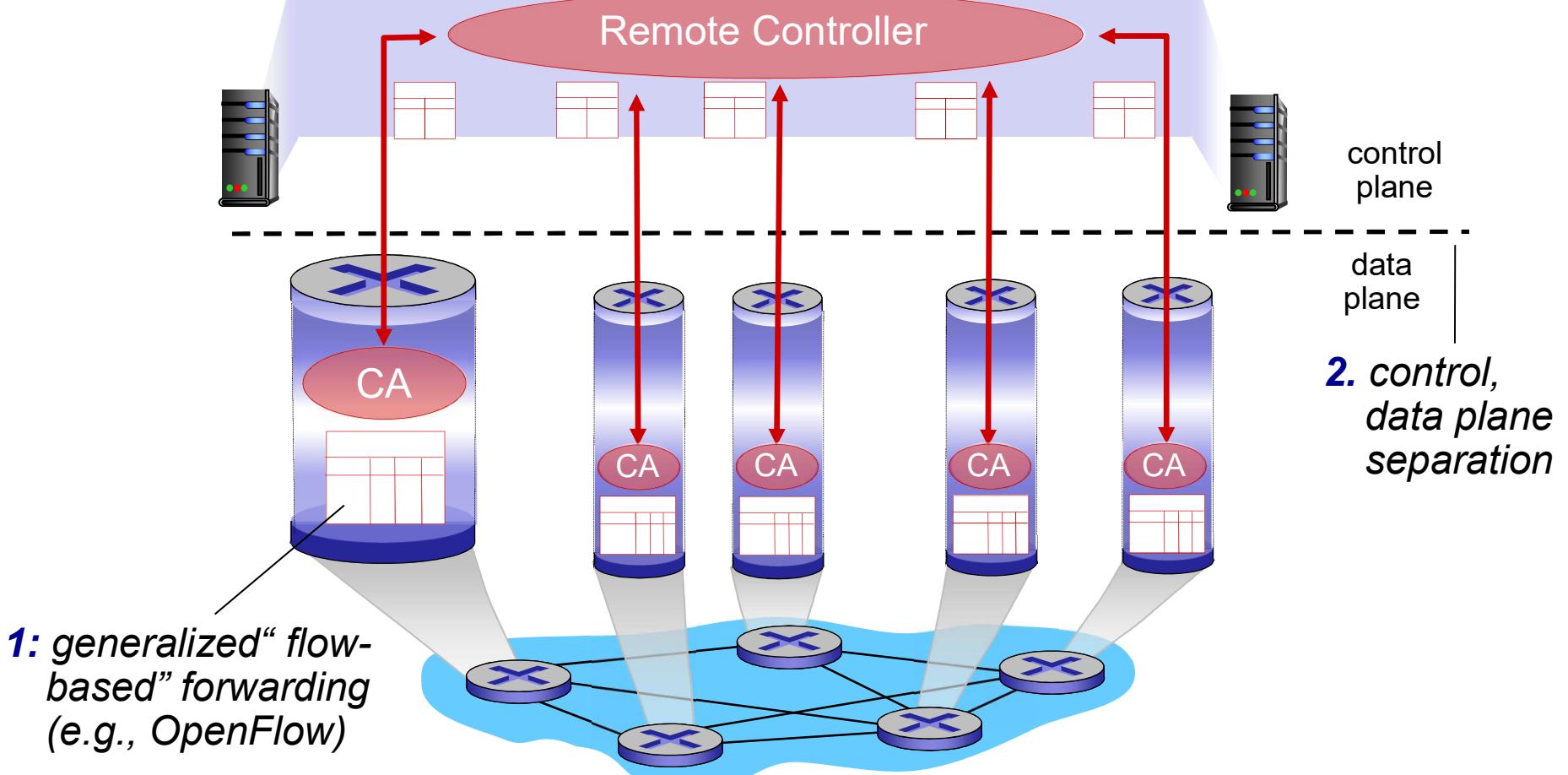
routing

access control

...

load balance

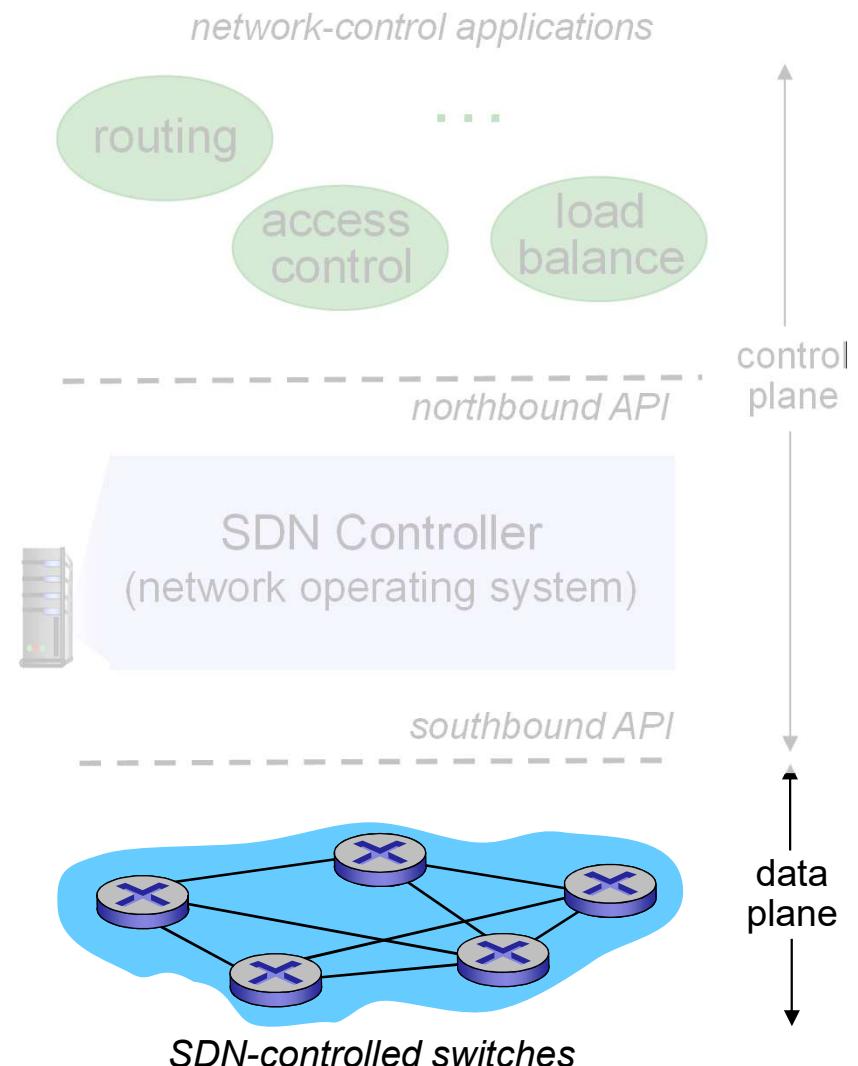
3. control plane functions external to data-plane switches



SDN perspective: data plane switches

Data plane switches

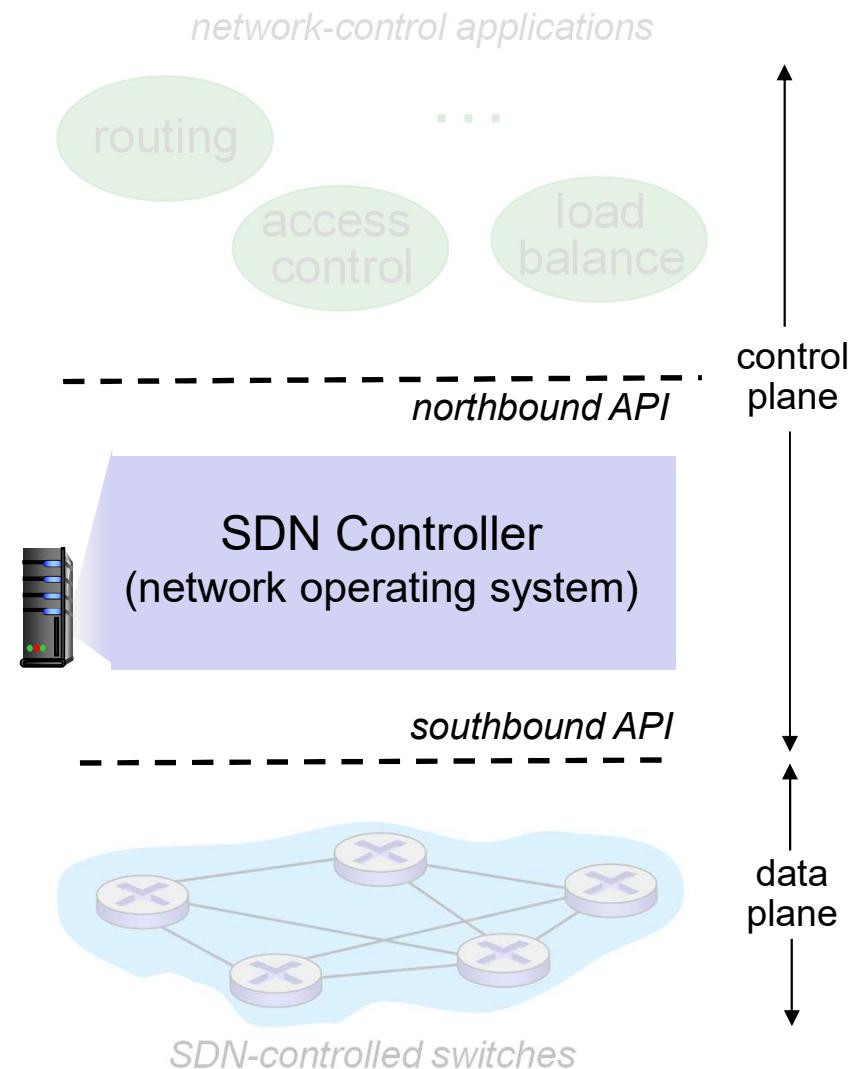
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable and what is not
- protocol for communicating with controller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



SDN perspective: SDN controller

SDN controller (network OS):

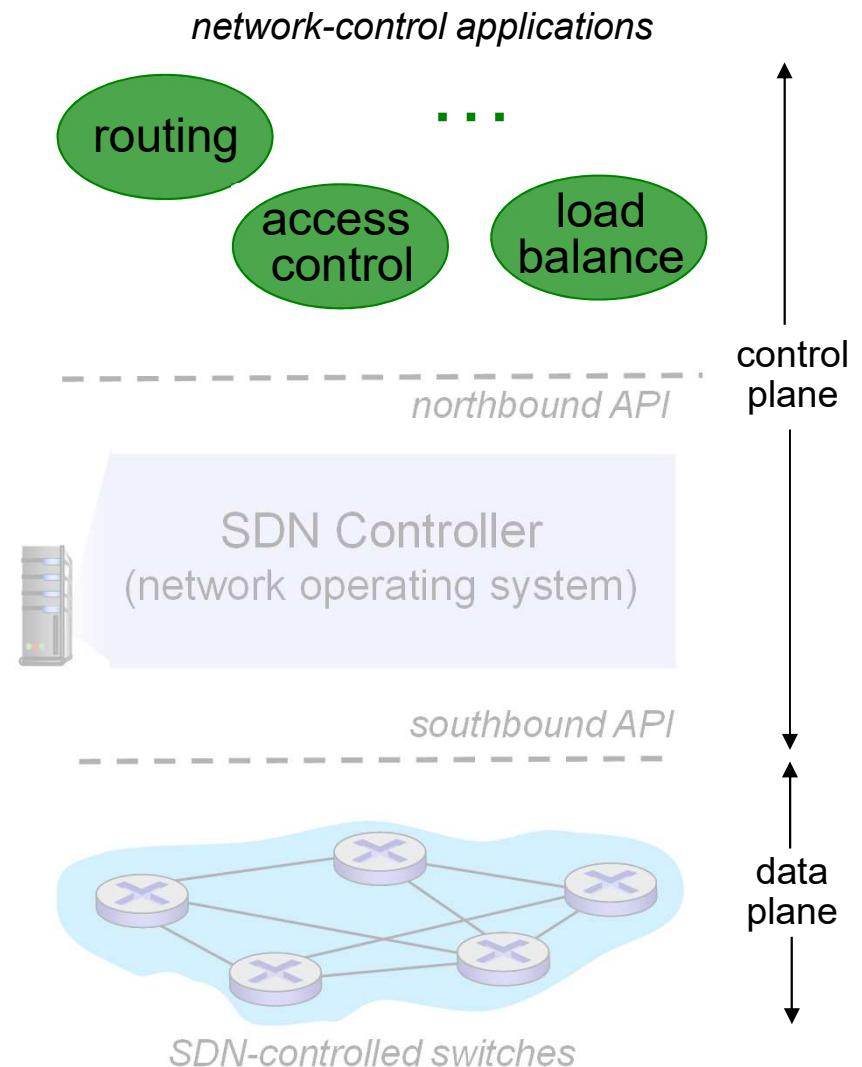
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



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 3rd party: distinct from routing vendor, or SDN controller

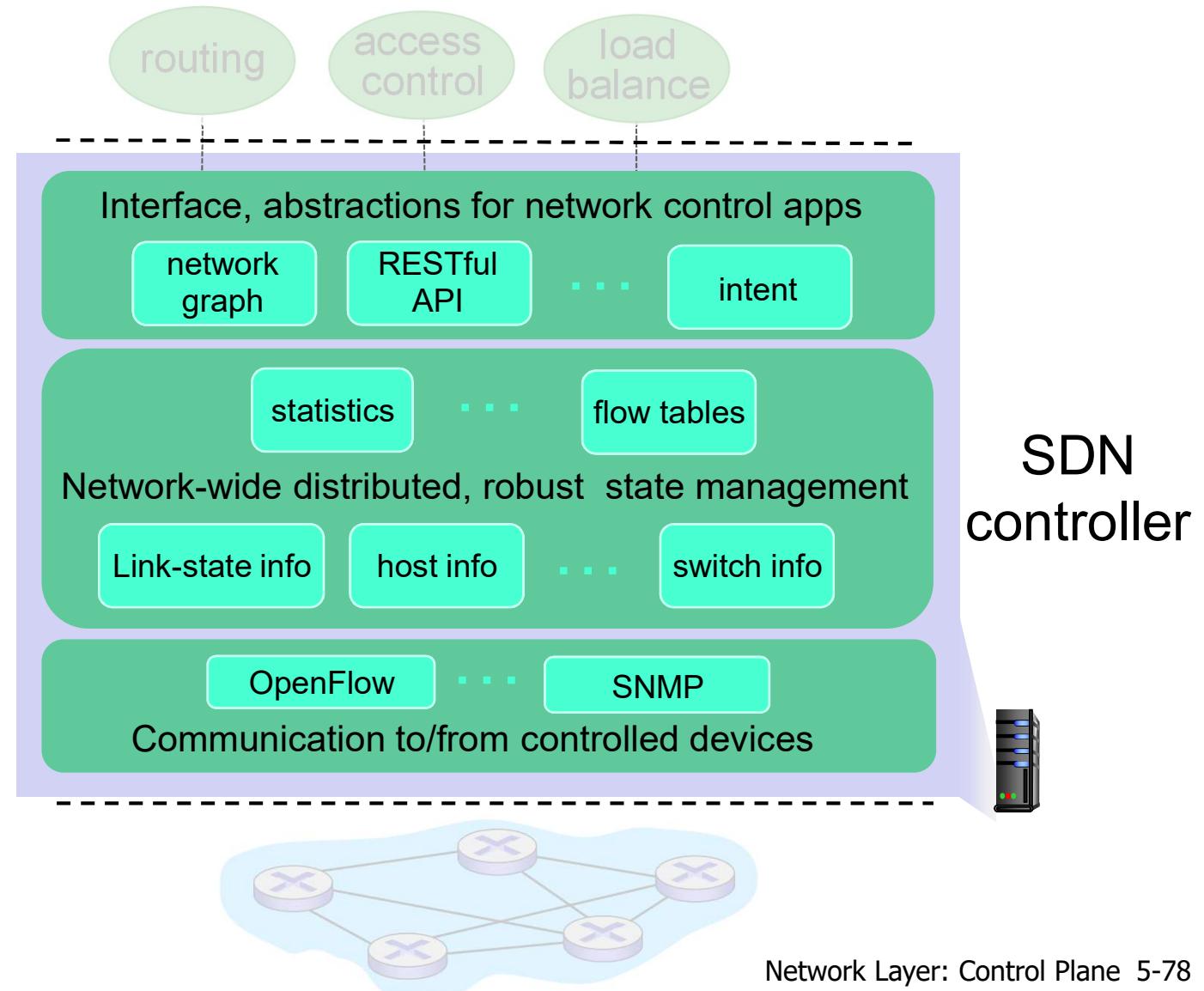


Components of SDN controller

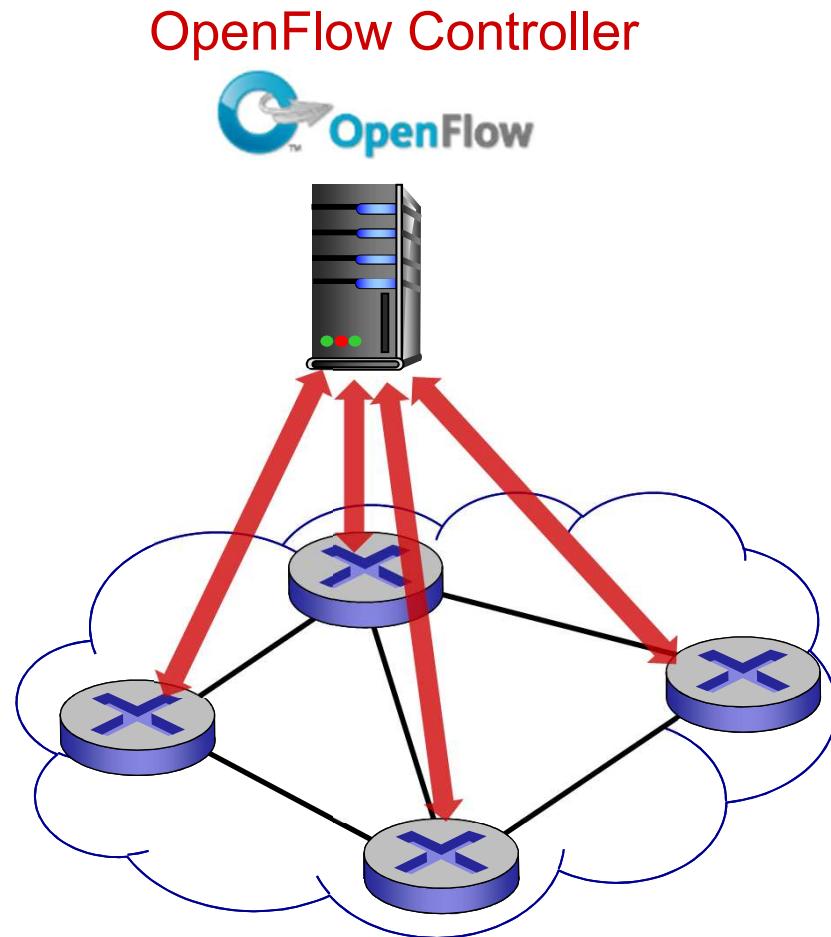
Interface layer to network control apps: abstractions API

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

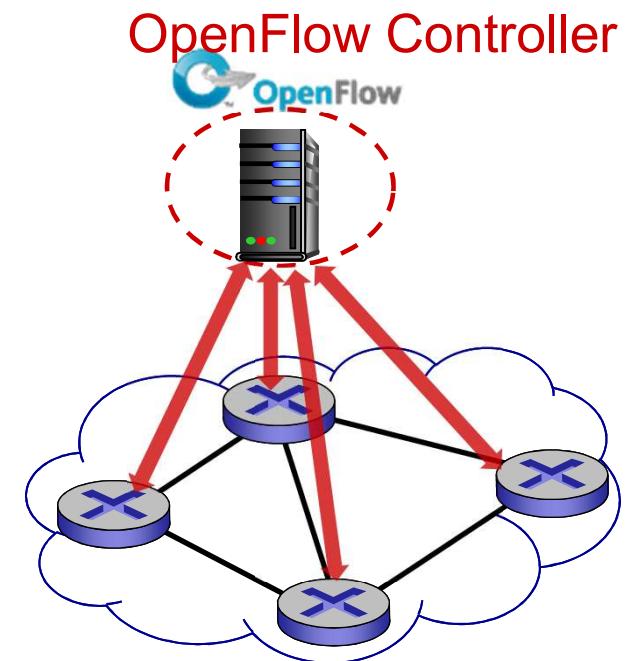


- operates between controller, switch
- TCP used to exchange messages
 - optional encryption
- three classes of OpenFlow messages:
 - controller-to-switch
 - asynchronous (switch to controller)
 - symmetric (misc)

OpenFlow: controller-to-switch messages

Key controller-to-switch messages

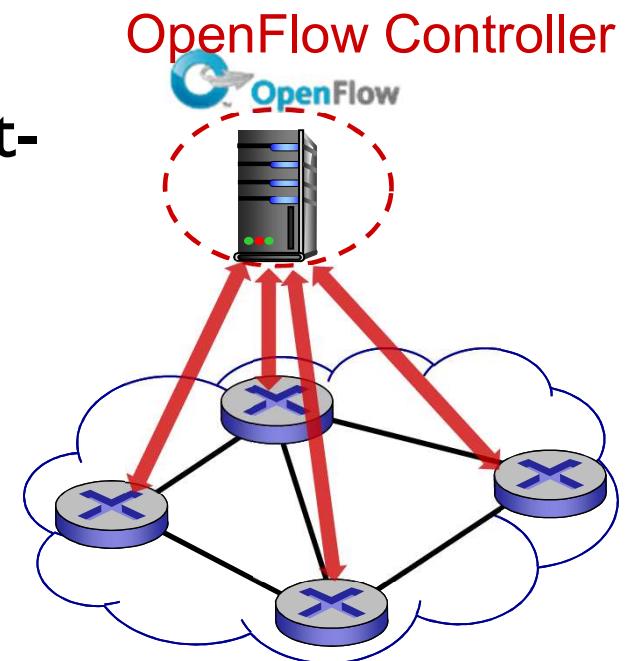
- **features**: controller queries switch features, switch replies
- **configure**: controller queries/sets switch configuration parameters
- **modify-state**: add, delete, modify flow entries in the OpenFlow tables
- **packet-out**: controller can send this packet out of specific switch port



OpenFlow: switch-to-controller messages

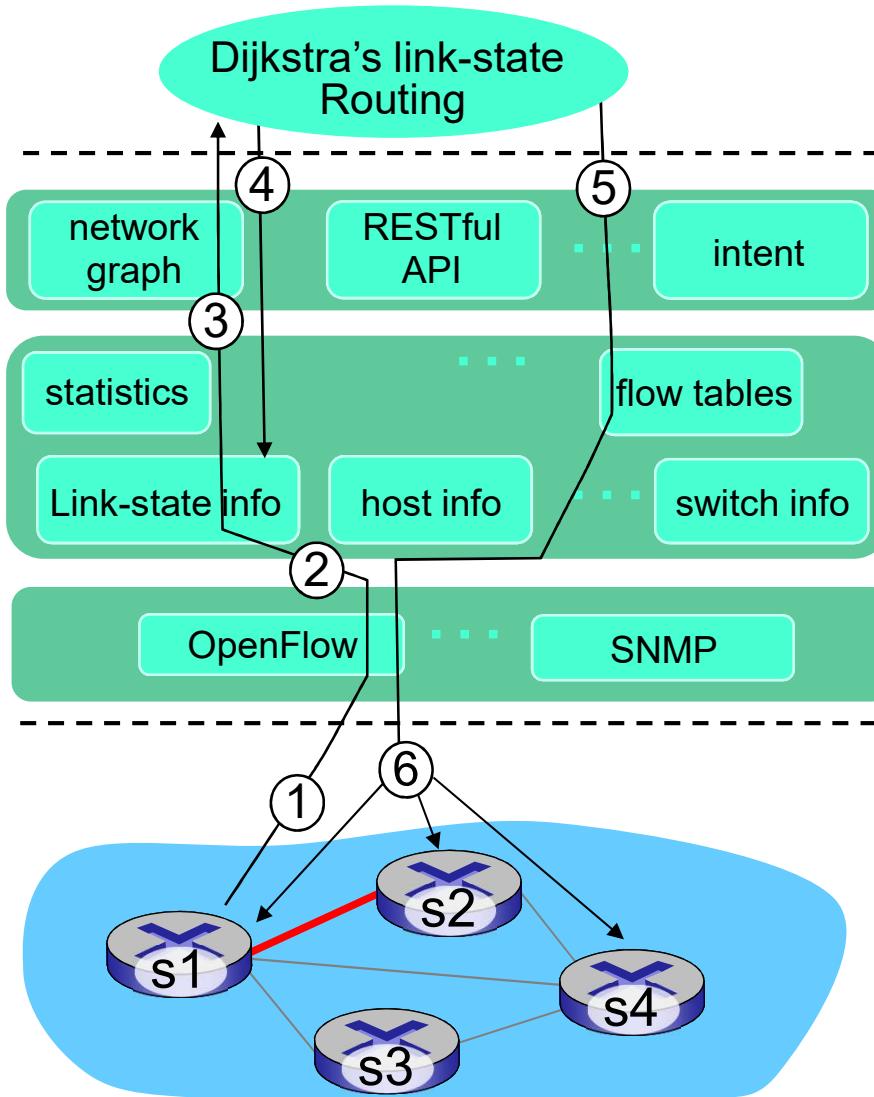
Key switch-to-controller messages

- **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.



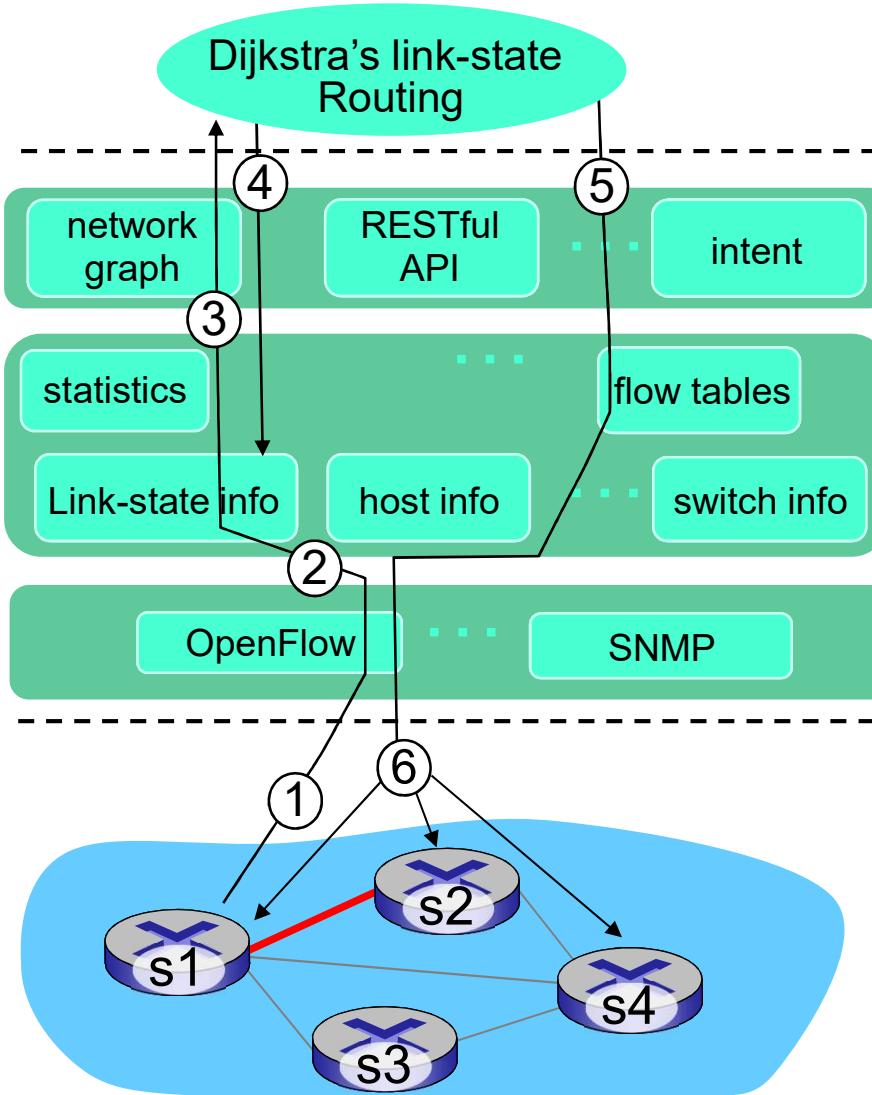
Fortunately, network operators don't “program” switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

SDN: control/data plane interaction example



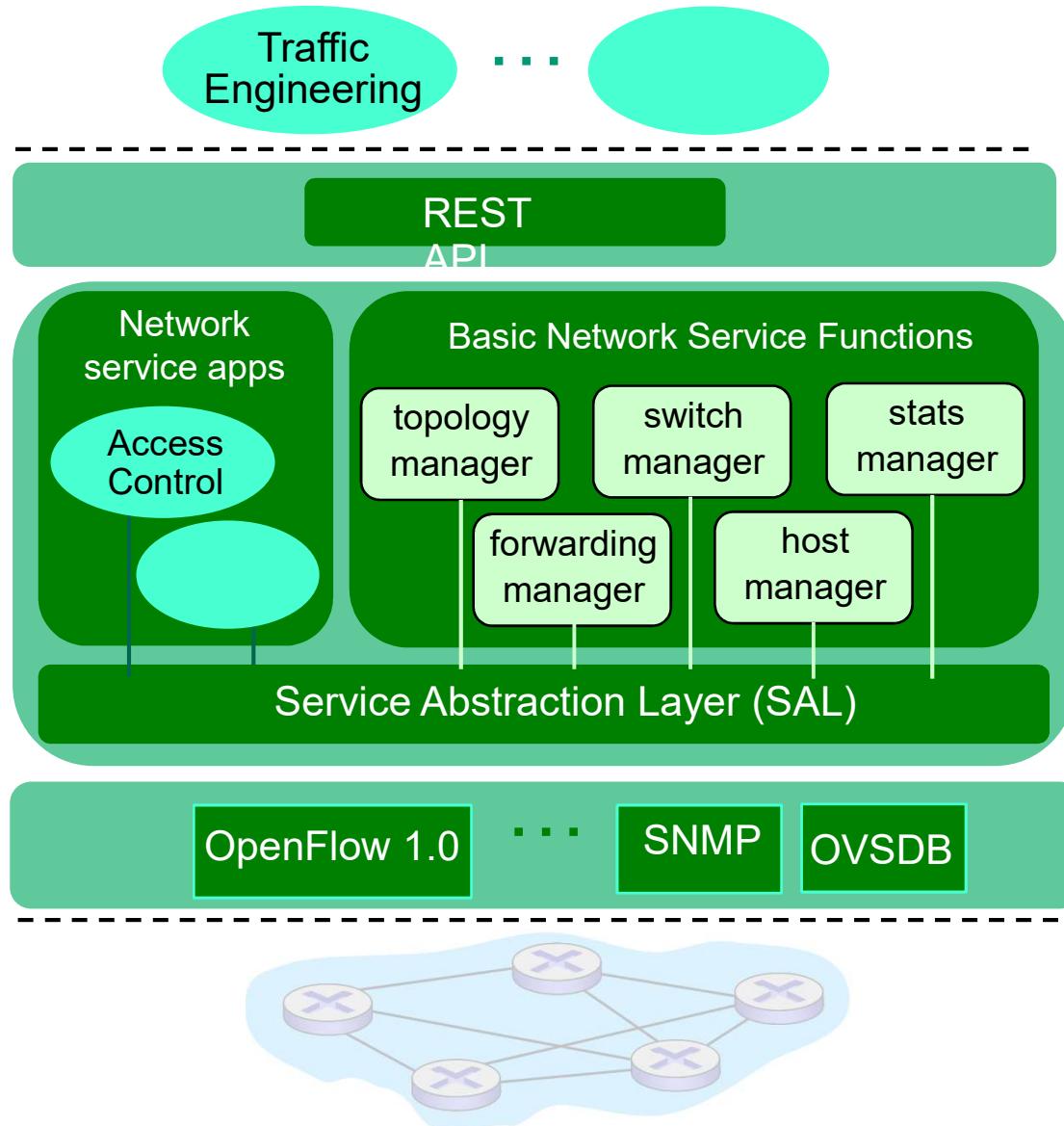
- ① S1, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
- ④ Dijkstra's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

SDN: control/data plane interaction example



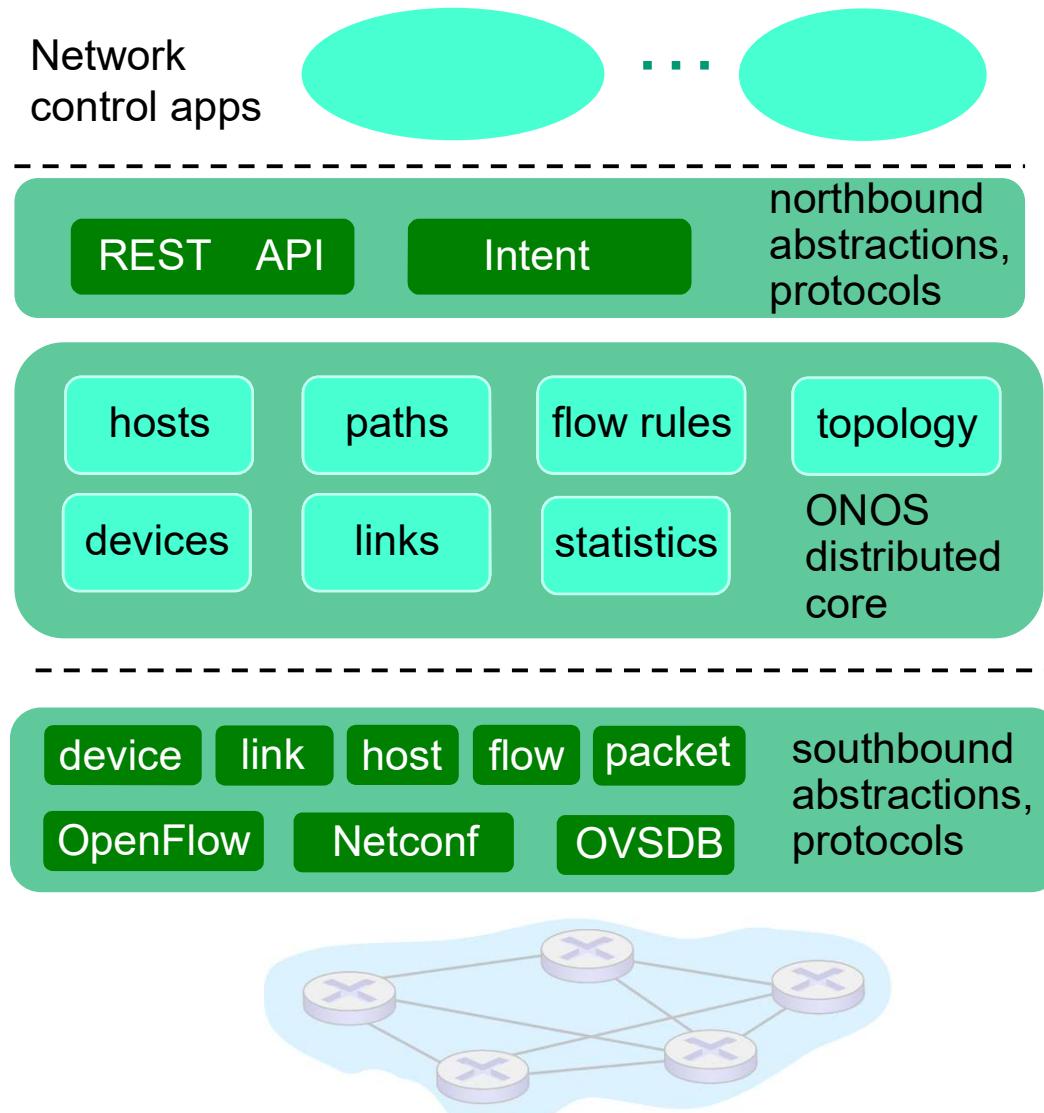
- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

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 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
- 5.7 Network management and 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

5.7 Network management and SNMP

What is network management?

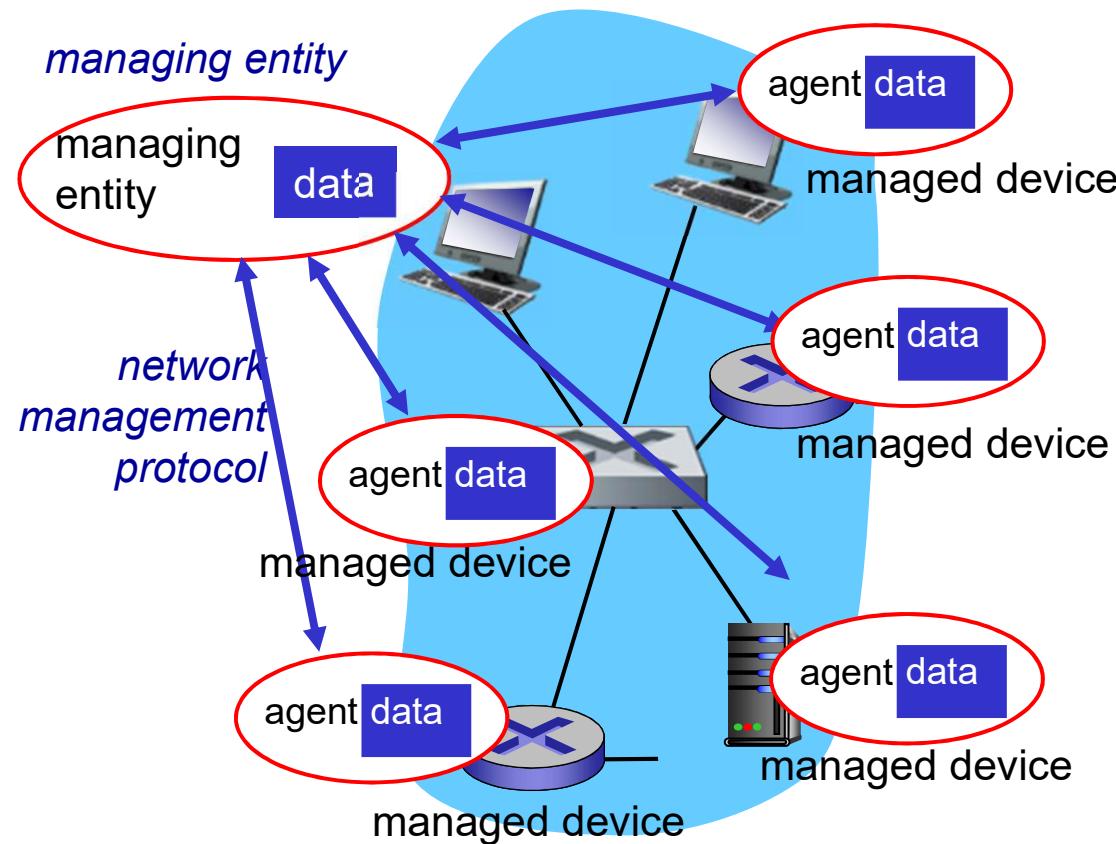
- **autonomous systems (aka “network”):** 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
 - jet airplane
 - nuclear power plant
 - others?



"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

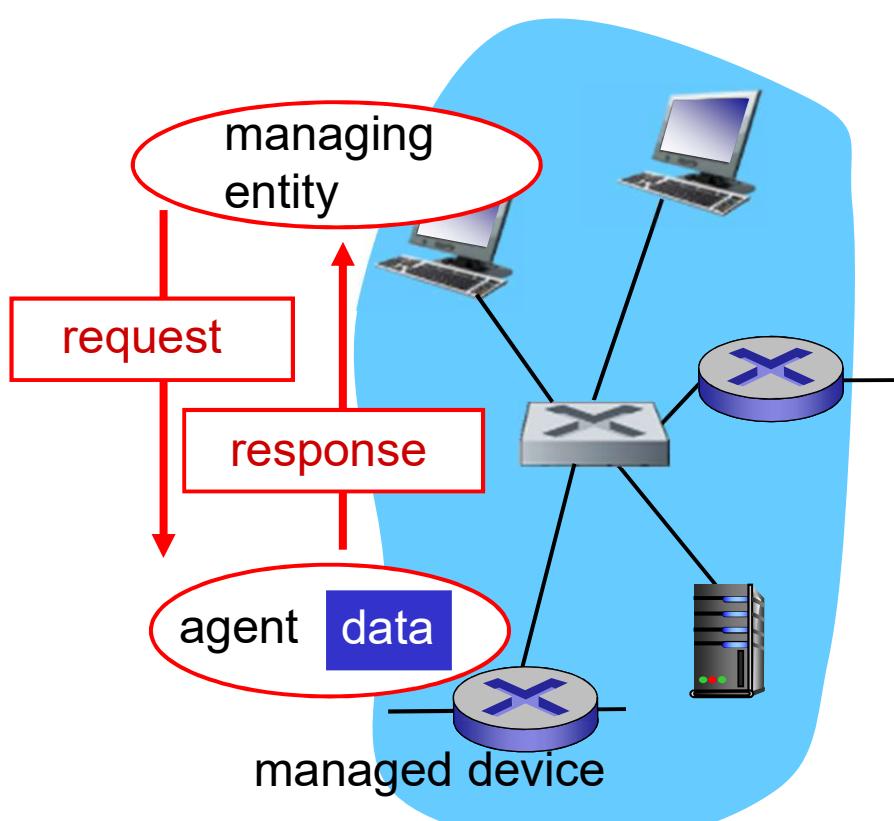
definitions:



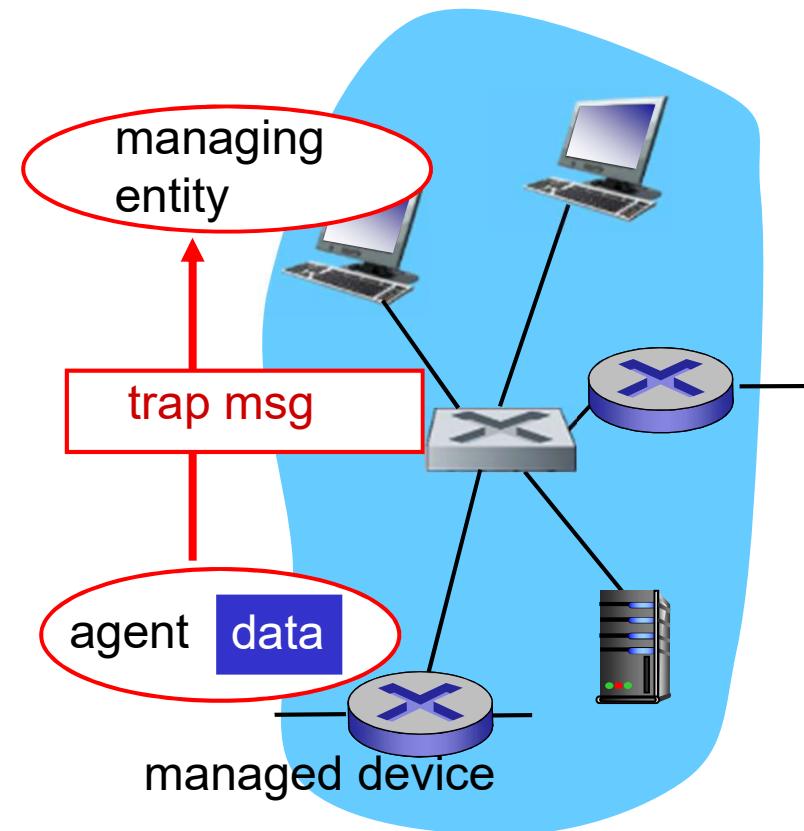
managed devices contain *managed objects* whose data is gathered into a *Management Information Base (MIB)*

SNMP protocol

Two ways to convey MIB info, commands:



request/response mode

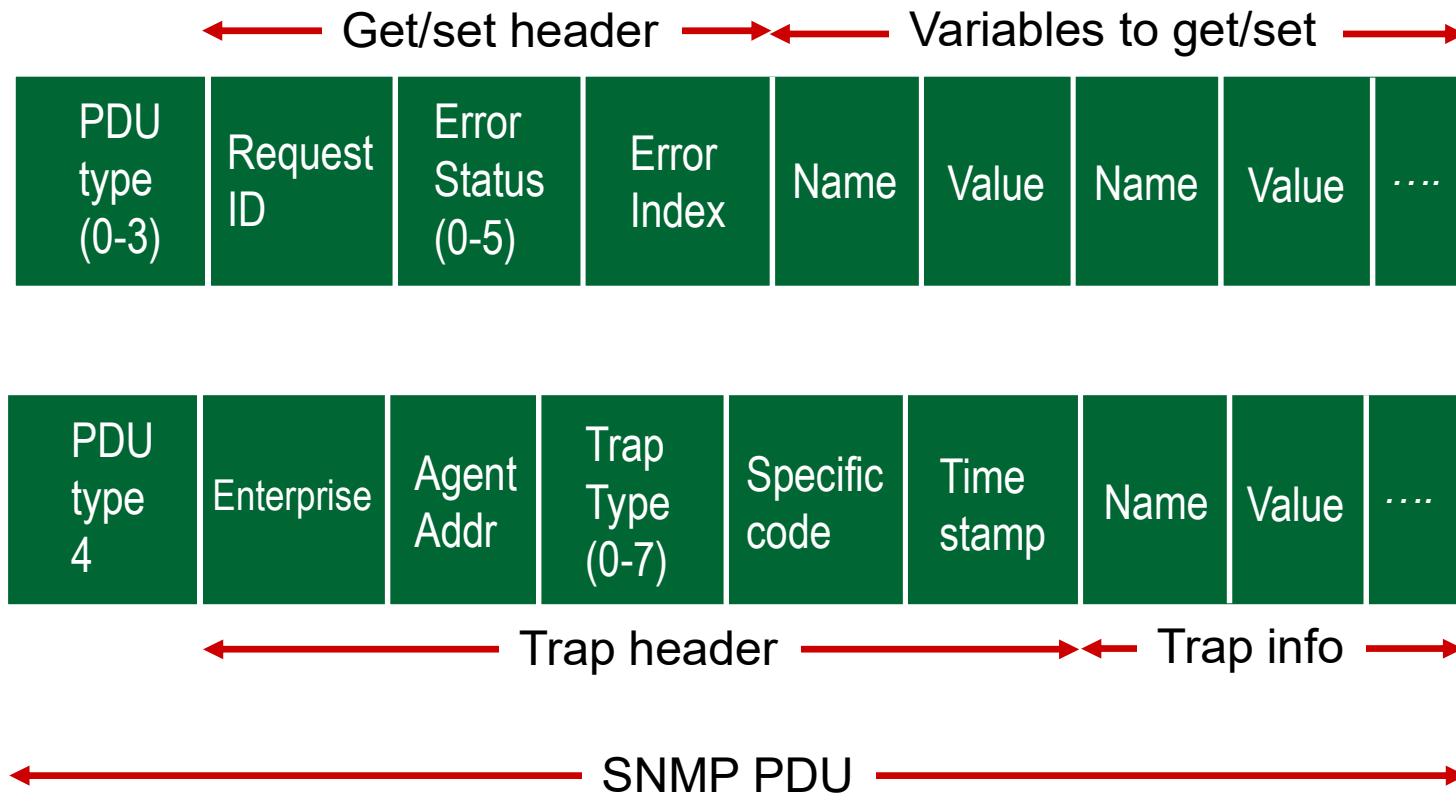


trap mode

SNMP protocol: message types

| <u>Message type</u> | <u>Function</u> |
|--|--|
| 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: summary

we've learned a lot!

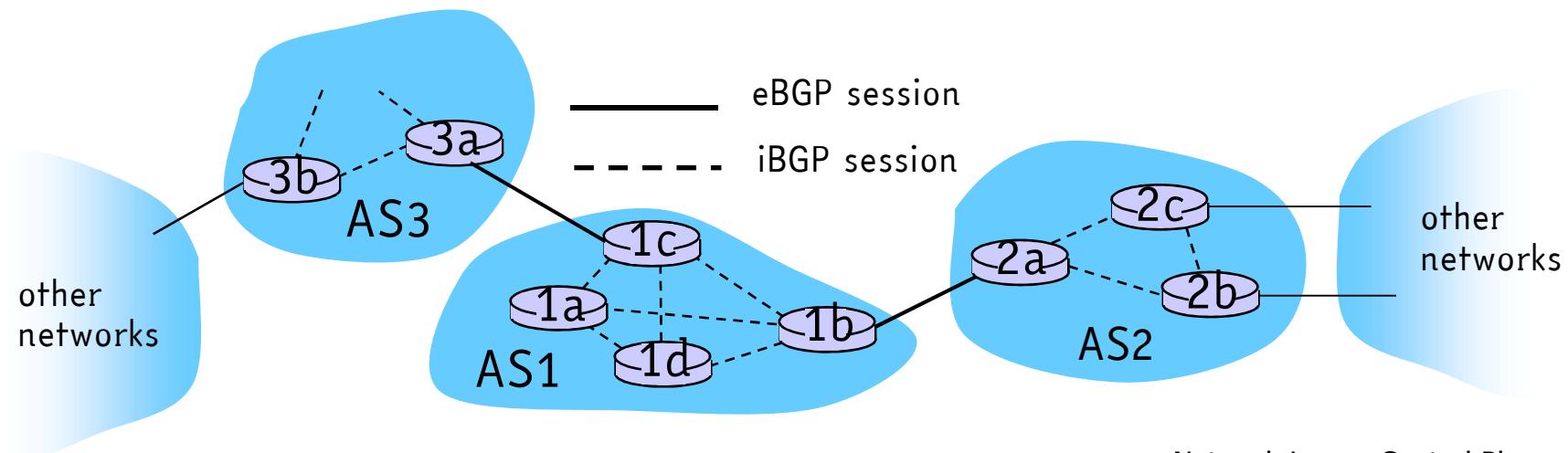
- approaches to network control plane
 - per-router control (traditional)
 - logically centralized control (software defined networking)
- traditional routing algorithms
 - implementation in Internet: OSPF, BGP
- SDN controllers
 - implementation in practice: ODL, ONOS
- network management

BGP: distribuzione dell'informazione

Usando una sessione **eBGP** fra 3a e 1c, AS3 invia la raggiungibilità di un prefisso ad AS1.

- 1c può allora usare **iBGP** per distribuire l'informazione sul nuovo prefisso a tutti i router in AS1, incluso 1b
- 1b può allora ri-annunciare ad AS2 la nuova informazione di raggiungibilità sulla sessione eBGP 1b-a-2a

Quando un router apprende un nuovo prefisso, crea una entry per quel prefisso nella sua tabella di inoltro



Esempio: tabella di inoltro del router Id

Supponiamo che AS1 impari (via eBGP) che la sottorete x sia raggiungibile via AS3 (NEXT-HOP interfaccia di 3a), ma non via AS2

- Il protocollo iBGP propaga la informazione di raggiungibilità a tutti router interni, fra cui Id

Interfaccia di 3a è propagata tramite ad es. OSPF (è di 1c)

Il router Id determina tramite OSPF che la propria interfaccia 1 è sul cammino a costo minimo per interfaccia di 3a (quindi tramite 1c)

- Installa nella tabella di instradamento la riga (x , 1)

