Chapter 5
Network Layer: The Control Plane

- Slides from J. Kurose and K. Ross based on their book Computer Networking: A Top Down Approach, 7th edition.

©

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.

Chapter 5: outline

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

Local forwarding table

header	output
0100	3
0110	2
0111	2
1001	1

Routing Algorithm

control plane

data plane
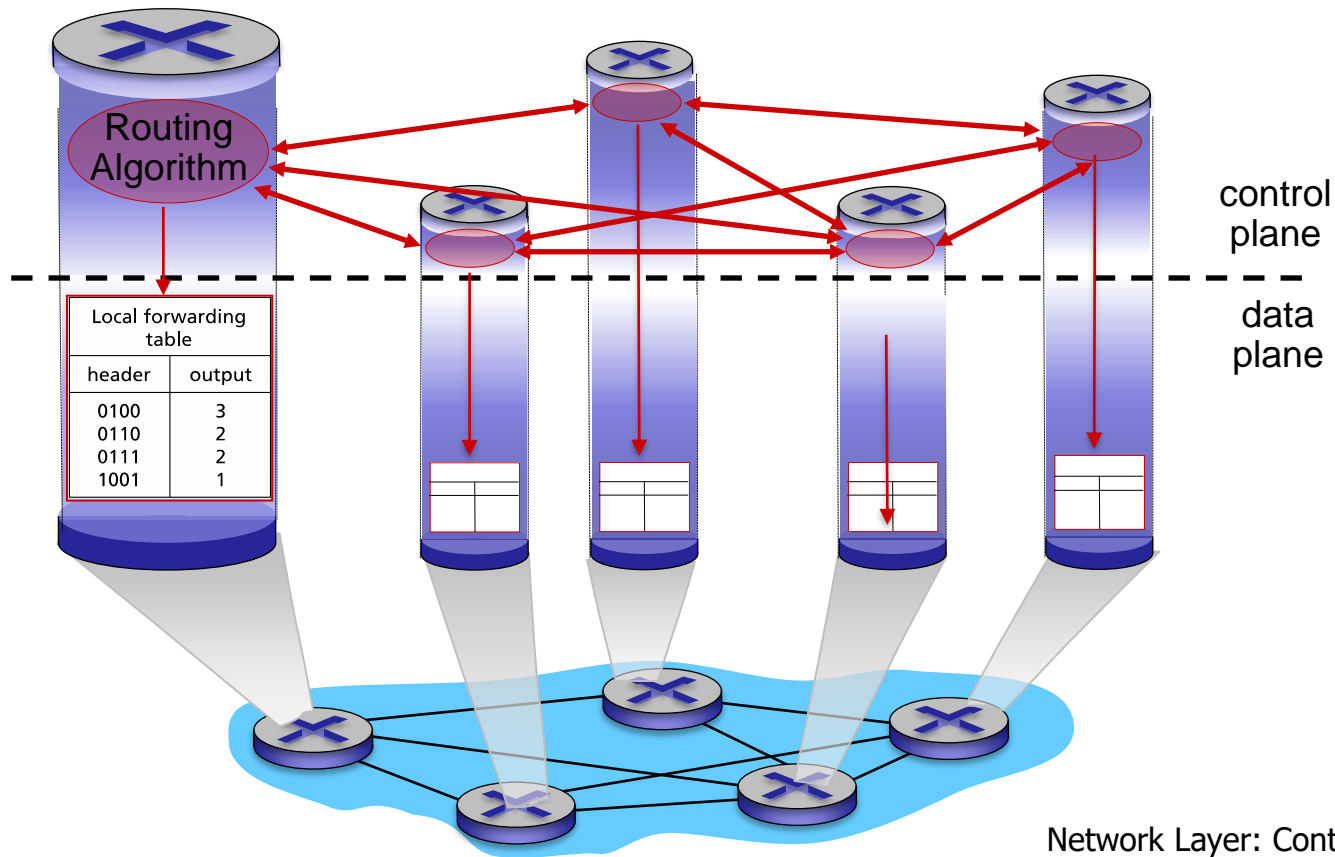
Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables
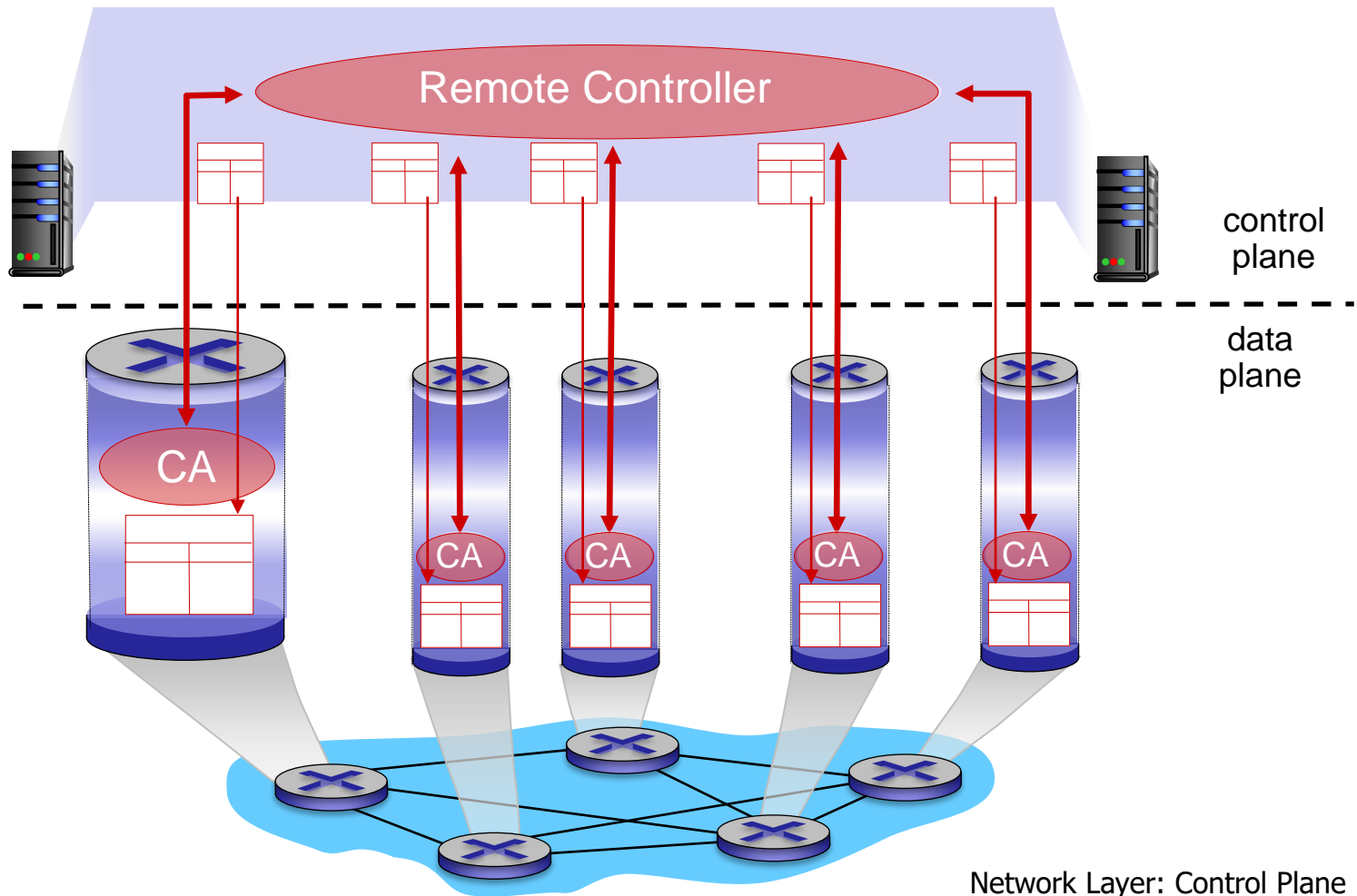
# Chapter 5: outline

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
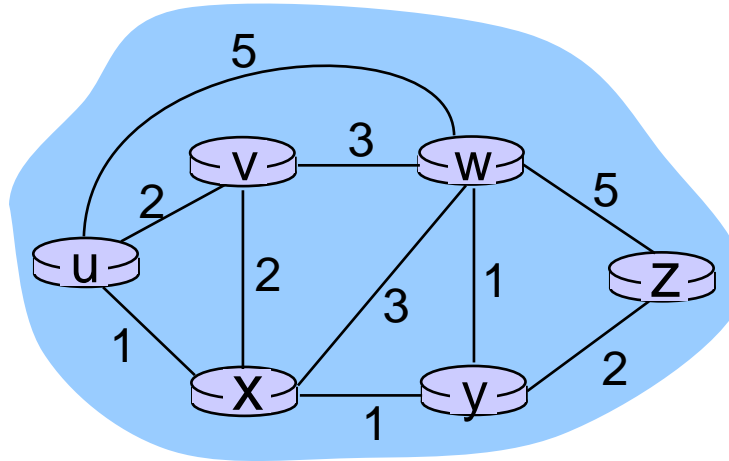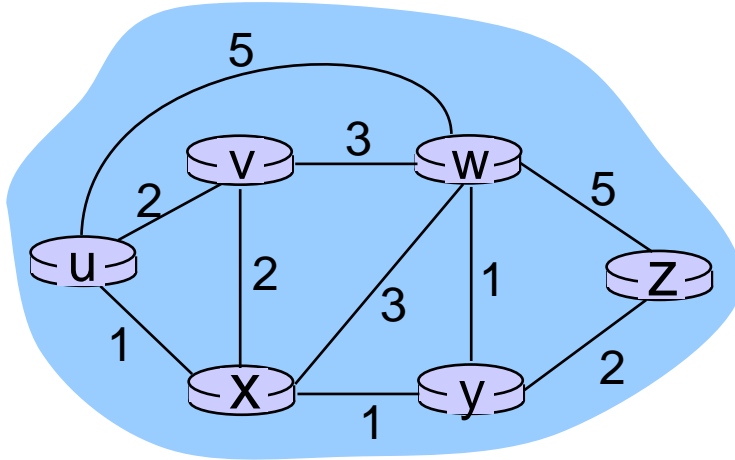

graph: G = (N,E)

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

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

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

Graph abstraction: costs

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

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

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

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

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

Dijsktra'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) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12      D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15  until all nodes in N'
```
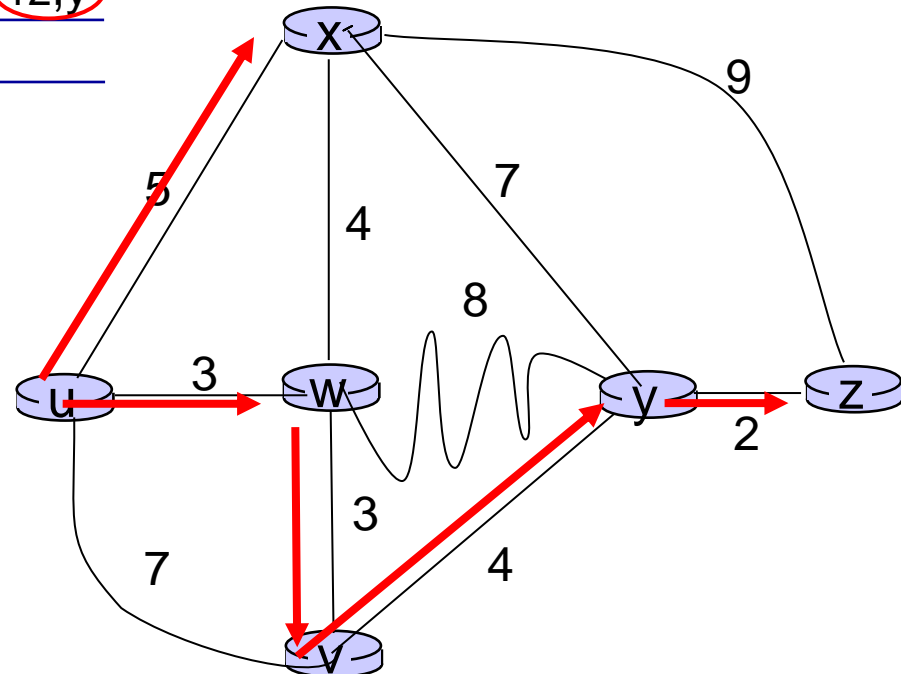
Dijkstra's algorithm: example

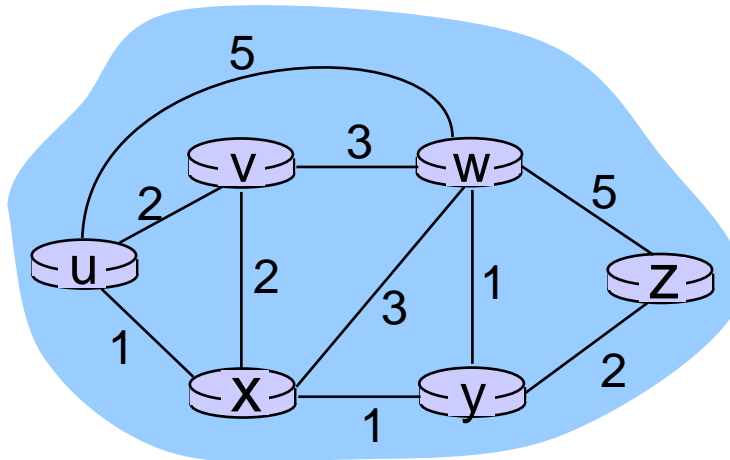| Step	N'	D(**v**) p(v)	D(**w**) p(w)	D(**x**) p(x)	D(**y**) p(y)	D(**z**) p(z)
0	u	7,u	⟨3,u⟩	5,u	∞	∞
1	uw	6,w		⟨5,u⟩	11,w	∞
2	uwx	⟨6,w⟩			11,w	14,x
3	uwxv				⟨10,v⟩	14,x
4	uwxvy					⟨12,y⟩
5	uwxvyz					

notes:

❖ construct shortest path tree by tracing predecessor nodes

❖ ties can exist (can be broken arbitrarily)

Dijkstra's algorithm: another example

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

Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:

resulting forwarding table in u:

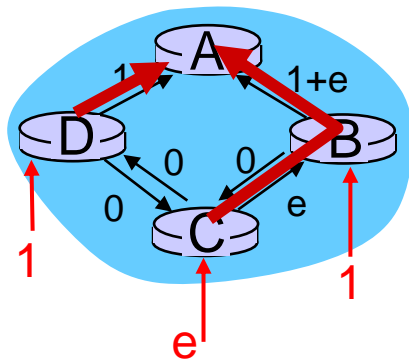| destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

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:

initially
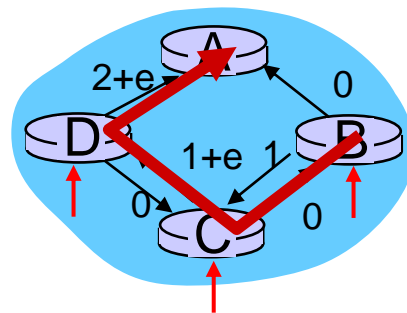
given these costs,
find new routing….
resulting in new costs
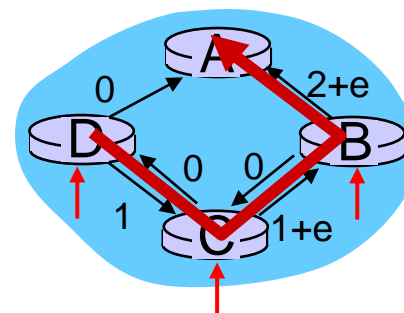
given these costs,
find new routing….
resulting in new costs
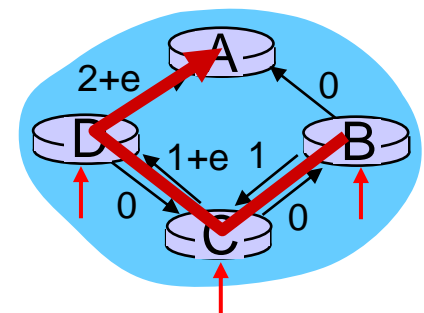
given these costs,
find new routing….
resulting in new costs

Chapter 5: outline

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

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

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example
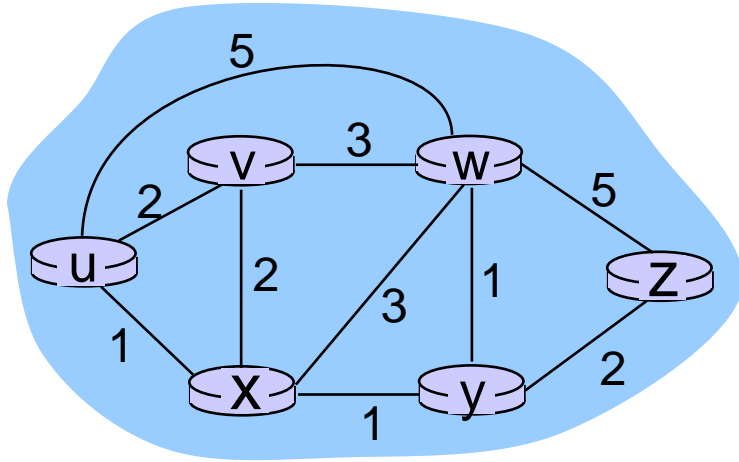
clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

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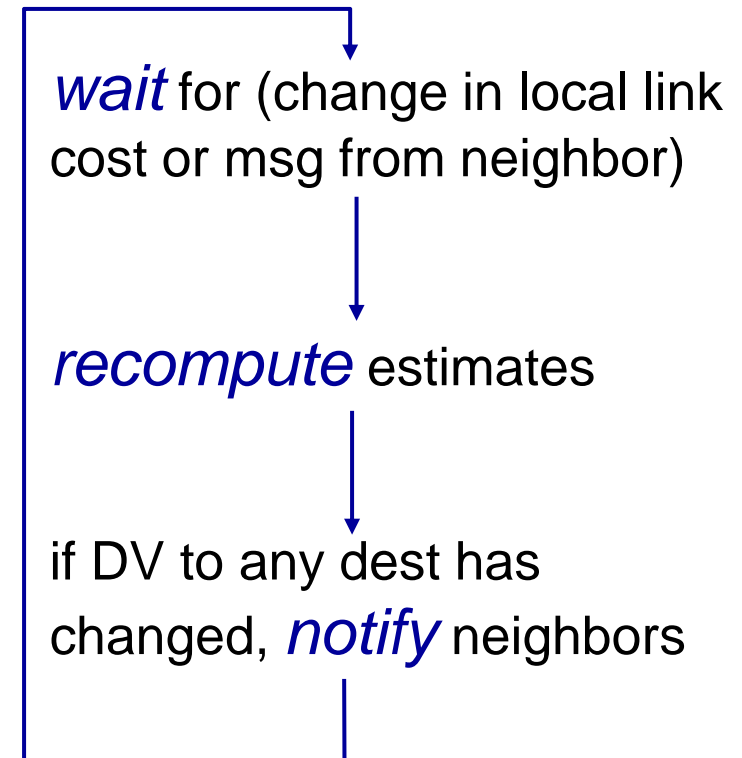
wait for (change in local link cost or msg from neighbor)

↓

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

cost to

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

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

time

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

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$
$$= \min\{2+1, 7+0\} = 3$$
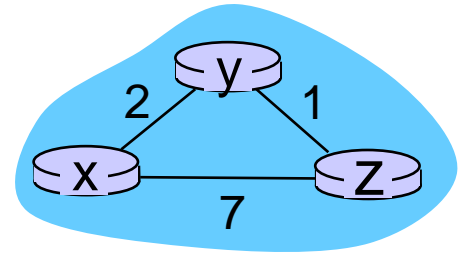
node x
table

cost to

	x	y	z
x	0	2	7
from y	∞	∞	∞
z	∞	∞	∞

cost to

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

cost to

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

node y
table

cost to

	x	y	z
x	∞	∞	∞
from y	2	0	1
z	∞	∞	∞

cost to

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

cost to

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

node z
table

cost to

	x	y	z
x	∞	∞	∞
from y	∞	∞	∞
z	7	1	0

cost to

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

cost to

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

time

Distance vector: link cost changes
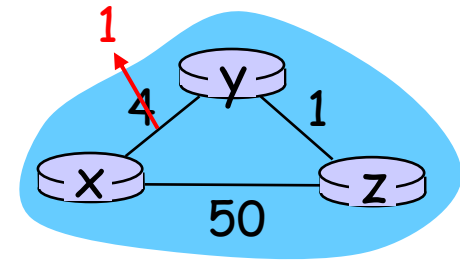
link cost changes:

❖ 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/

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
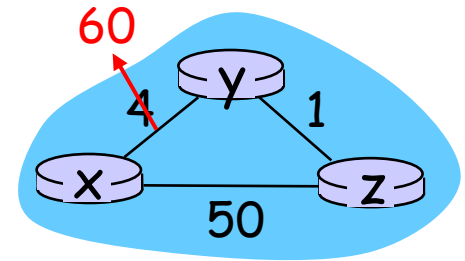
poisoned reverse:

- ❖ 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(n²) 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

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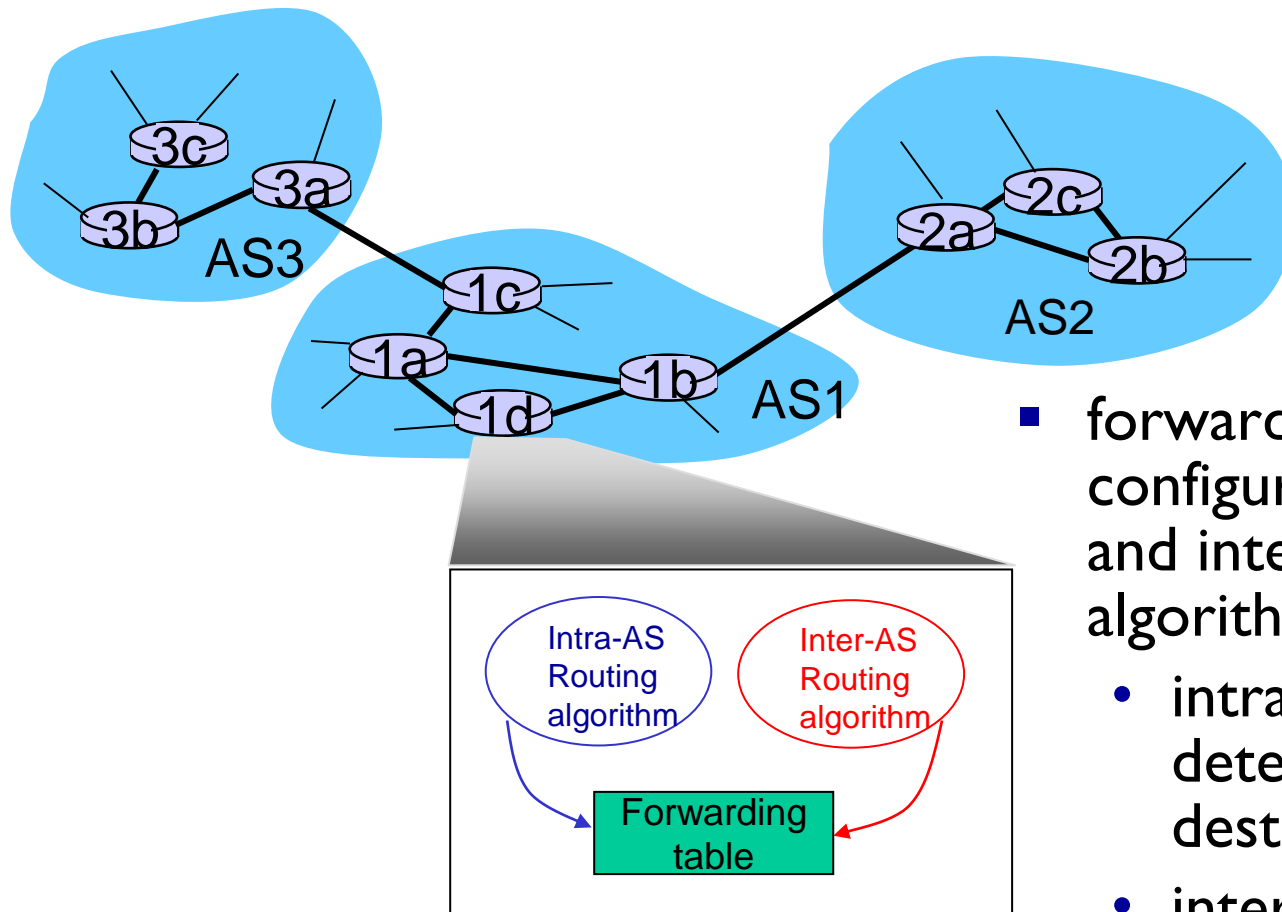
- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS routing determine entries for destinations within AS
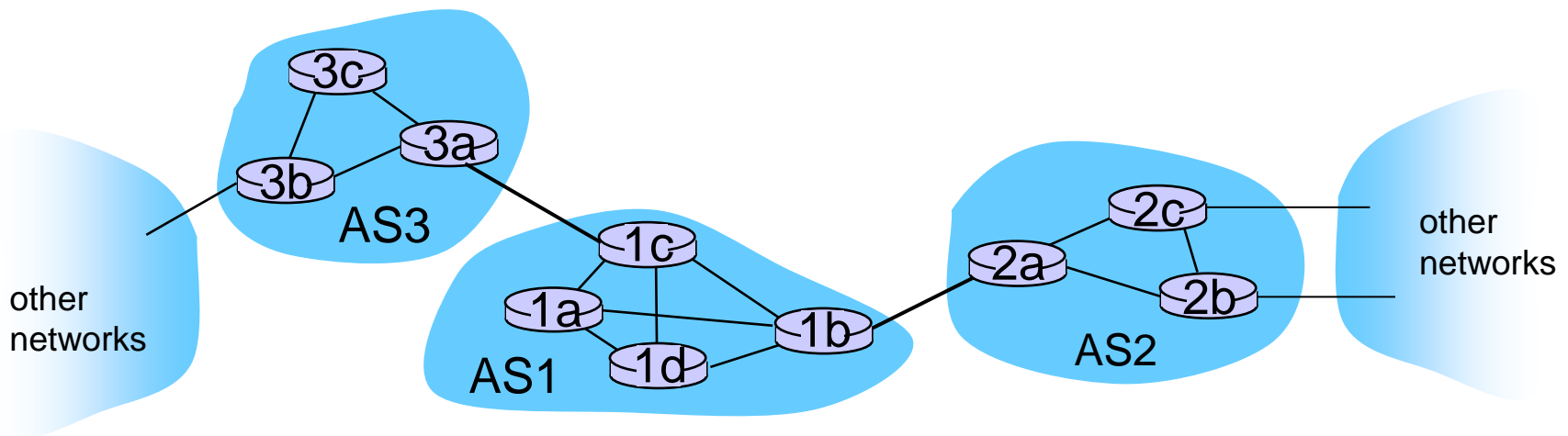 - inter-AS & intra-AS determine entries for external destinations

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)
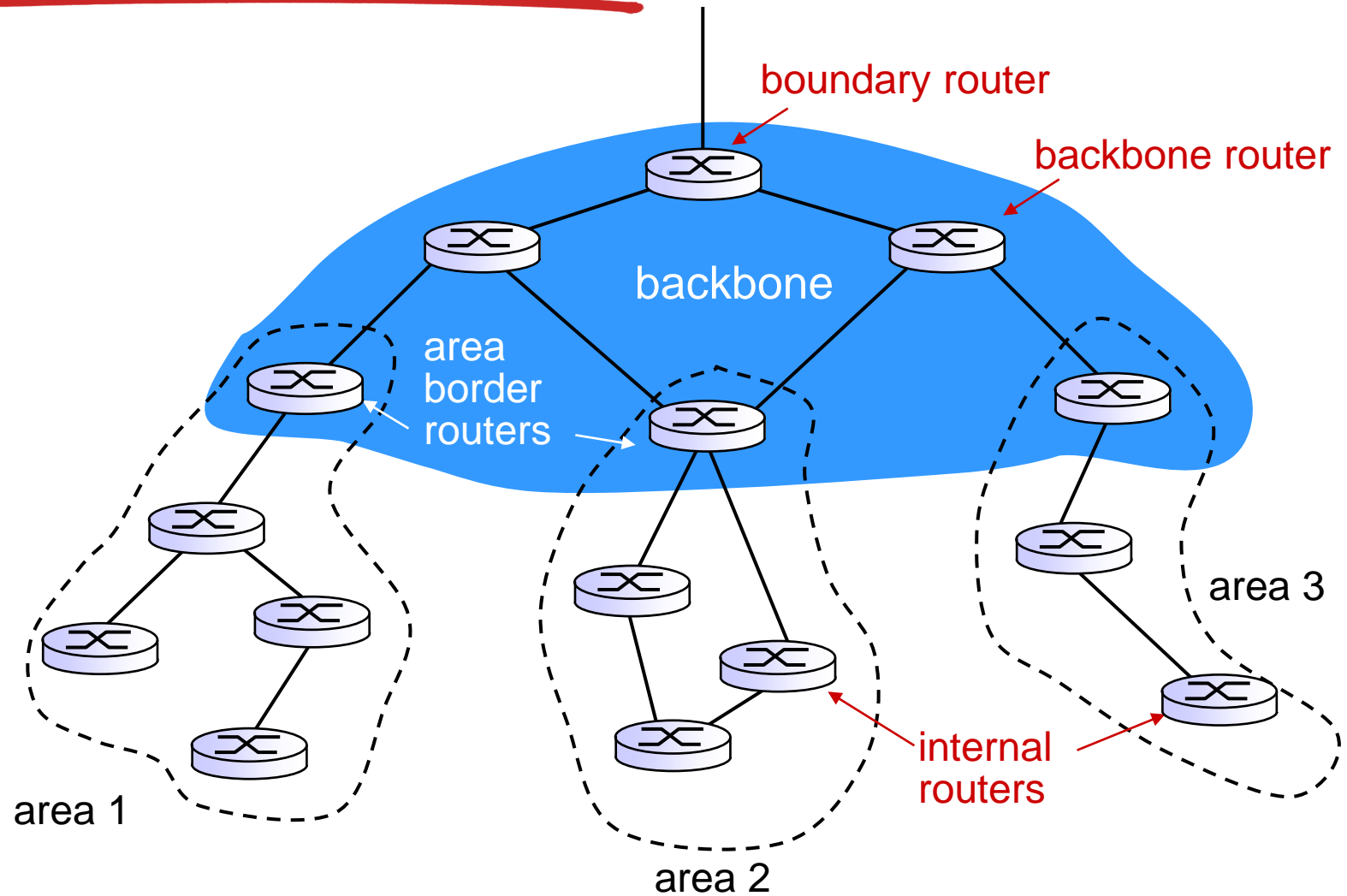
OSPF (Open Shortest Path First)

- "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

boundary router

backbone router

backbone

area border routers

internal routers

area 1

area 2

area 3

Hierarchical OSPF

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

Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state

- distance vector

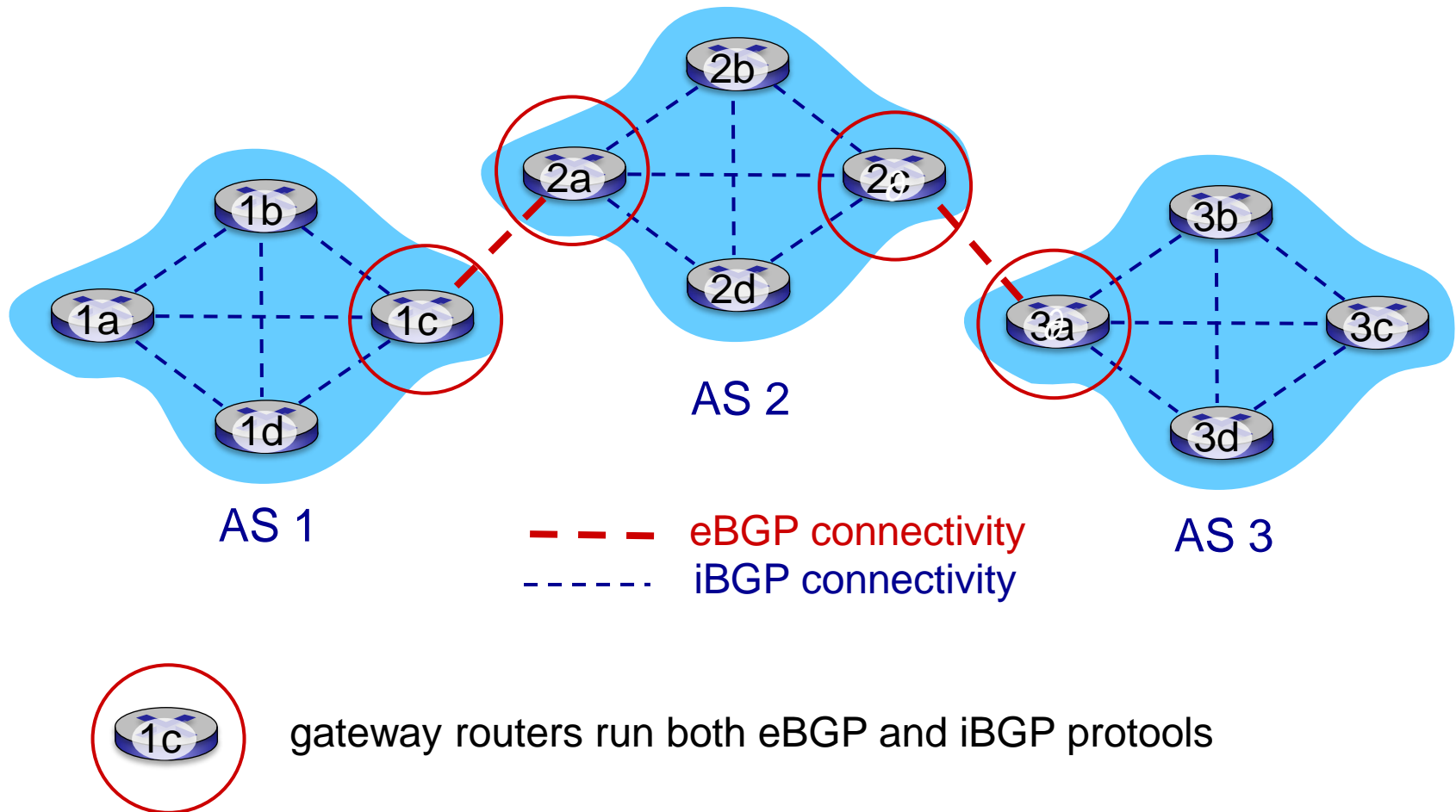5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

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

AS 1

AS 2

AS 3

- - - - eBGP connectivity

- - - - - - iBGP connectivity

gateway routers run both eBGP and iBGP protools

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

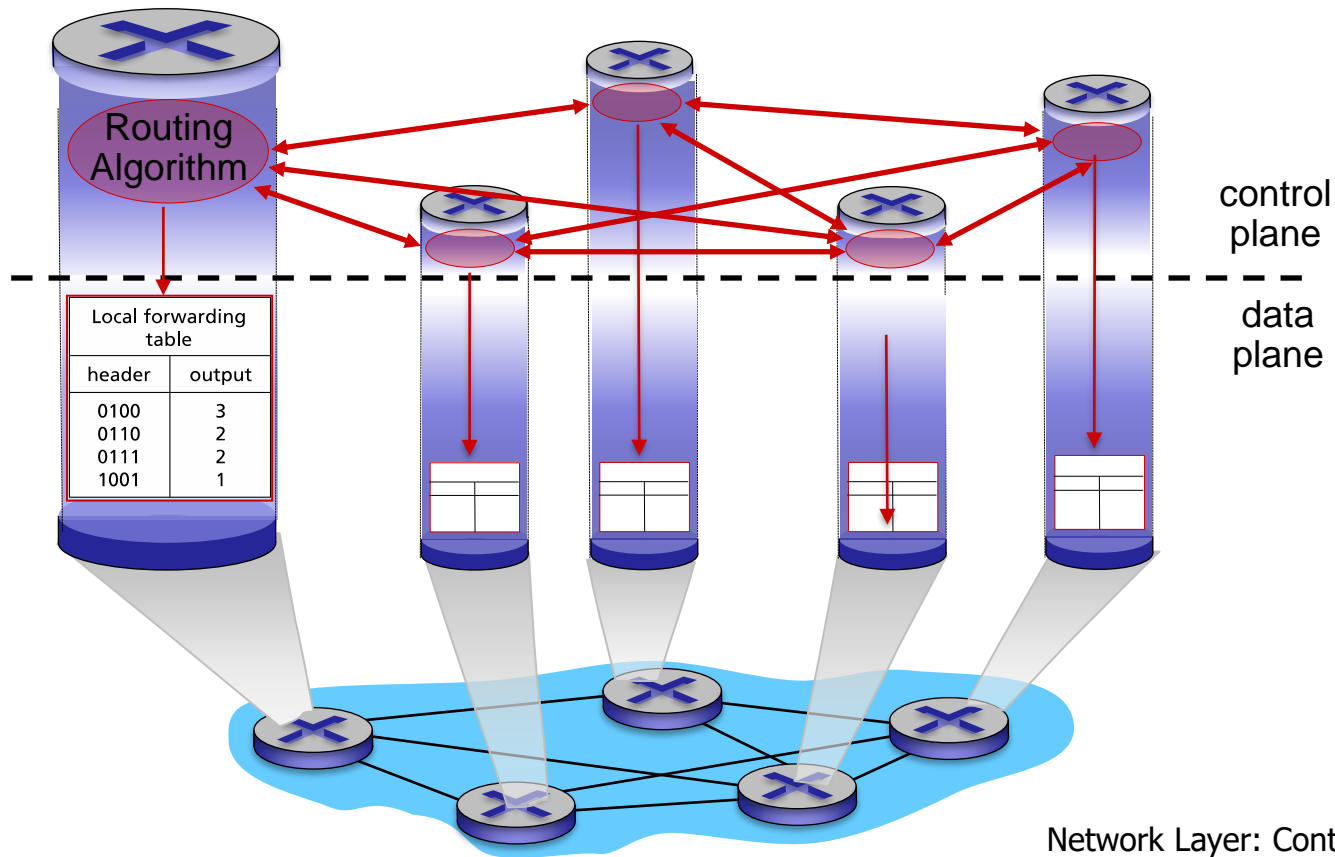# Chapter 5: outline

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

Routing Algorithm

Local forwarding table	
header	output
0100	3
0110	2
0111	2
1001	1

control plane

data plane

Recall: logically centralized control plane

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

Remote Controller

control plane

data plane

CA

CA

CA
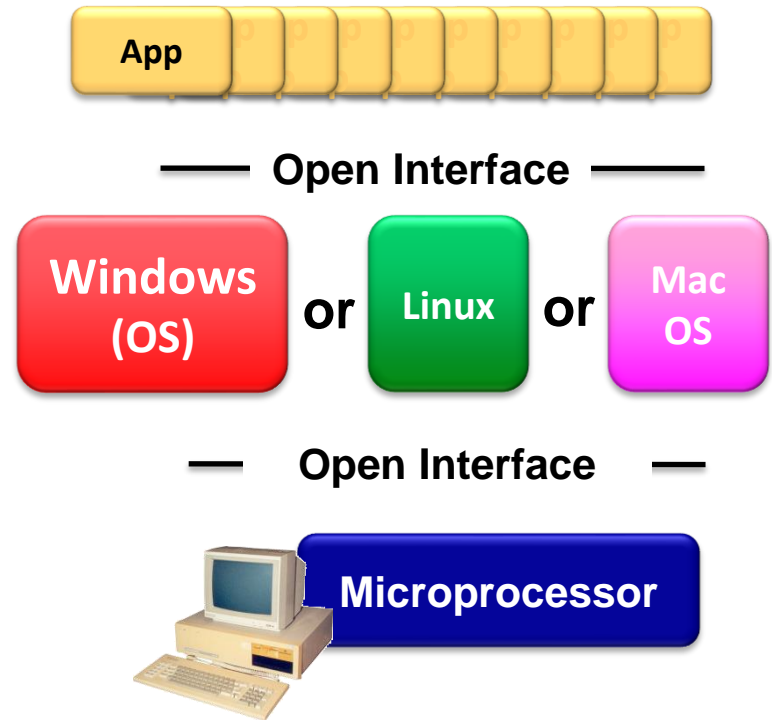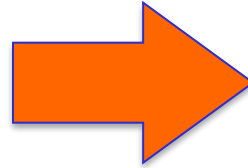
CA

CA
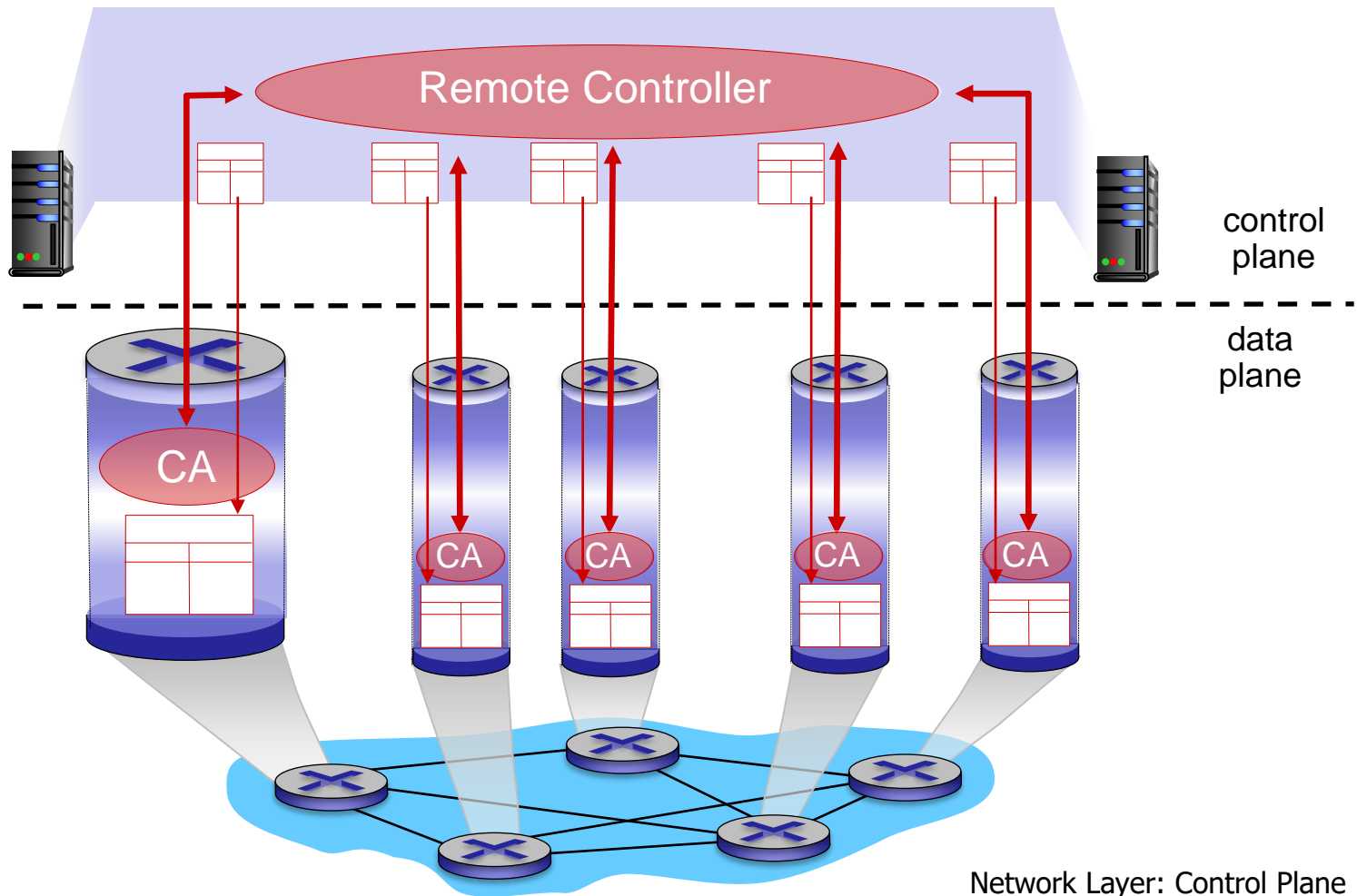
Software defined networking (SDN)

Why a *logically centralized* control plane?

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

- table-based forwarding 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*

Specialized Applications

Specialized Operating System

Specialized Hardware

Vertically integrated
Closed, proprietary
Slow innovation
Small industry

App

—— **Open Interface** ——

Windows (OS) or **Linux** or **Mac OS**

—— **Open Interface** ——

Microprocessor

Horizontal
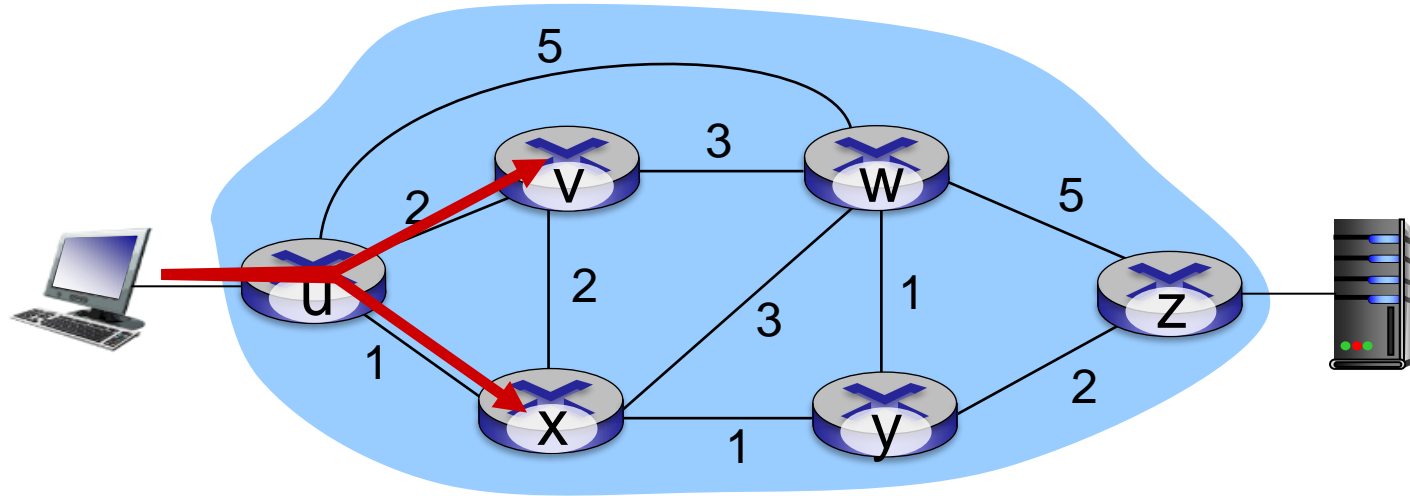Open interfaces
Rapid innovation
Huge industry

Traffic engineering: difficult traditional routing
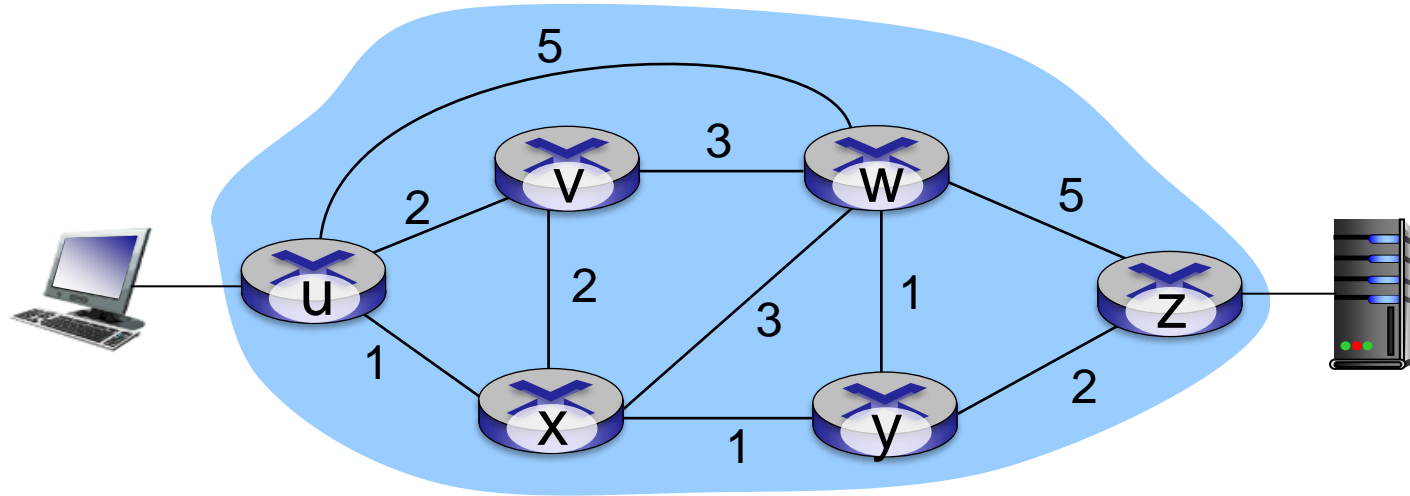
Q: what if network operator wants u-to-z traffic to flow along *uvw*z, x-to-z traffic to flow *xwyz*?

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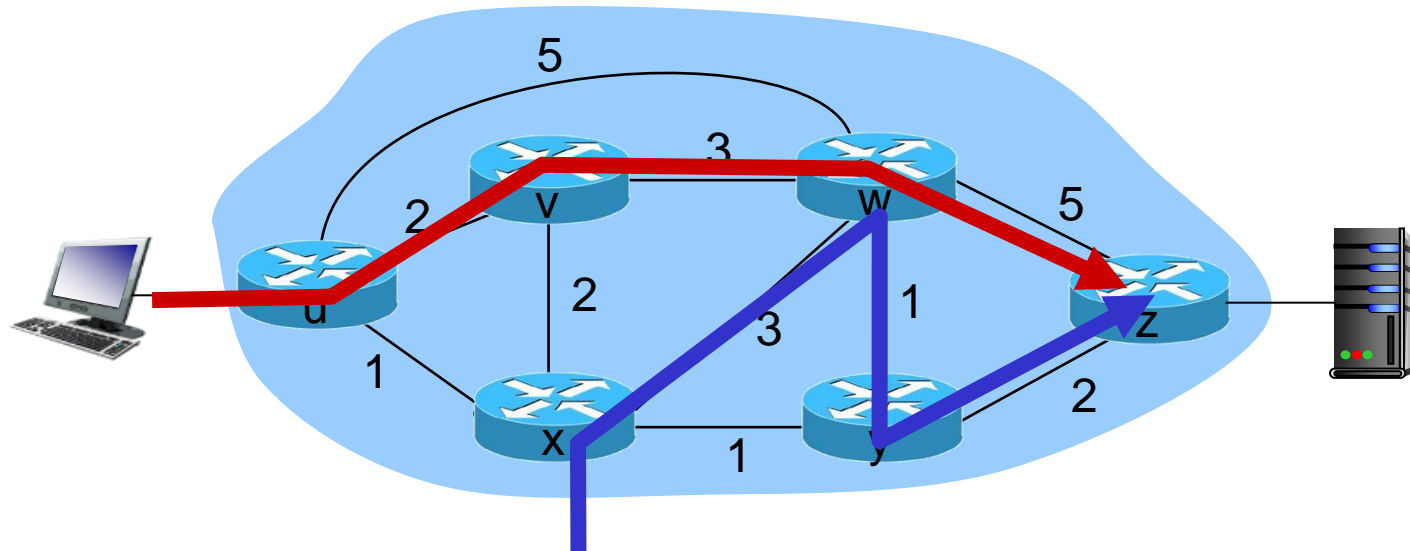
Traffic engineering: difficult

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*
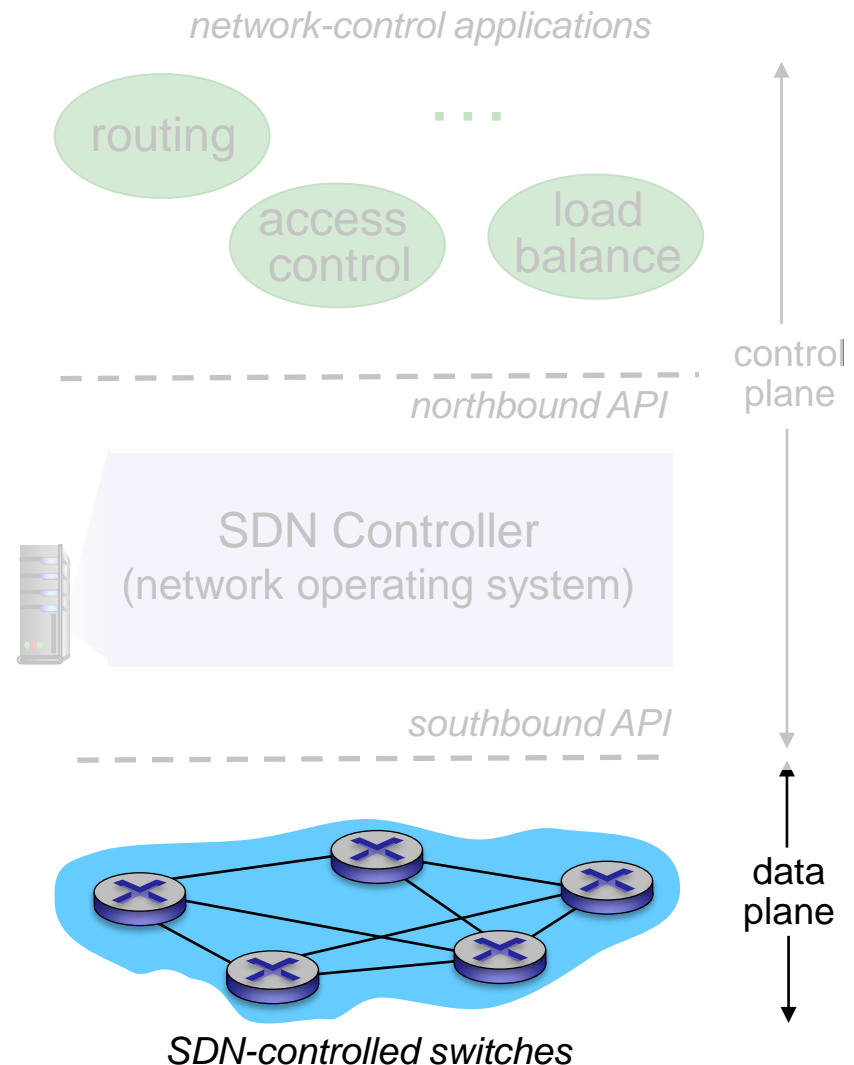
Remote Controller

control plane

- -

data plane

CA

2. *control, data plane separation*

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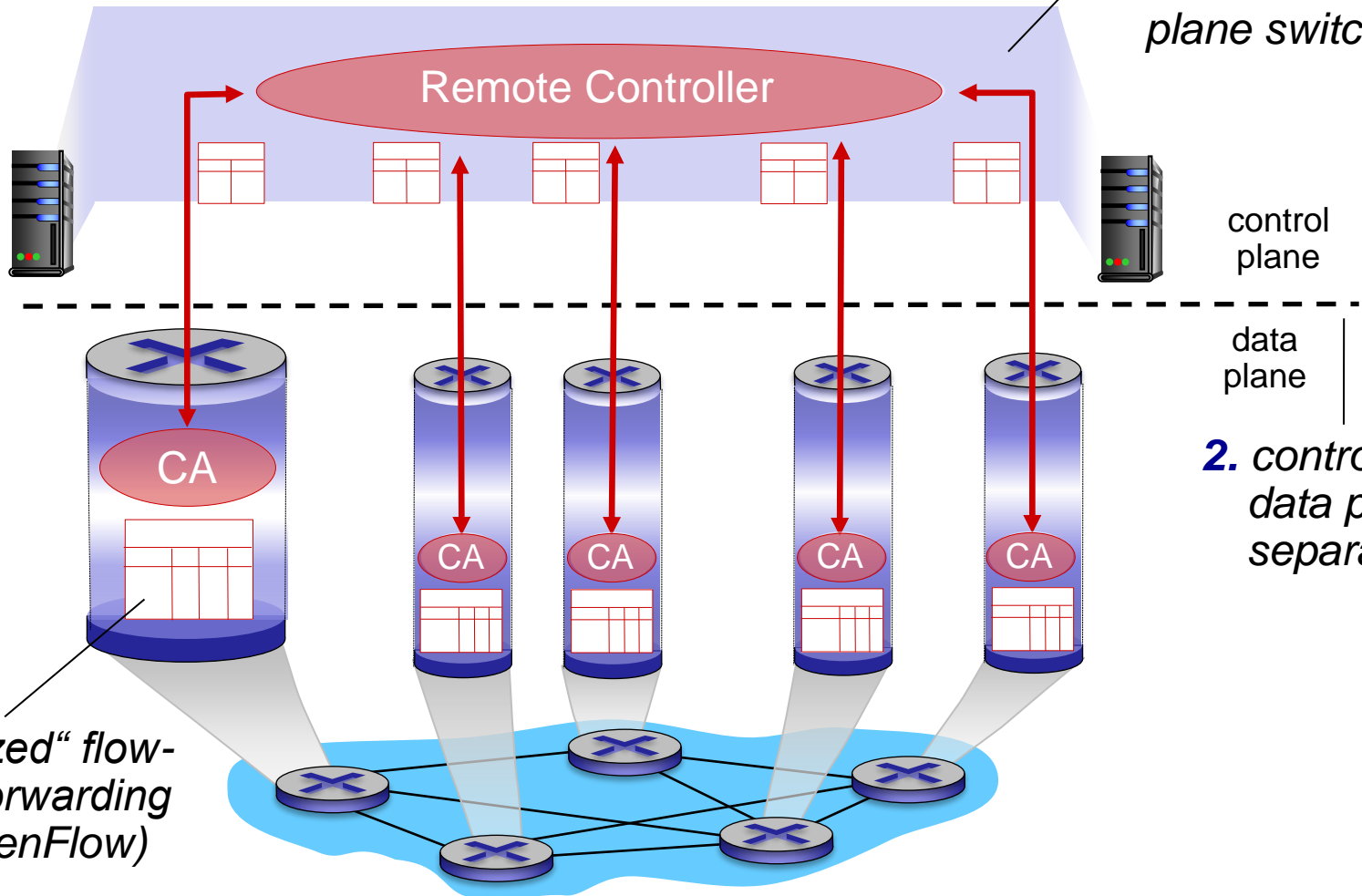
1: *generalized" flow-based" forwarding (e.g., OpenFlow)*

SDN perspective: data plane switches

Data plane switches

- fast, simple, commodity switches implementing generalized data-plane forwarding 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)

network-control applications

routing

· · ·

access control

load balance

control plane

northbound API

SDN Controller
(network operating system)

southbound API

data plane

SDN-controlled switches

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

network-control applications

routing

. . .

access control

load balance

control plane

northbound API

SDN Controller
(network operating system)

southbound API

data plane

SDN-controlled switches
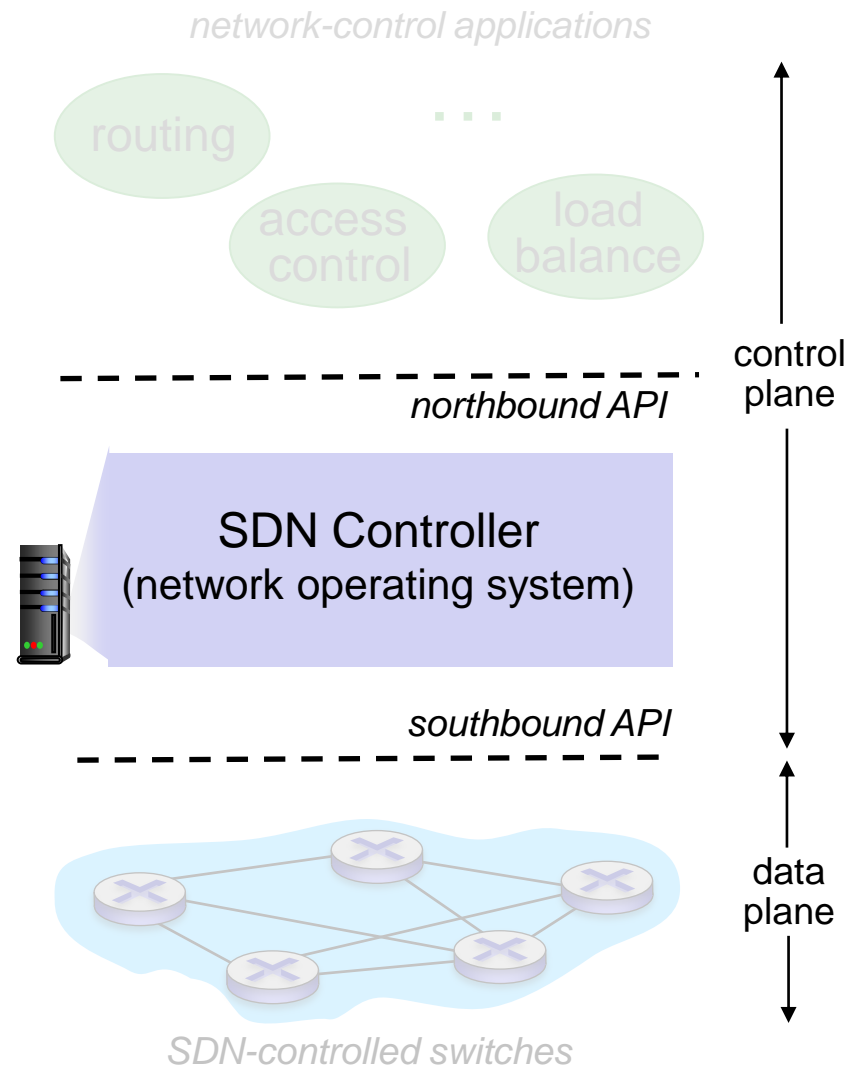
SDN perspective: control applications

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
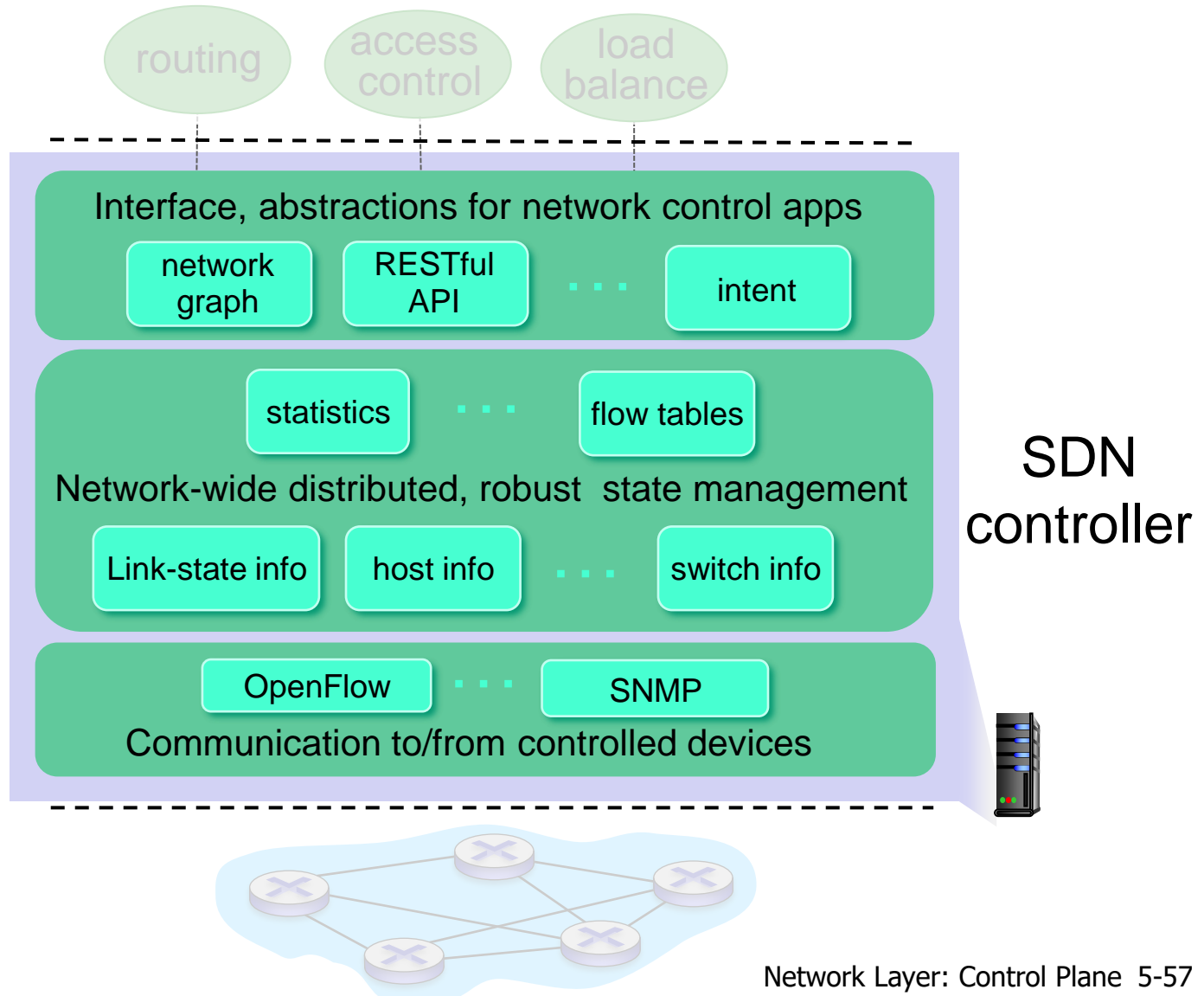
network-control applications

routing   ...   access control   load balance

northbound API

SDN Controller (network operating system)

southbound API

control plane

data plane

SDN-controlled switches
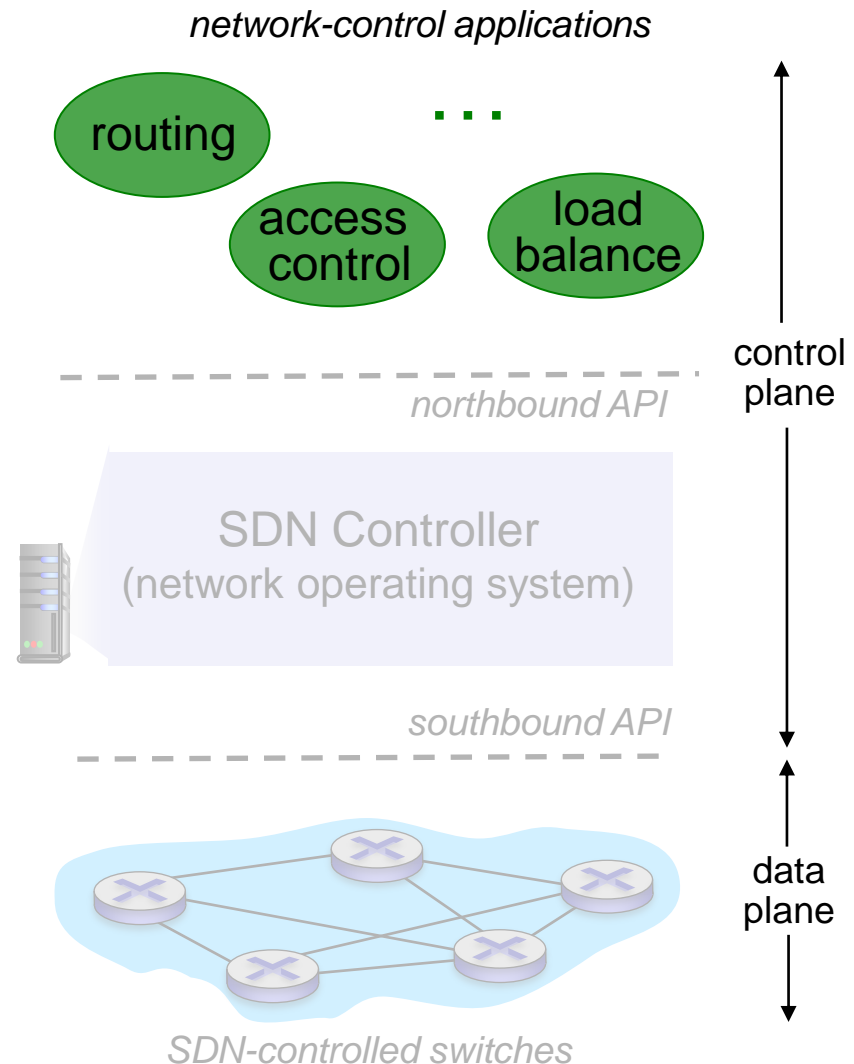
Components of SDN controller

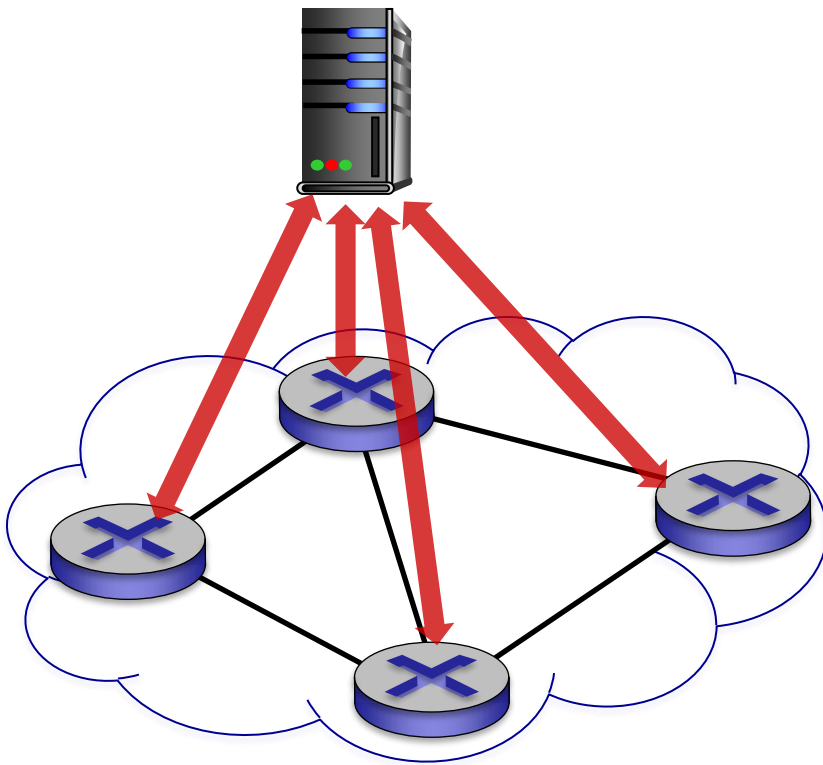**Interface layer to network control apps:** abstractions API

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

communication layer: communicate between SDN controller and controlled switches

routing   access control   load balance

Interface, abstractions for network control apps

| network graph | RESTful API | . . . | intent |

Network-wide distributed, robust state management

| statistics | . . . | flow tables |

| Link-state info | host info | . . . | switch info |

Communication to/from controlled devices

| OpenFlow | . . . | SNMP |

SDN controller

OpenFlow protocol

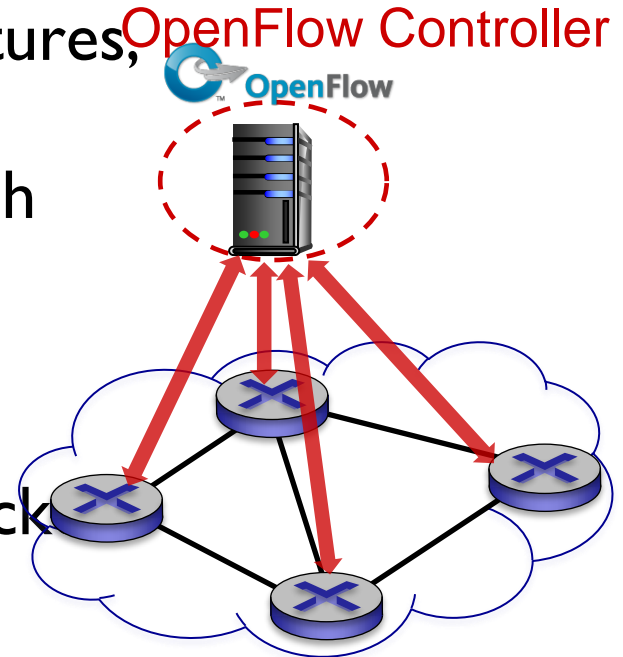OpenFlow Controller

OpenFlow

- 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

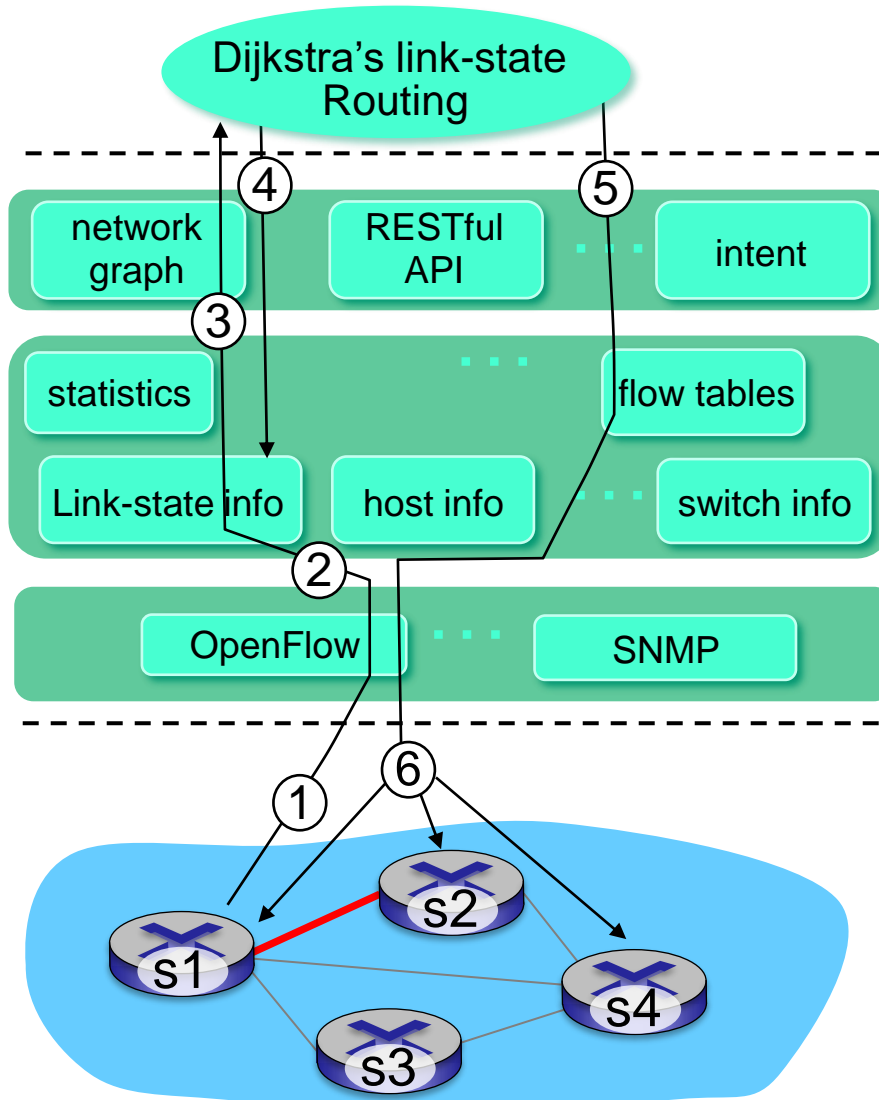OpenFlow Controller

OpenFlow: switch-to-controller messages

Key switch-to-controller messages

- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller

- *flow-removed:* flow table entry deleted at switch

- *port status:* inform controller of a change on a port.
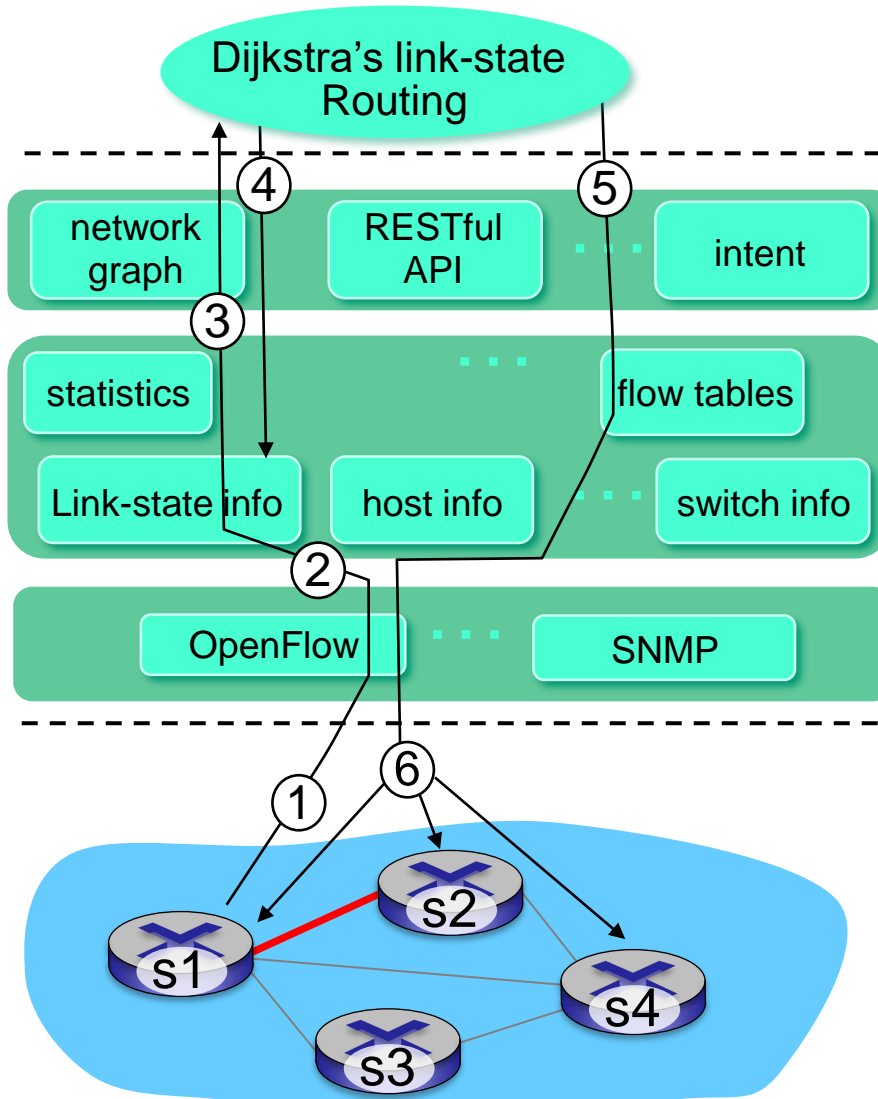
OpenFlow Controller
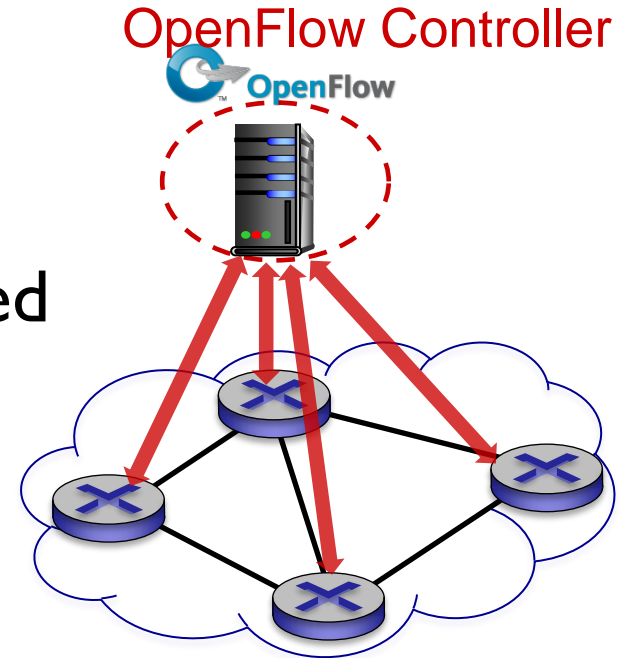
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller
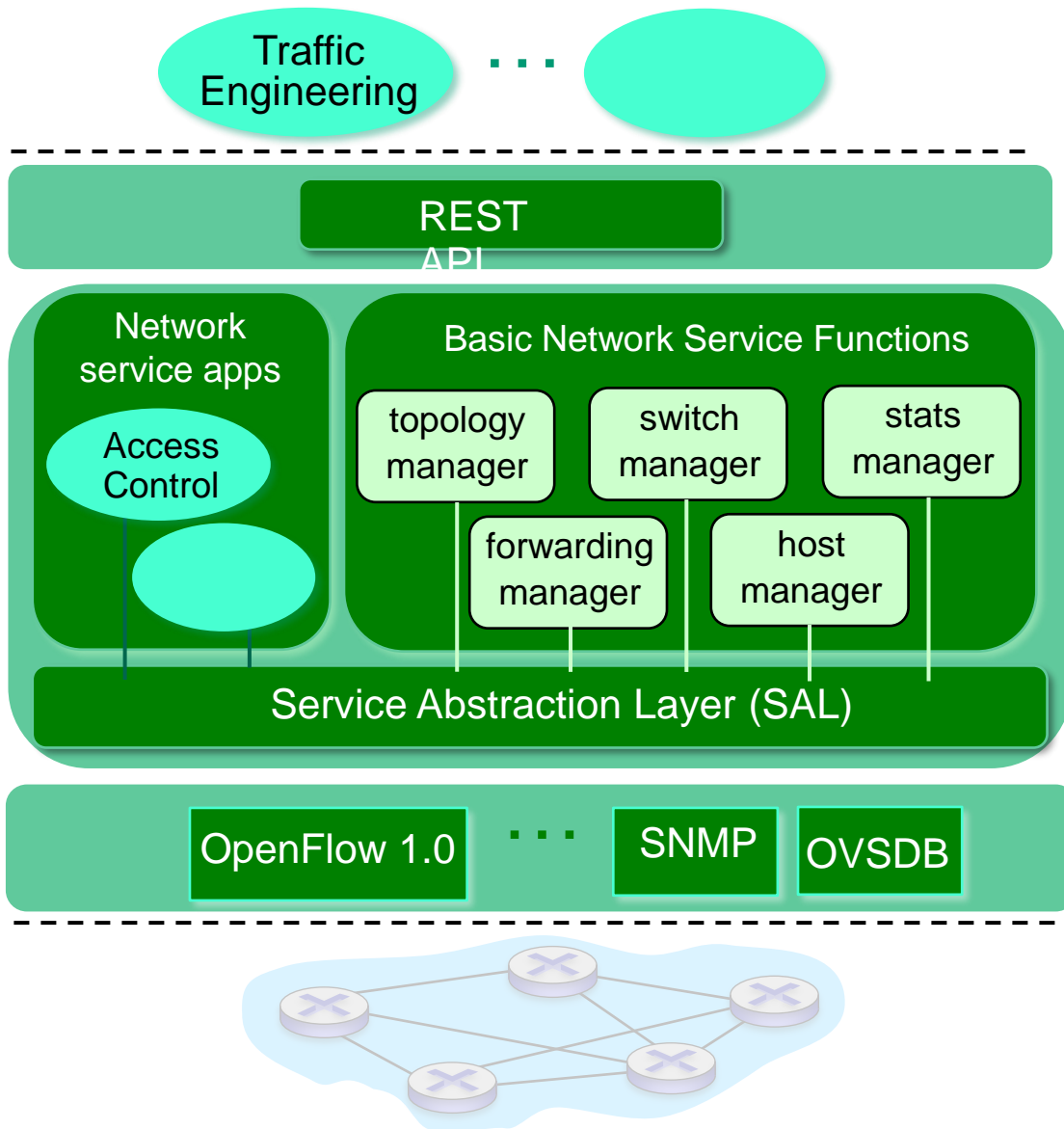
SDN: control/data plane interaction example

Dijkstra's link-state Routing

network graph   RESTful API   intent

statistics   flow tables

Link-state info   host info   switch info

OpenFlow   SNMP

s1   s2   s3   s4

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

Traffic Engineering   . . .

REST API

Network service apps

Access Control

Basic Network Service Functions

topology manager

switch manager

stats manager

forwarding manager

host manager

Service Abstraction Layer (SAL)

OpenFlow 1.0   . . .   SNMP   OVSDB

- 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

Network control apps

REST API   Intent

northbound abstractions, protocols

hosts   paths   flow rules   topology

devices   links   statistics

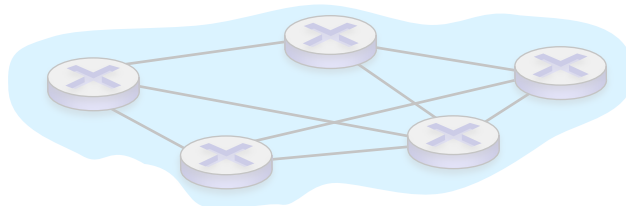ONOS distributed core

device   link   host   flow   packet

OpenFlow   Netconf   OVSDB

southbound abstractions, protocols

- **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