Network Layer: Control Plane

Goal: understand principles behind network control plane

- traditional (intra-domain) routing algorithms
- SDN controllers

and their instantiation, implementation in the Internet:

- OSPF, RIP, OpenFlow, ODL and ONOS controllers

The following will be discussed in separate lecture notes

- inter-domain routing & BGP
- Internet Control Message Protocol: ICMP
- network management and SNMP

Readings: Textbook: Chapter 5, Sections 5.1-5.3 & 5.5

Network Layer Functions

Recall: two network-layer functions:

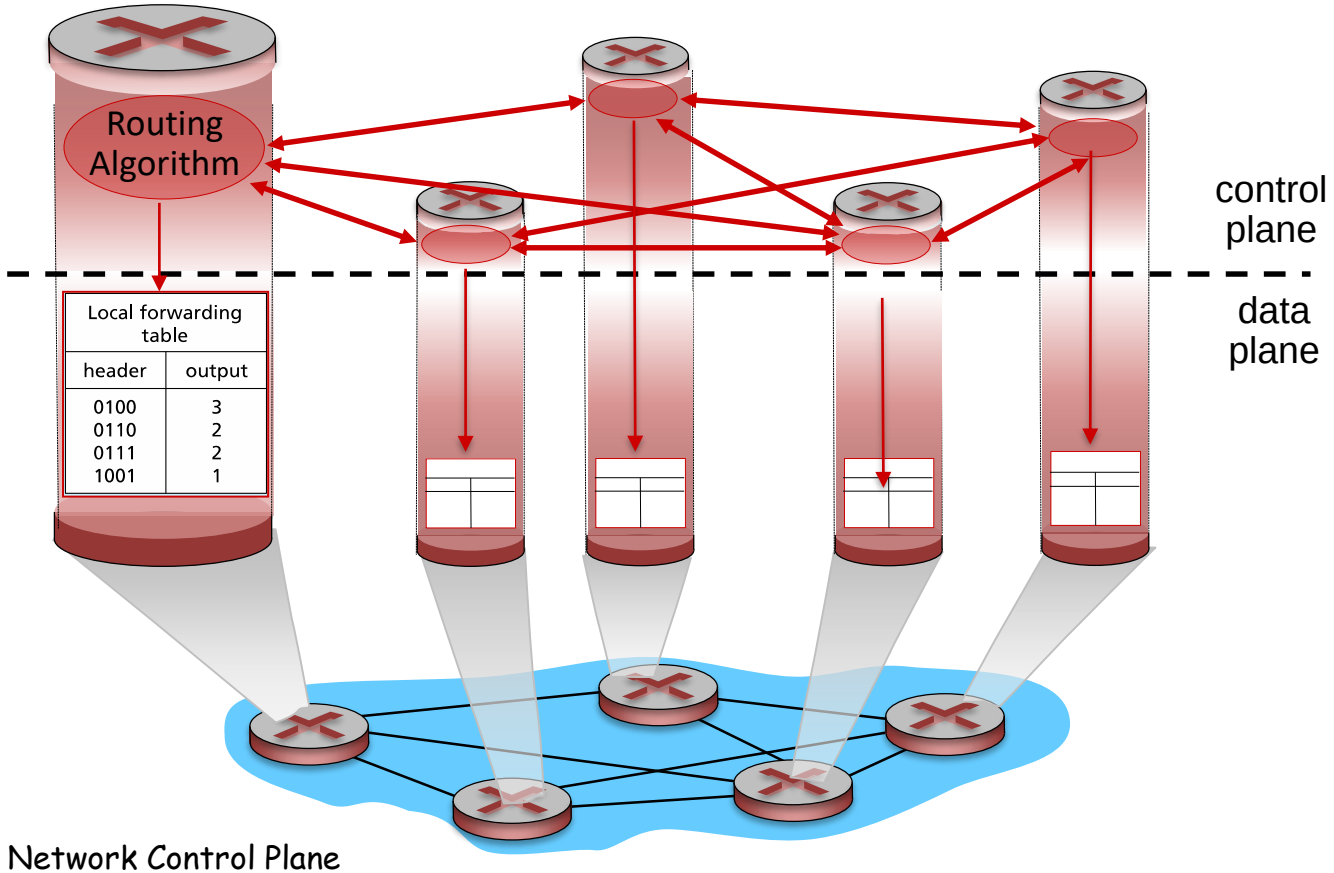
- *forwarding:* move packets from router's input to appropriate router output — *data plane*
- *routing:* determine route taken by packets from source to destination — *control plane*

Two approaches to structuring network control plane:

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

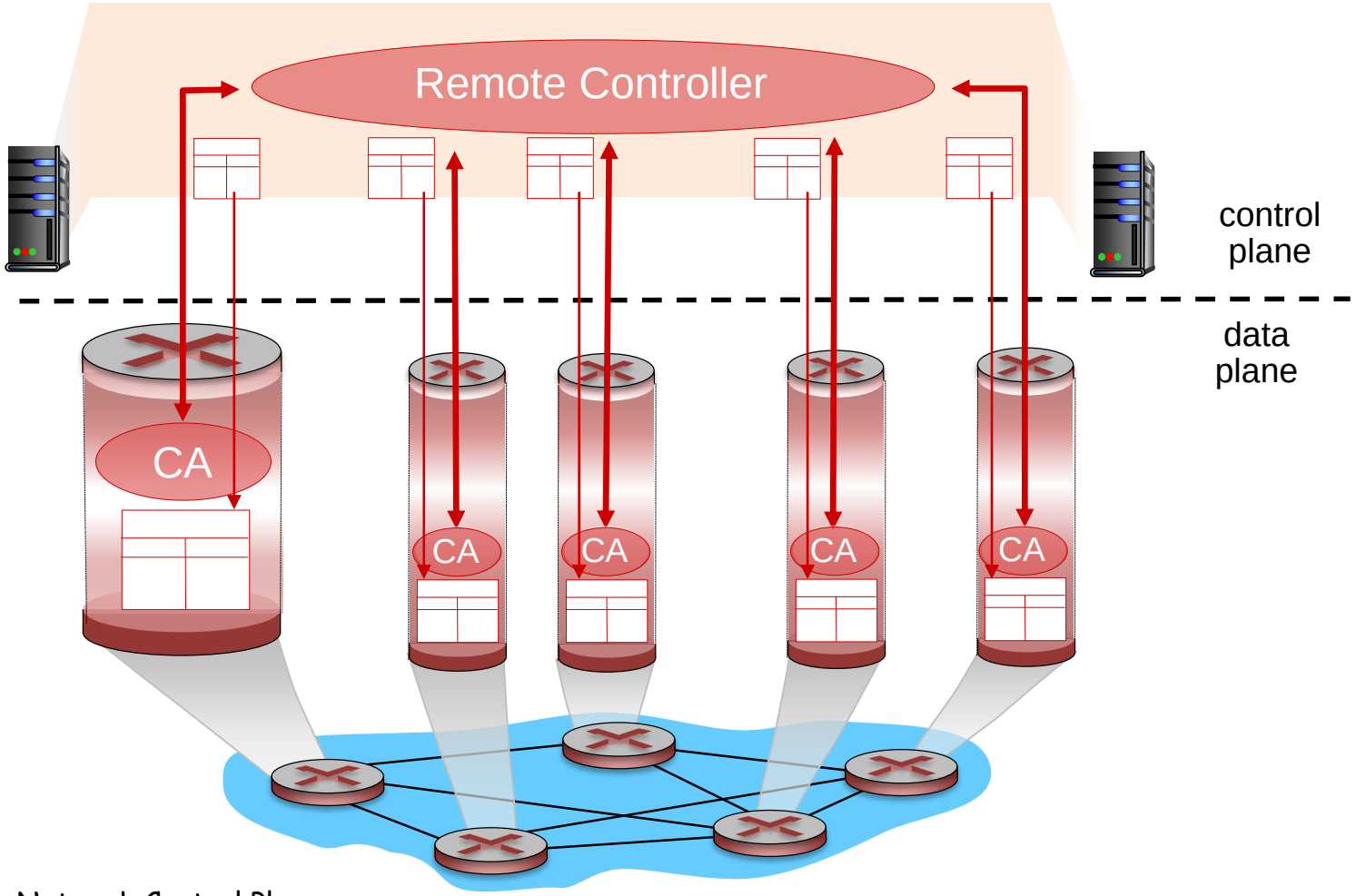
Per-router Distributed Control Plane

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

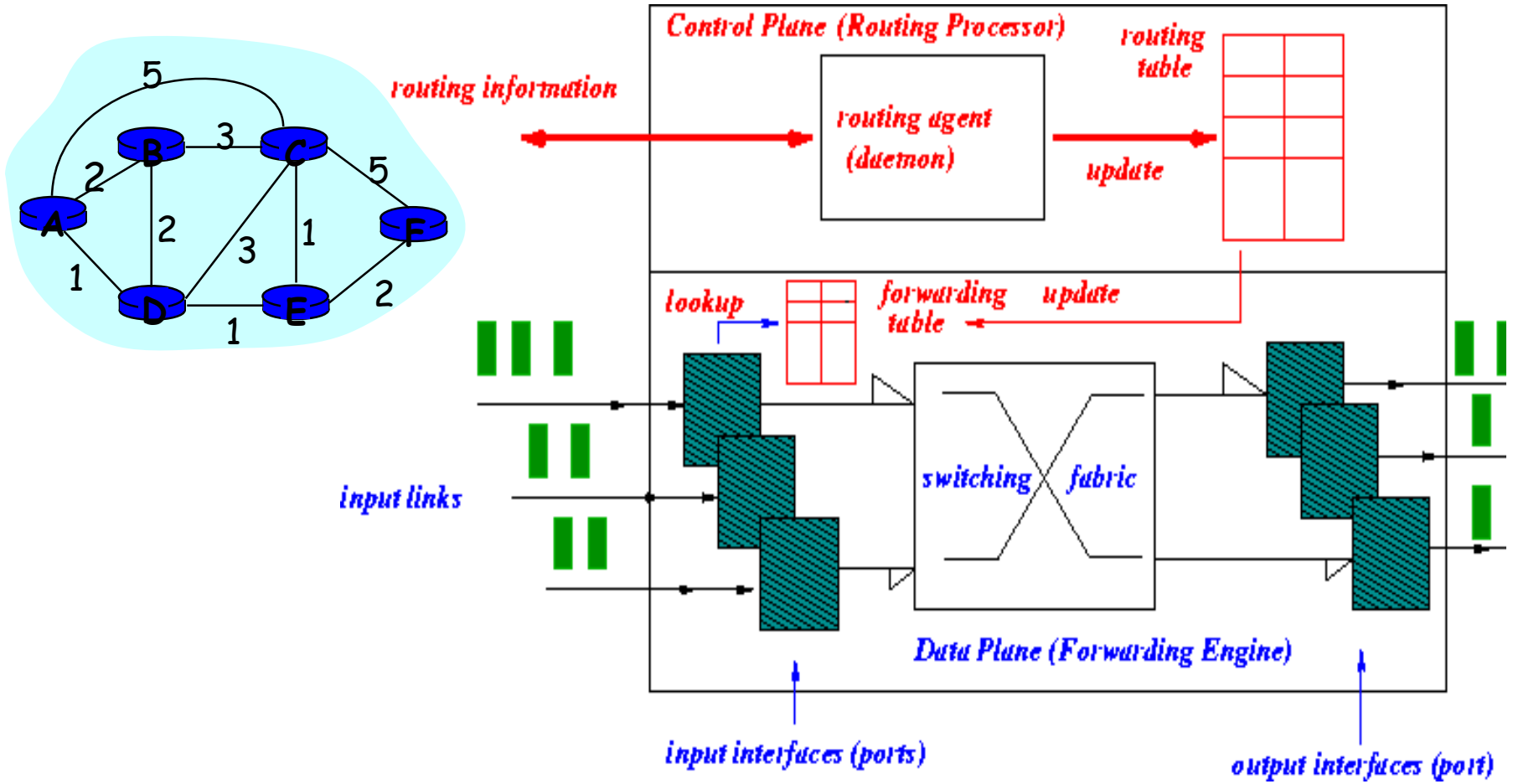


Logically Centralized Control Plane

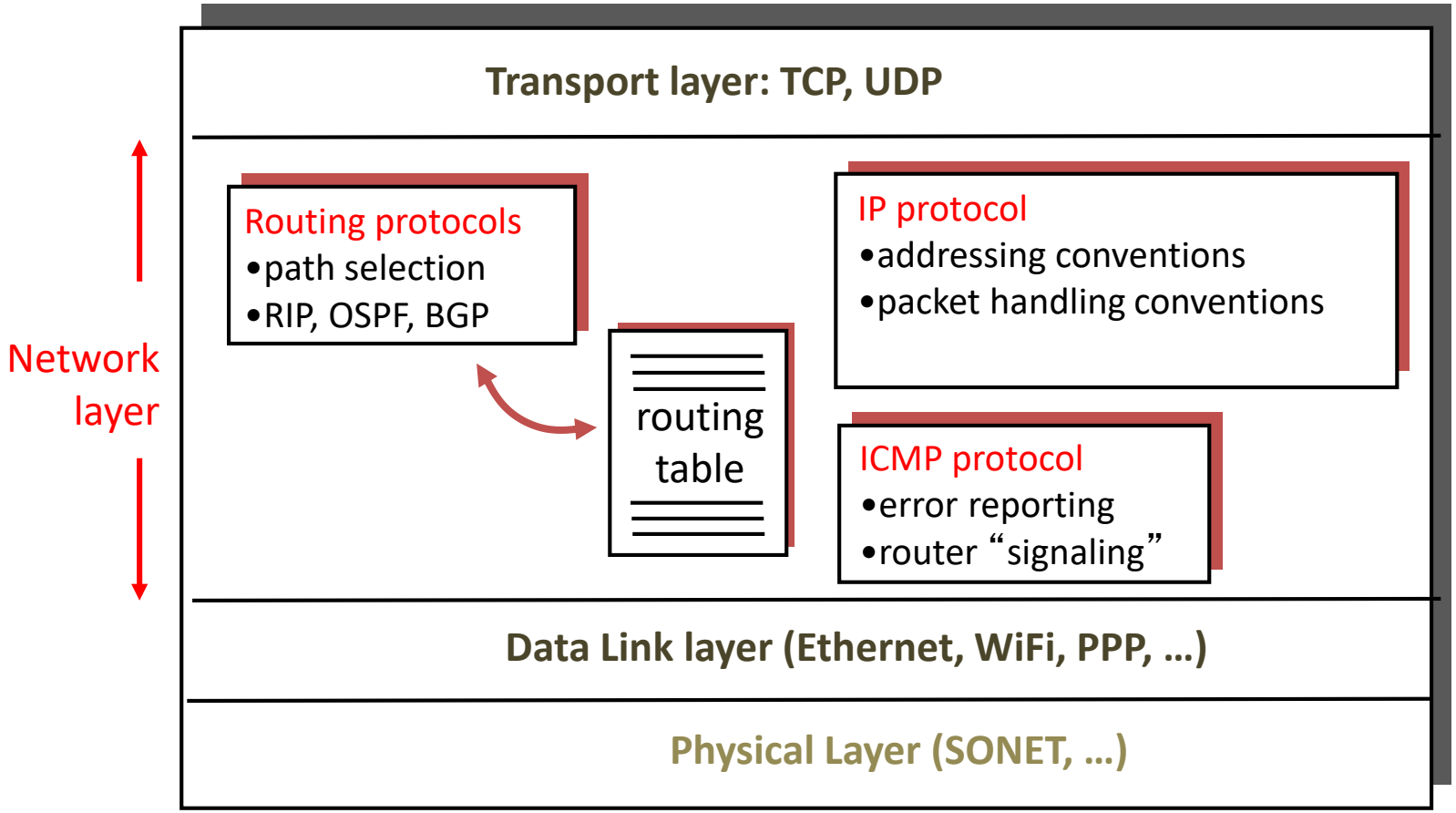
A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



Routing & Forwarding:
Logical View of a (Classical) Router



IP Forwarding & IP/ICMP Protocol



Routing: Issues

- How are routing tables determined?
- Who determines table entries?
- What info used in determining table entries?
- When do routing table entries change?
- Where is routing info stored?
- How to control routing table size?

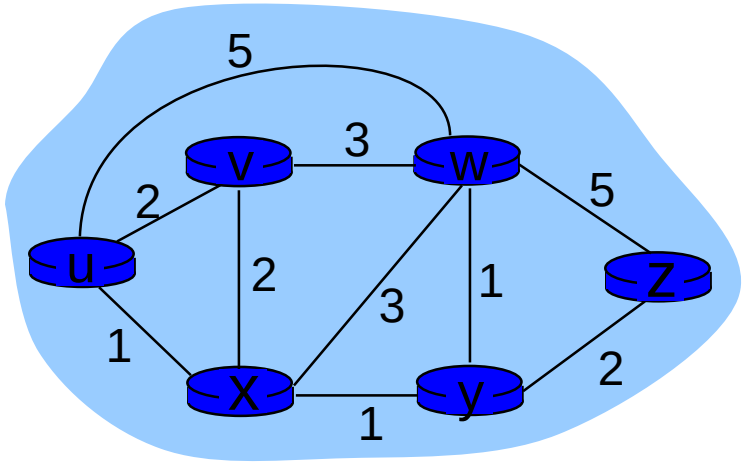
Answer these questions, we are done!

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



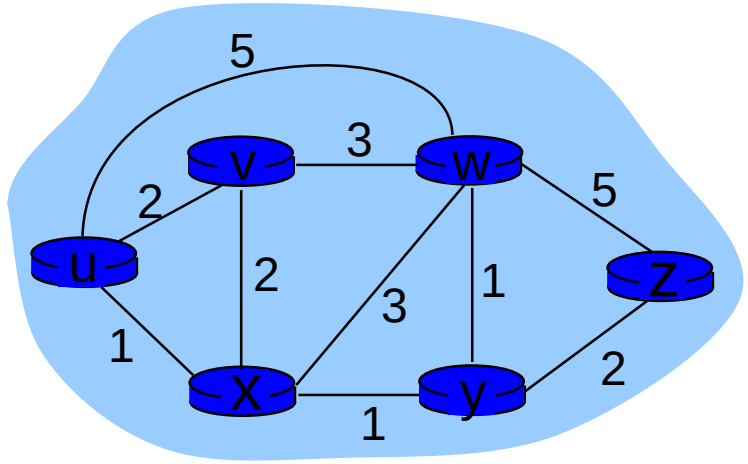
graph: G = (N,E)

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 $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

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

Routing Algorithms/Protocols

Issues Need to Be Addressed:

- Route selection may depend on different criteria
  - Performance: choose route with smallest delay
  - Policy: choose a route that doesn't cross .gov network
- Adapt to changes in network topology or condition
  - Self-healing: little or no human intervention
- Scalability
  - Must be able to support large number of hosts, routers

Classical Distributed Routing Paradigms

- Hop-by-hop Routing
  - Each packet contains destination address
  - Each router chooses next-hop to destination
    - routing decision made at each (intermediate) hop!
    - packets to same destination may take different paths!
  - Example: IP's default datagram routing
- Source Routing
  - Sender selects the path to destination precisely
  - Routers forward packet to next-hop as specified
    - Problem: if specified path no longer valid due to link failure!
  - Example:
    - IP's loose/strict source route option (you'll see later)
    - virtual circuit setup phase (or MPLS)

Centralized vs. Distributed Routing Algorithms

Centralized:

- A centralized route server collects routing information and network topology, makes route selection decisions, then distributes them to routers

Distributed:

- Routers cooperate using a distributed protocol
  - to create mutually consistent routing tables
- Two standard distributed routing algorithms
  - Link State (LS) routing
  - Distance Vector (DV) routing

Link State *vs.* Distance Vector

- Both assume that
  - The address of each neighbor is known
  - The cost of reaching each neighbor is known
- Both find global information
  - By exchanging routing info among neighbors
- Differ in info exchanged and route computation
  - LS: tells every other node its distance to neighbors
  - DV: tells neighbors its distance to every other node

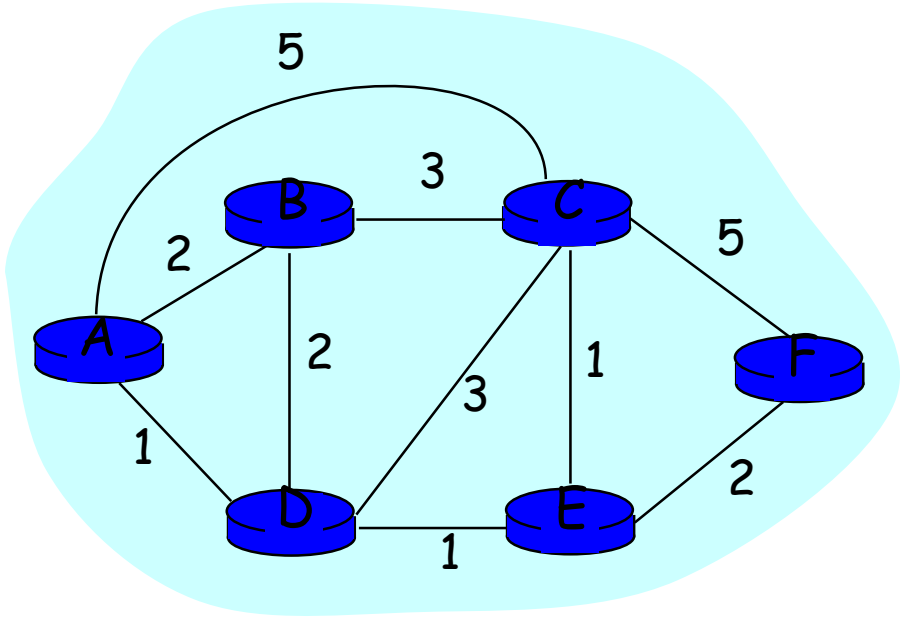
Link State Algorithm

- Basic idea: Distribute to all routers
  - Topology of the network
    - Cost of each link in the network
- Each router independently computes optimal paths
  - From itself to every destination
  - Routes are guaranteed to be loop free if
    - Each router sees the same cost for each link
    - Uses the same algorithm to compute the best path

Topology Dissemination

- Each router creates a set of link state packets (LSPs)
  - Describing its links to neighbors
  - LSP contains
    - Router id, neighbor's id, and cost to its neighbor
- Copies of LSPs are distributed to all routers
  - Using controlled flooding
- Each router maintains a topology database
  - Database containing all LSPs

Topology Database: Example



c(x,y)	A	B	C	D	E	F
A	0	2	5	1	∞	∞
B	2	0	3	2	∞	∞
C	5	3	0	3	1	5
D	1	2	3	0	1	∞
E	∞	∞	1	1	0	2
F	∞	∞	5	∞	2	0

link state database

Constructing Routing Table: Dijkstra's Algorithm

- Given the network topology
  - How to compute shortest path to each destination?
- Some notation
  - X: source node
  - N: set of nodes to which shortest paths are known so far
    - N is initially empty
  - D(V): cost of known shortest path from source X
  - C(U,V): cost of link U to V
    - C(U,V) = ∞ if not neighbors

Dijsktra's Algorithm (at Node X)

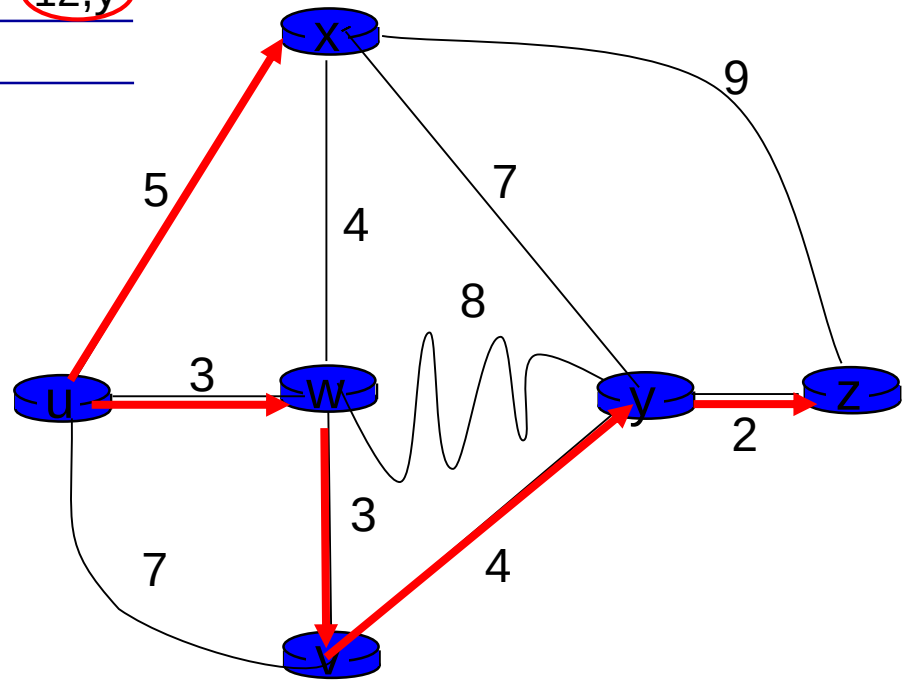
- Initialization
  - N = {X}
  - For all nodes V
    - If V adjacent to X, D(V) = C(X,V)
    - else D(V) = ¥
- Loop
  - Find U not in N such that D(U) is smallest
  - Add U into set N
  - Update D(V) for all V not in N
    - D(V) = min{D(V), D(U) + C(U,V)}
  - Until all nodes in N

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					

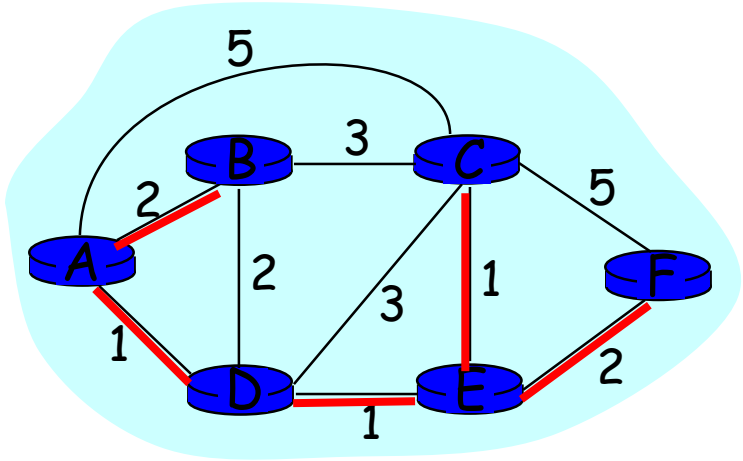
notes:

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



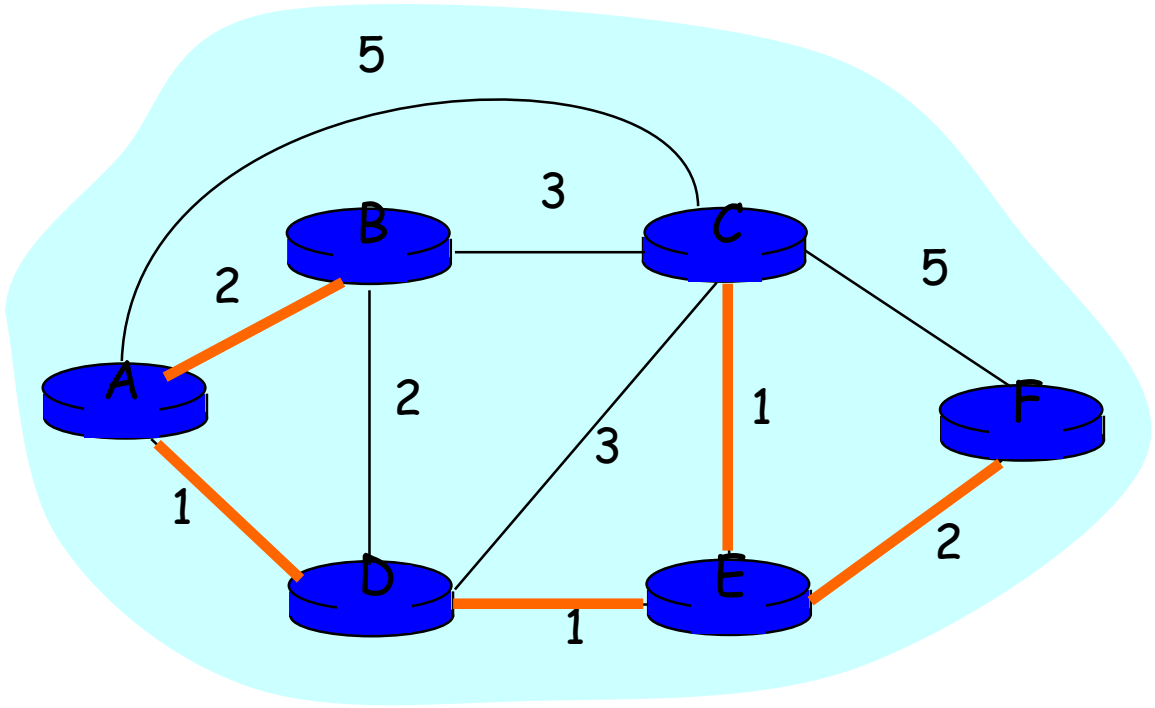
Dijkstra's Algorithm: Another Example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
→ 0	A	2,A	5,A	1,A	infinity	infinity
→ 1	AD	2,A	4,D		2,D	infinity
→ 2	ADE	2,A	3,E			4,E
→ 3	ADEB		3,E			4,E
→ 4	ADEBC					4,E
5	ADEBCF					



Routing Table Computation

dest	next
B	B
C	D
D	D
E	D
F	D



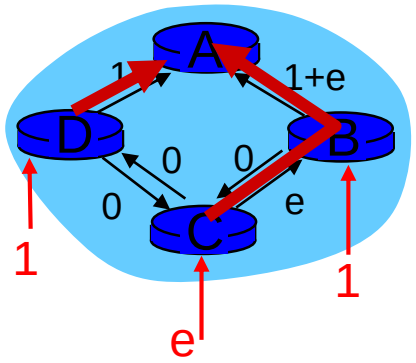
Dijkstra's Algorithm: Discussion

algorithm complexity: n nodes

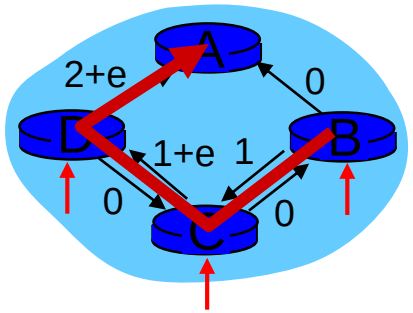
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

oscillations possible:

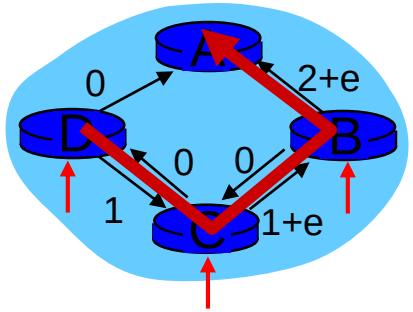
- e.g., support link cost equals amount of carried traffic:



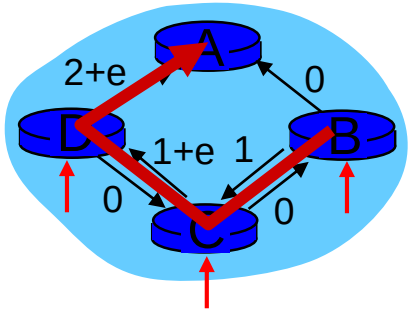
initially



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



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

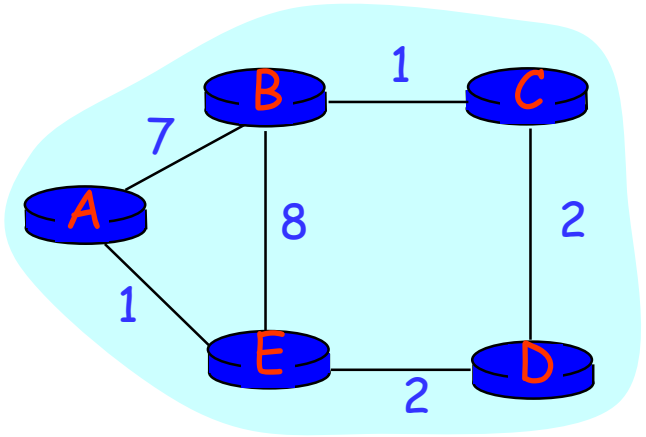


given these costs, find new routing…. resulting in new costs

Distance Vector Routing

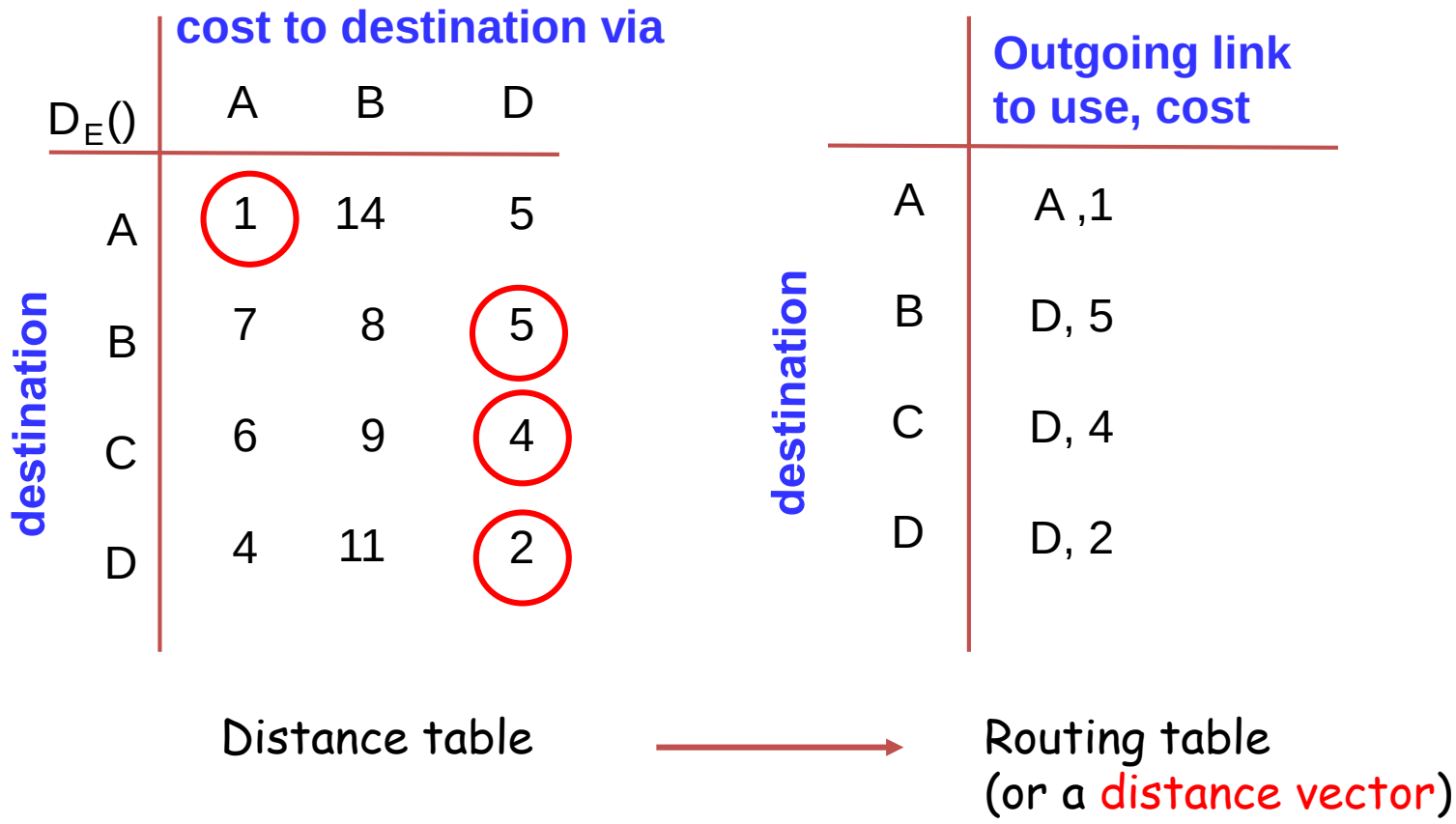
- A router tells neighbors its distance to every router
  - Communication between neighbors only
- Based on Bellman-Ford algorithm
  - Computes "shortest paths"
- Each router maintains a distance table
  - A row for each possible destination
  - A column for each neighbor
    - $D_X(Y,Z)$: distance from X to Y via Z
    - $D_X(Y)$: = $\min_Z\{D_x(Y,Z)\}$: shortest path from X to Y
- Exchanges distance vector with neighbors
  - Distance vector: current least cost from X to each destination

Distance Table: Example



$D_E()$	cost to destination via A	B	D
destination A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

From Distance Table to Routing Table



Distance Vector Algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

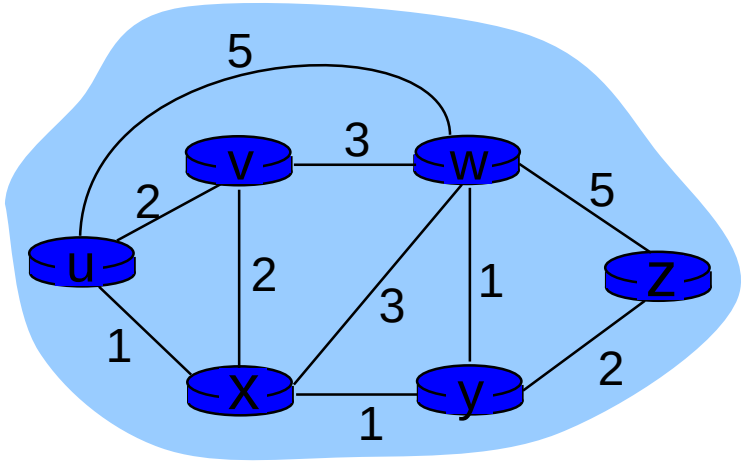
$$d^x(y) = \min_v \{c(x,v) + d^v(y)\}$$

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

Distance Vector Algorithm

key idea:

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

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

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Distance Vector Algorithm

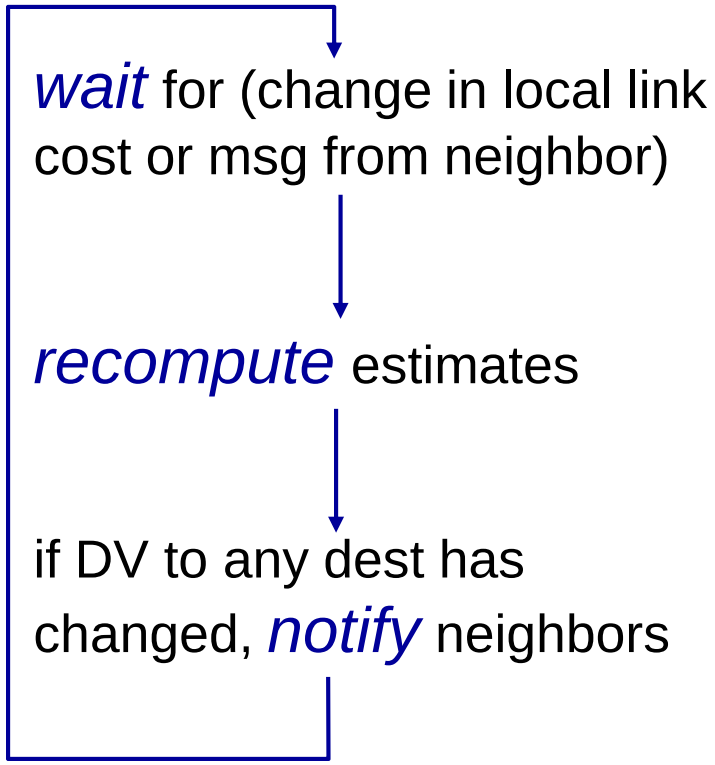
iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

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

each node:



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

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

node x table

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

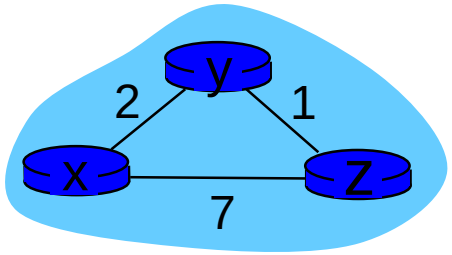
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

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

node z table

from \ cost to	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

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

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

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

node y table

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

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

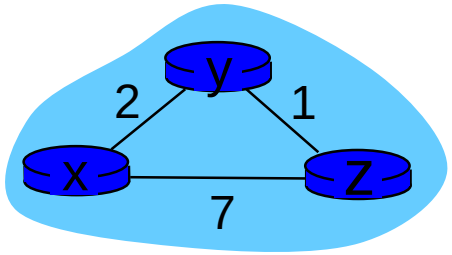
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

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

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

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

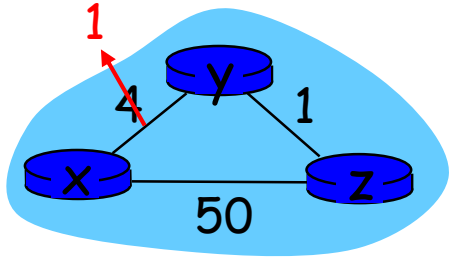


time

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.

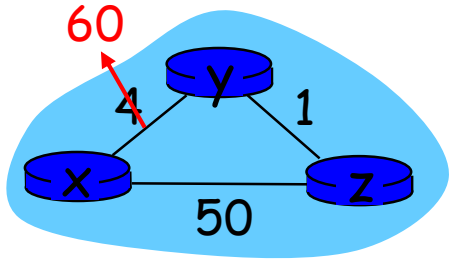
t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

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

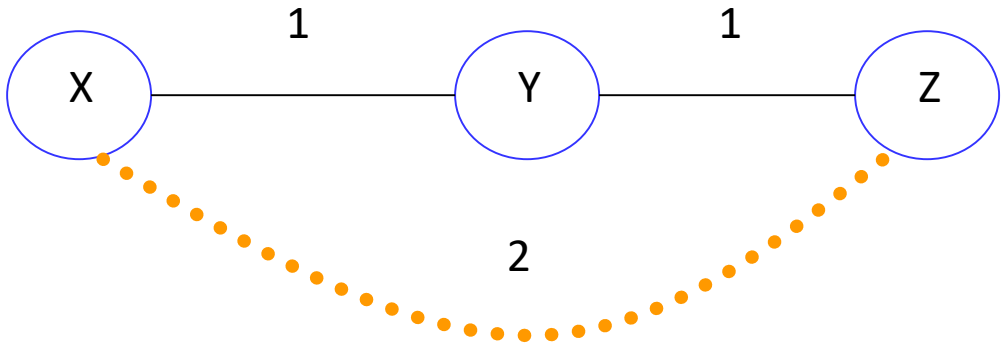
Distance Vector: Link Cost Changes

link cost changes:

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



"Count-to-Infinity" Problem: A Simple Example



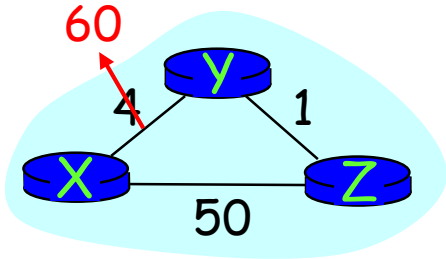
"Fixes" to Count-to-Infinity Problem

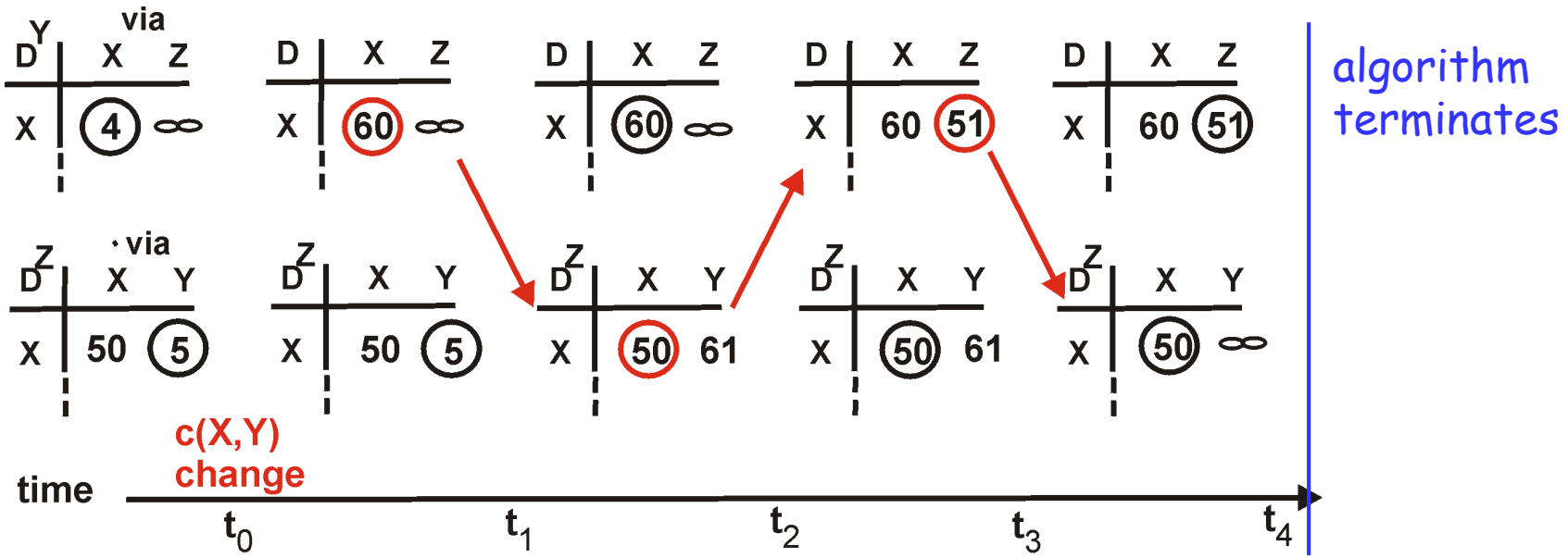
- Split horizon
  - A router never advertises the cost of a destination to a neighbor
    - If this neighbor is the next hop to that destination
- Split horizon with poisonous reverse
  - If X routes traffic to Z via Y, then
    - X tells Y that its distance to Z is infinity
      - Instead of not telling anything at all
  - Accelerates convergence

Split Horizon with 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)

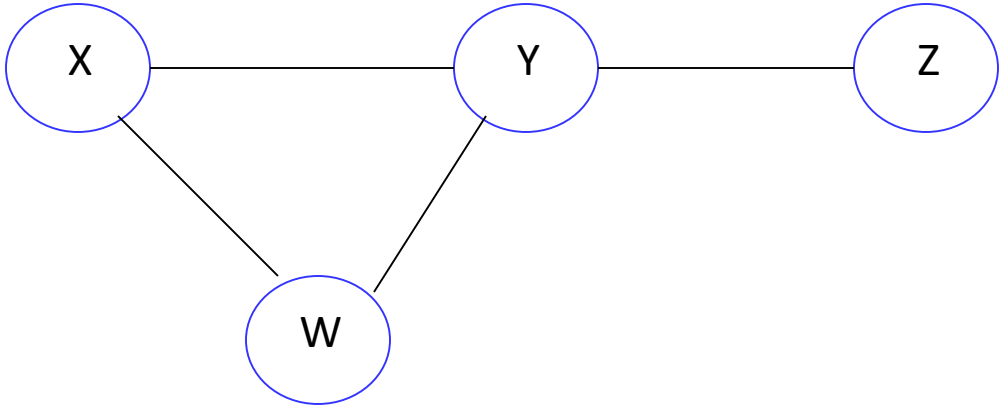




"Fixes" to Count-to-Infinity Problem

- Split horizon
  - A router never advertises the cost of a destination to a neighbor
    - If this neighbor is the next hop to that destination
- Split horizon with poisonous reverse
  - If X routes traffic to Z via Y, then
    - X tells Y that its distance to Z is infinity
      - Instead of not telling anything at all
  - Accelerates convergence
- Will this completely solve count to infinity problem?

Count-to-Infinity Problem Revisited



Link State *vs* Distance Vector

- Tells everyone about neighbors
- Controlled flooding to exchange link state
- Dijkstra's algorithm
- Each router computes its own table
- May have oscillations
- Open Shortest Path First (OSPF)

- Tells neighbors about everyone
- Exchanges distance vectors with neighbors
- Bellman-Ford algorithm
- Each router's table is used by others
- May have routing loops
- Routing Information Protocol (RIP)

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

Routing in the Real World

Our routing study thus far - idealization

- all routers identical
- network "flat"

How to do routing in the Internet

- scalability and policy issues

scale: with 200 million destinations:

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

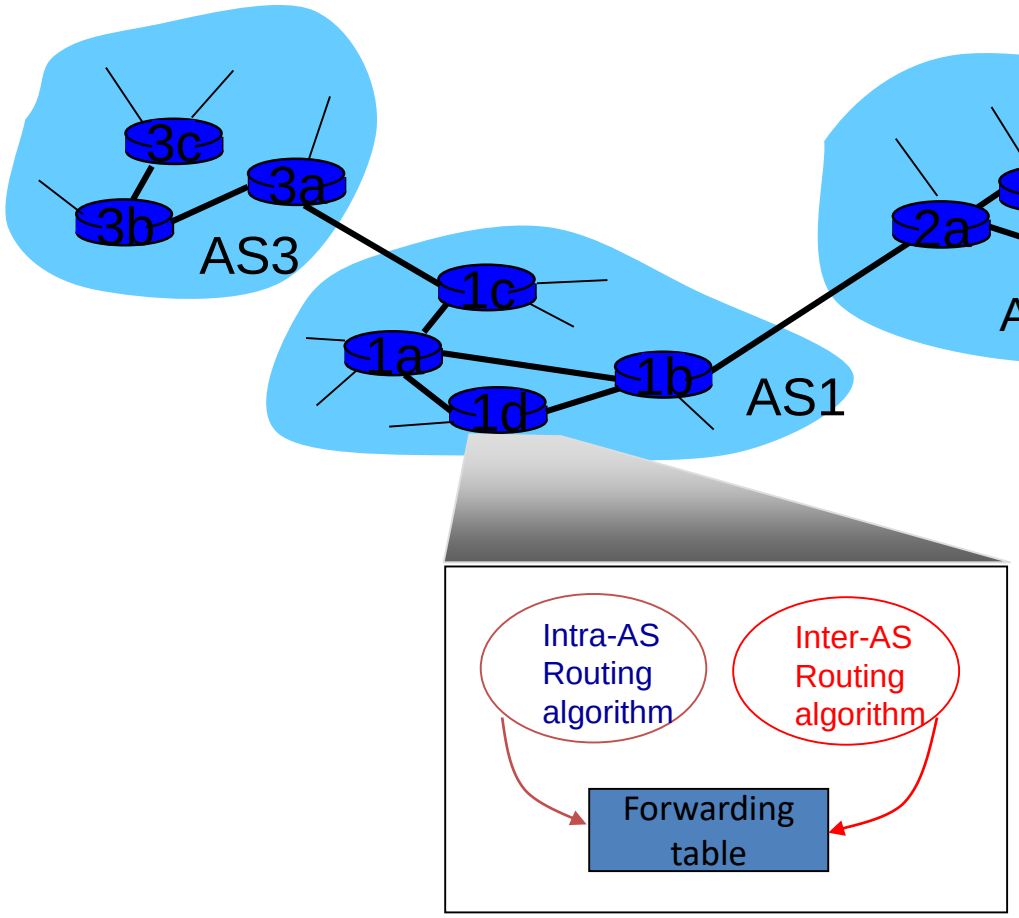
administrative autonomy

- internet = network of networks
- each network admin may want to control routing in its own network

Routing in the Internet

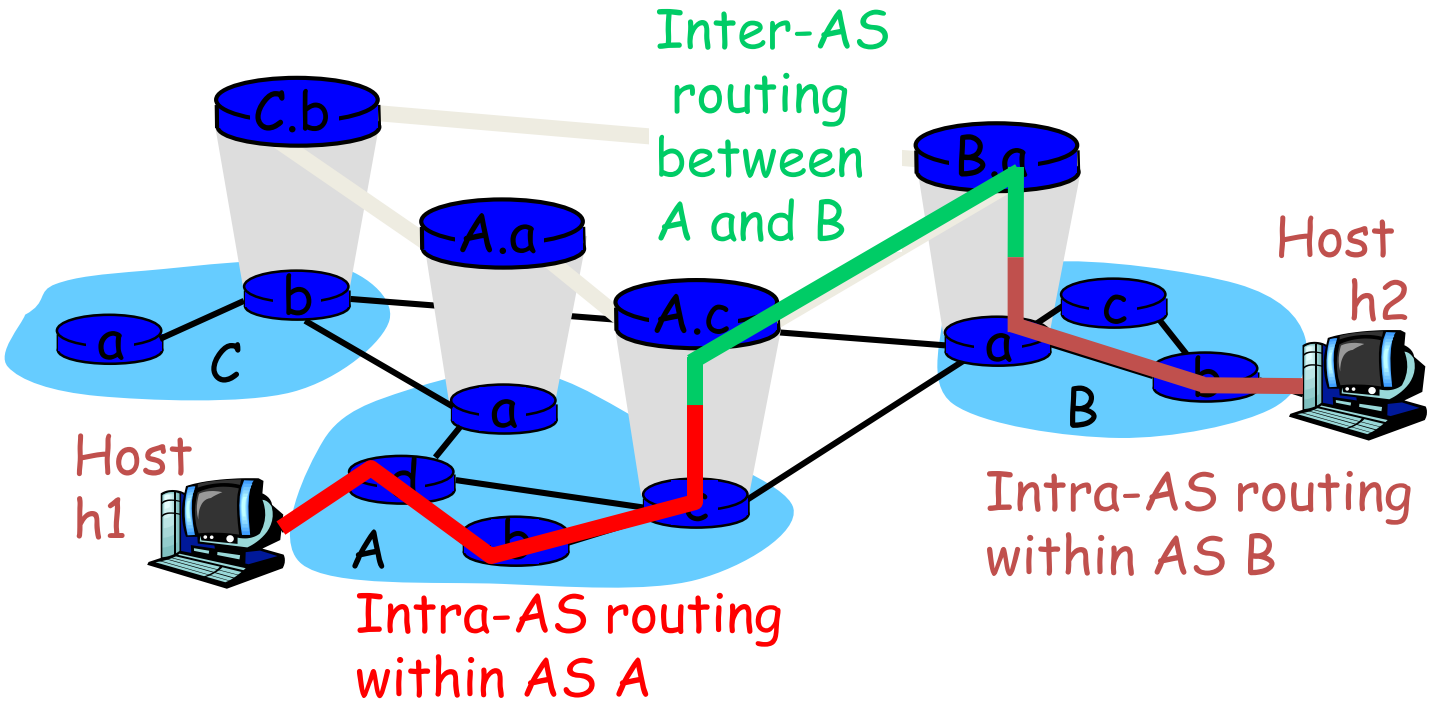
- The Global Internet consists of Autonomous Systems (AS) interconnected with each other:
  - Stub AS: small corporation: one connection to other AS's
  - Multihomed AS: large corporation (no transit): multiple connections to other AS's
  - Transit AS: provider, hooking many AS's together
- Two-level routing:
  - Intra-AS: administrator responsible for choice of routing algorithm within network
  - Inter-AS: unique standard for inter-AS routing: BGP

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

Intra-AS vs. Inter-AS Routing



Why Different Intra- and 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, update traffic

Performance:

- Intra-AS: can focus on performance
- Inter-AS: policy may dominate over performance

Will Talk about Inter-AS routing (& BGP) later!

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: Intermediate System to Intermediate System (OSI Standard)
  - EIGRP: Extended Interior Gateway Routing Protocol (Cisco proprietary)

RIP (Routing Information Protocol)

- Distance vector algorithm
- Included in BSD-UNIX Distribution in 1982
- Distance metric: # of hops (max = 15 hops)
  - *Can you guess why?*

- Distance vectors: exchanged among neighbors every 30 sec via Response Message (also called **advertisement**)
- Each advertisement: list of up to 25 destination nets within AS

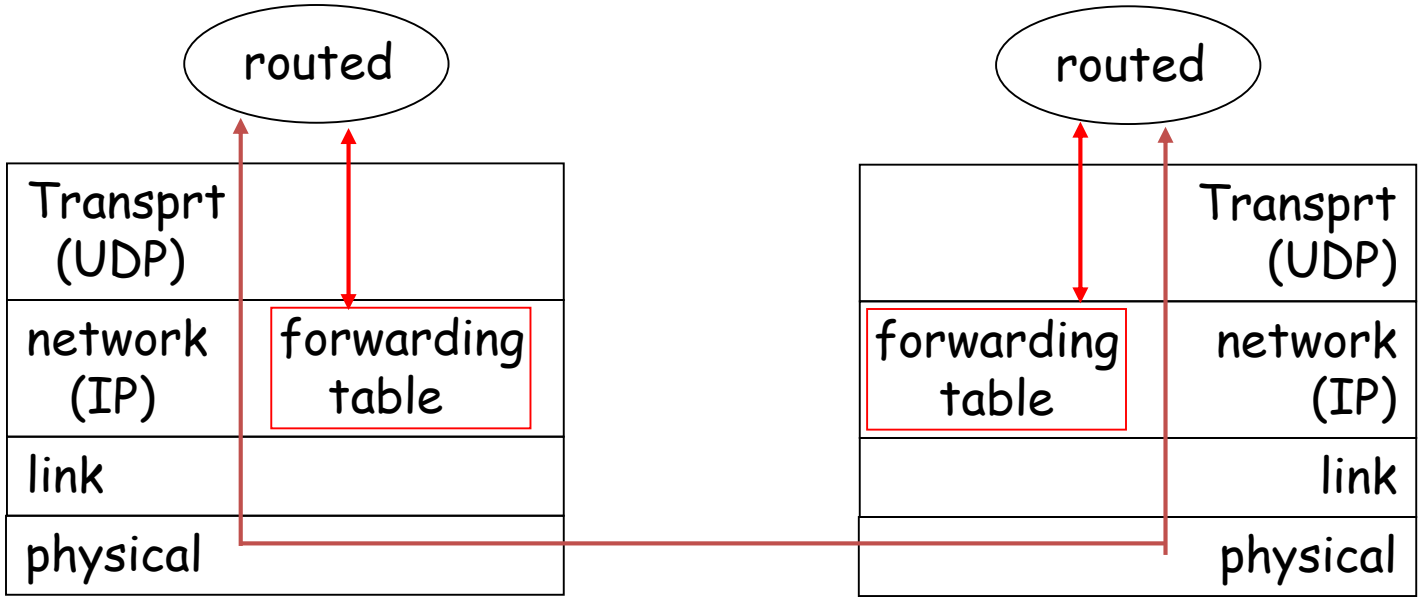
RIP: Link Failure and Recovery

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

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

RIP Table Processing

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



OSPF (Open Shortest Path First)

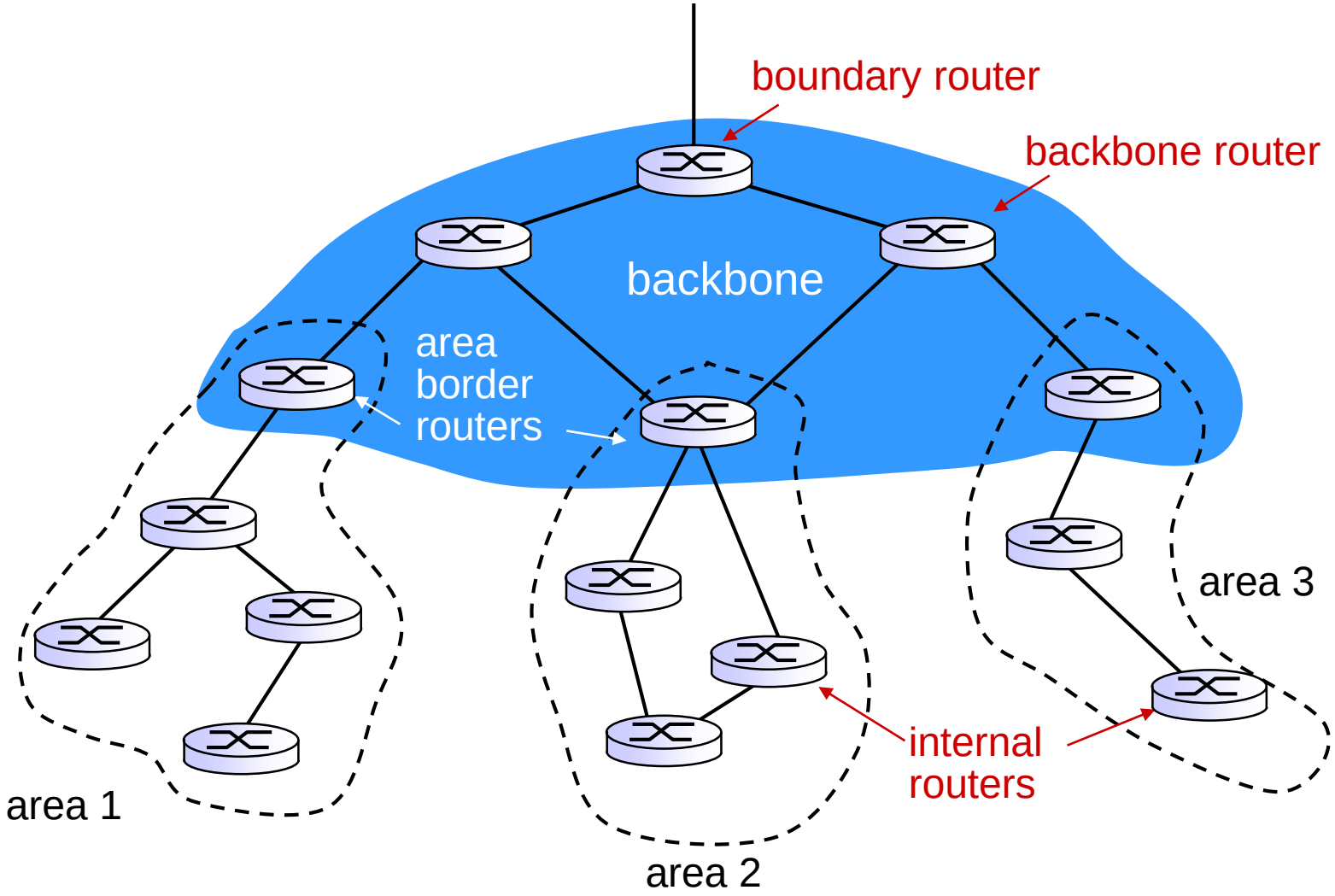
- "open": publicly available
- uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - route computation using Dijkstra's algorithm
- router floods OSPF link-state advertisements to all other routers in *entire* AS
  - carried in OSPF messages directly over IP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

OSPF "Advanced" Features (not in RIP)

- 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 ("Type-of-Services")
  - e.g., satellite link cost set "low" for best effort; high for real time)

- Hierarchical OSPF in large domains.

Hierarchical OSPF



Hierarchical OSPF

- Two-level hierarchy: local area, backbone.
  - Link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- Area border routers: "summarize" distances to nets in own area, advertise to other Area Border routers.
- Backbone routers: run OSPF routing limited to backbone.
- Boundary routers: connect to other ASes.

Software Defined Networking (SDN)

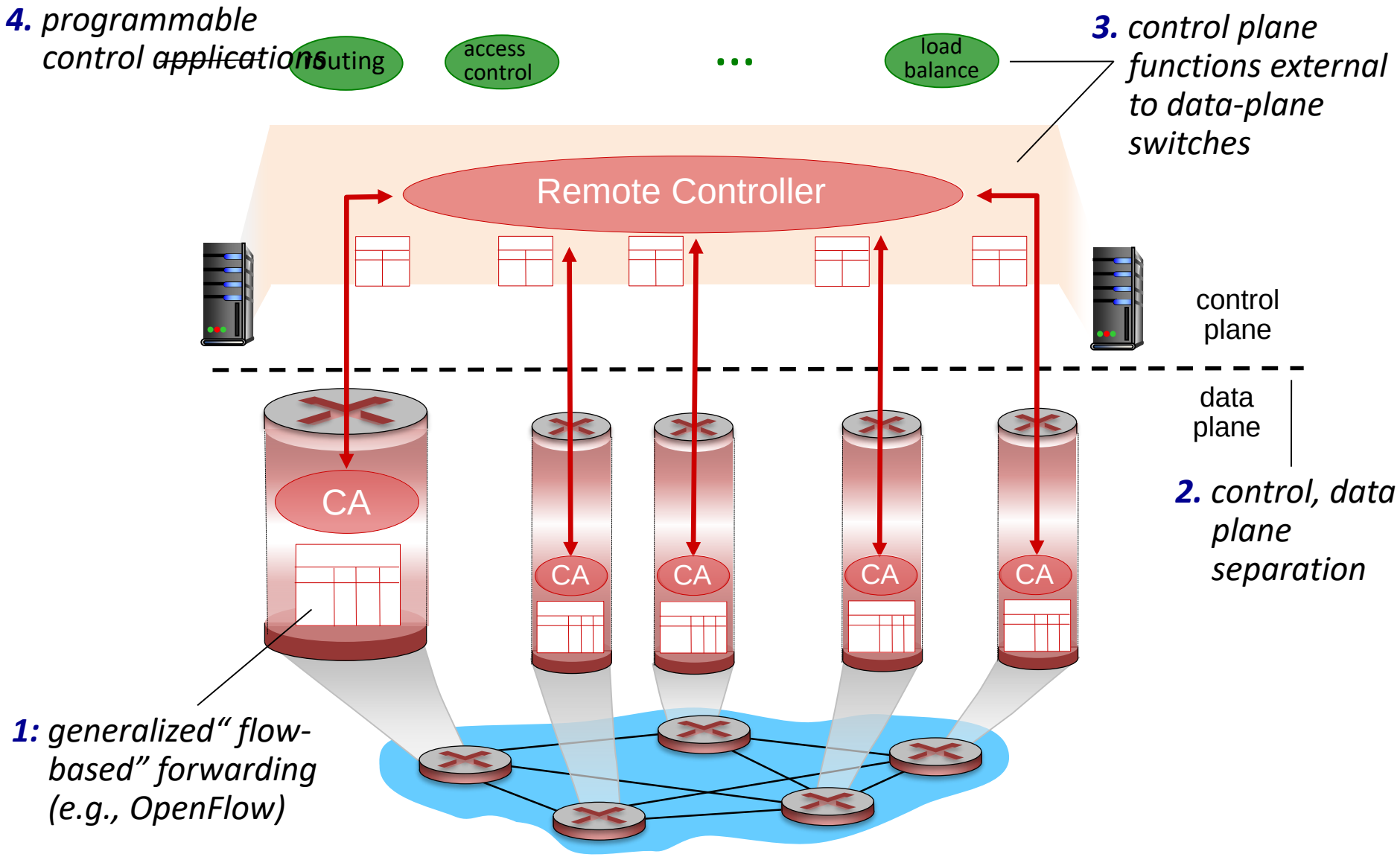
- Internet network layer: historically has been implemented via distributed, per-router approach
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

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

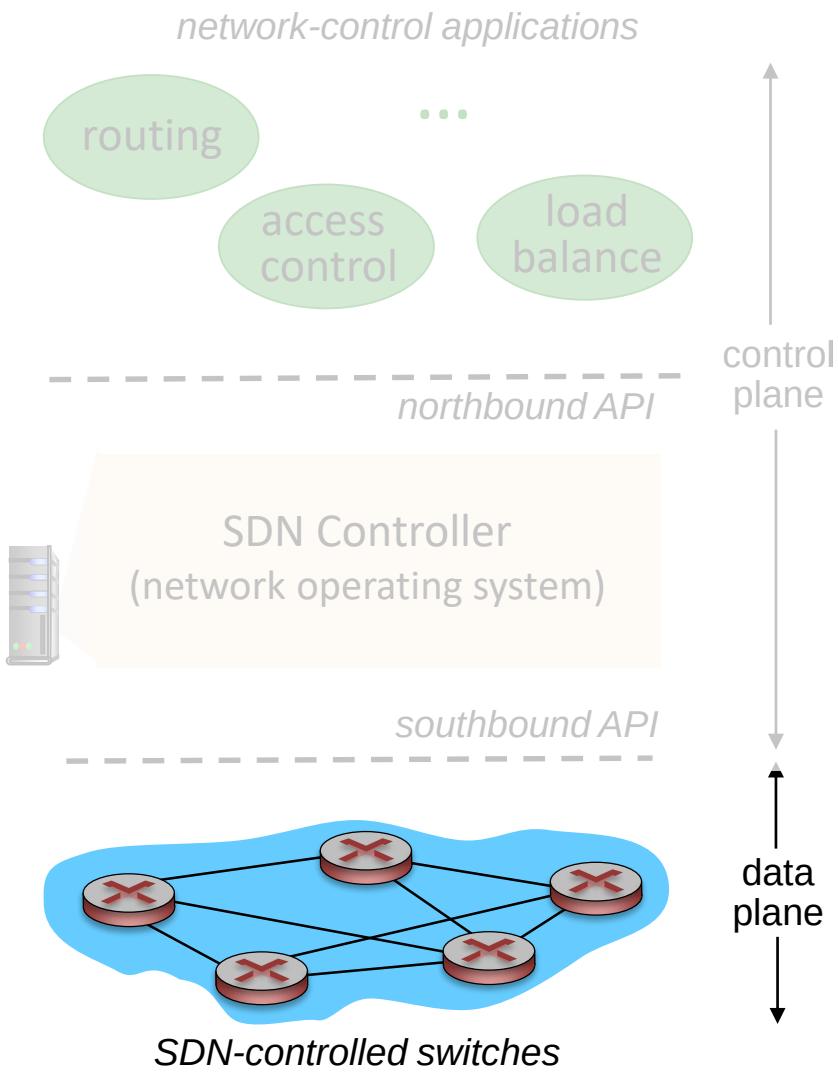
Software Defined Networking (SDN)



SDN Perspective: Data Plane Switches

Data plane switches

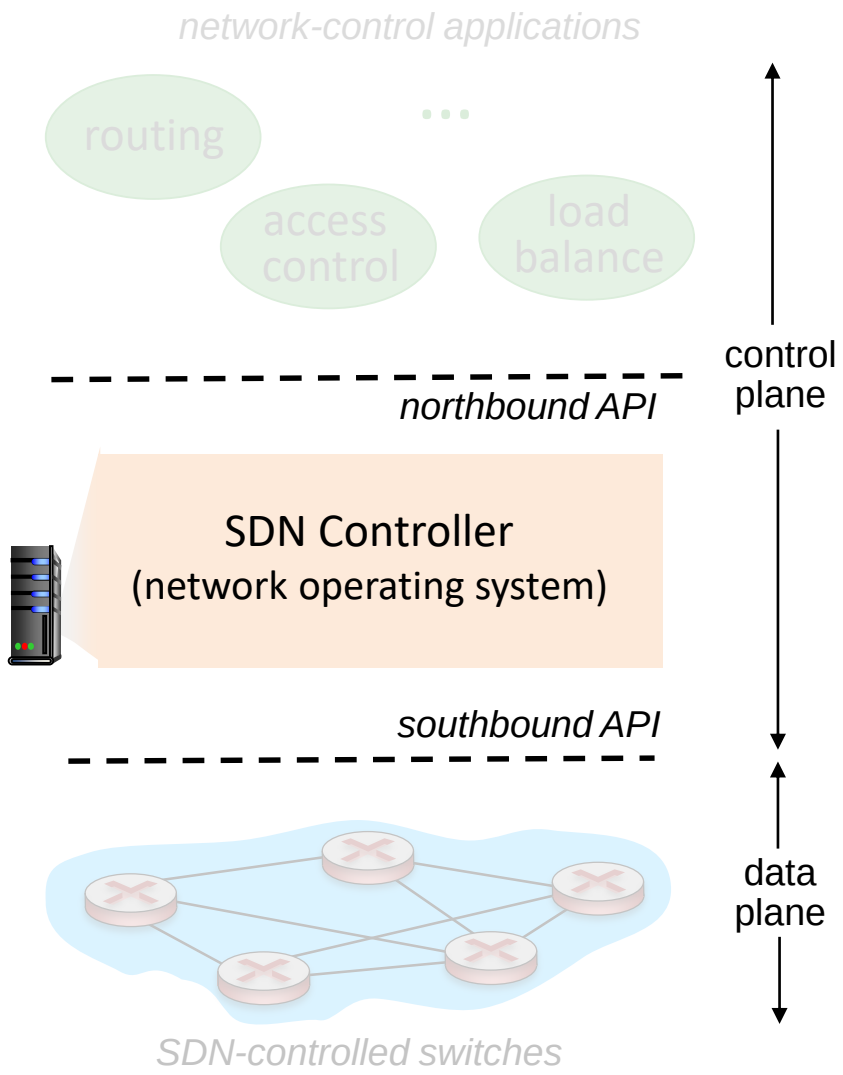
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller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



SDN perspective: SDN Controller

SDN controller (network OS):

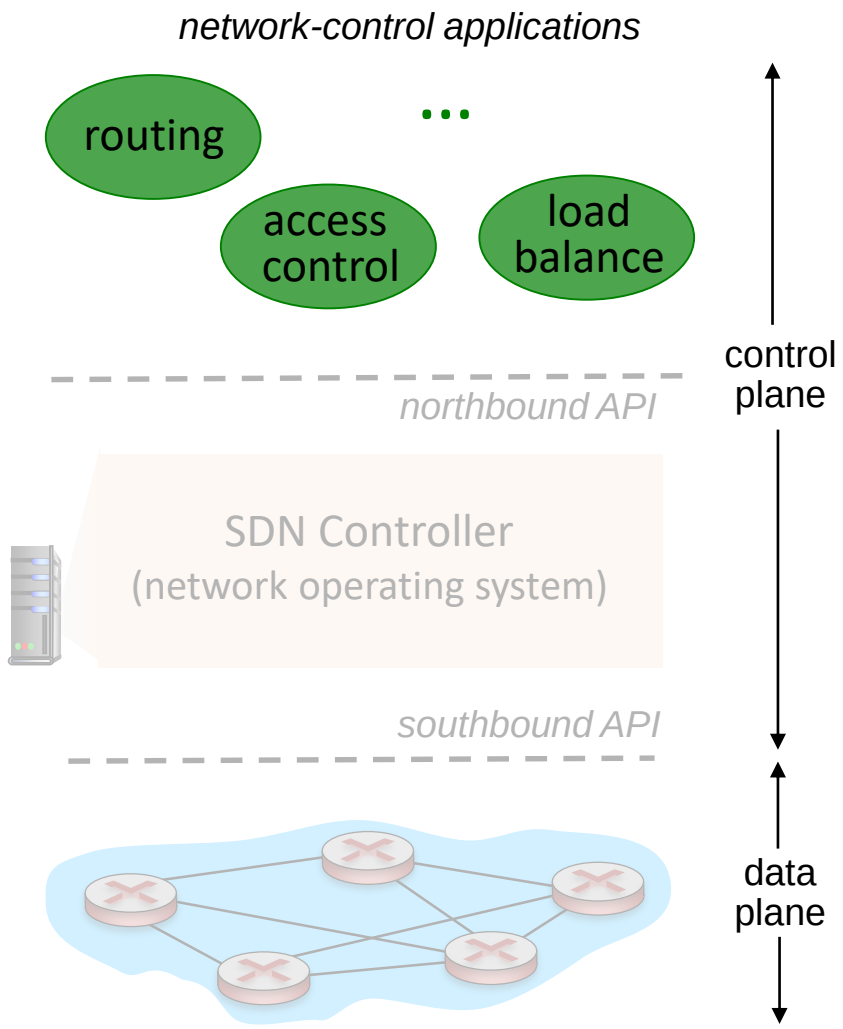
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



SDN Perspective: Control Applications

network-control apps:

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

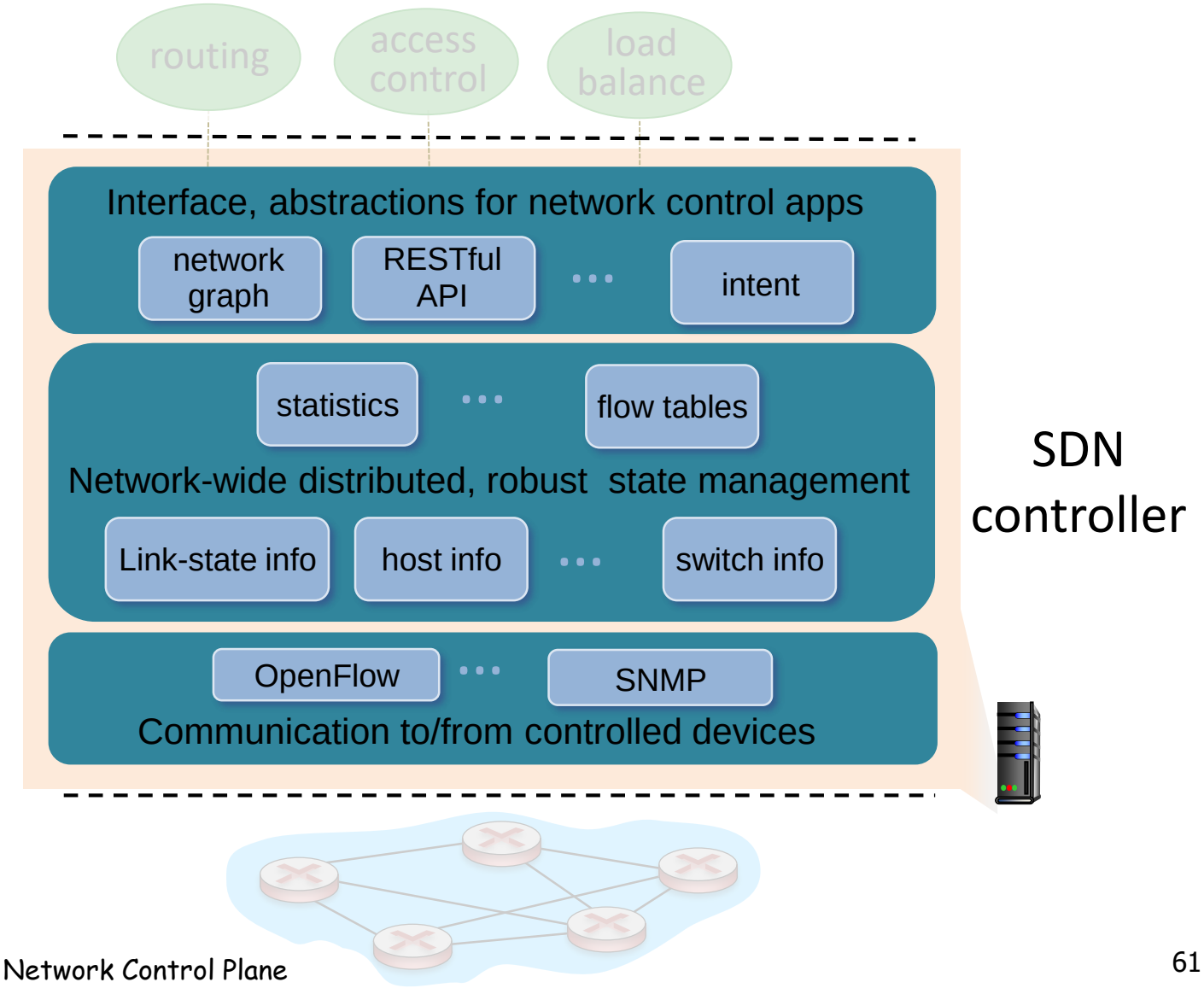


Components of SDN Controller

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



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