# Chapter 5
# Network Layer: Control Plane

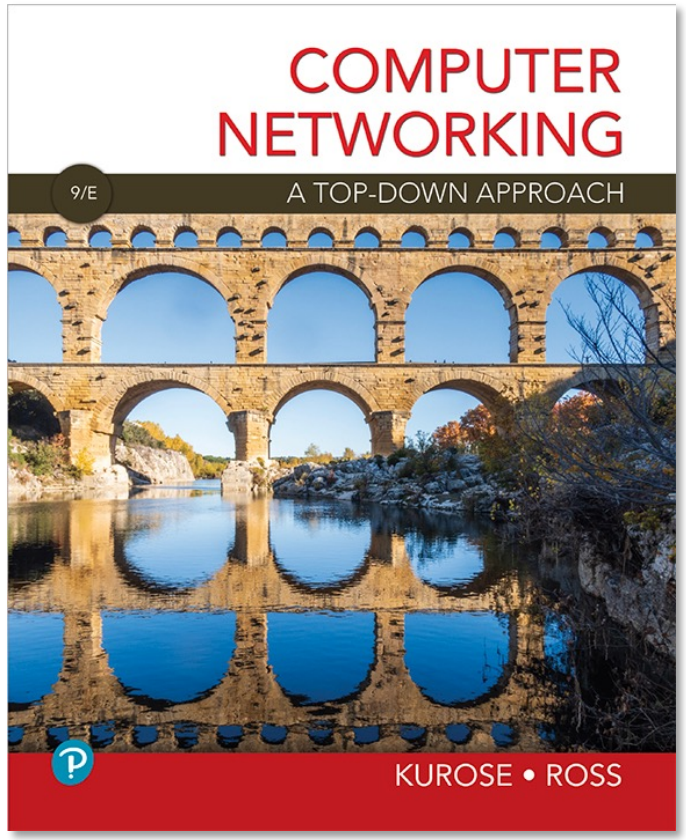
A note on the use of these PowerPoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent 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.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR






*Computer Networking: A Top-Down Approach*

Jim Kurose, Keith Ross

# Review: network layer data plane

- ✓Network layer: overview
- ✓What's inside a router
- ✓IP: the Internet Protocol
- ✓Generalized Forwarding
- ✓Middleboxes

*Question:* how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

*Answer:* by the control plane

# Network layer control plane: our goals

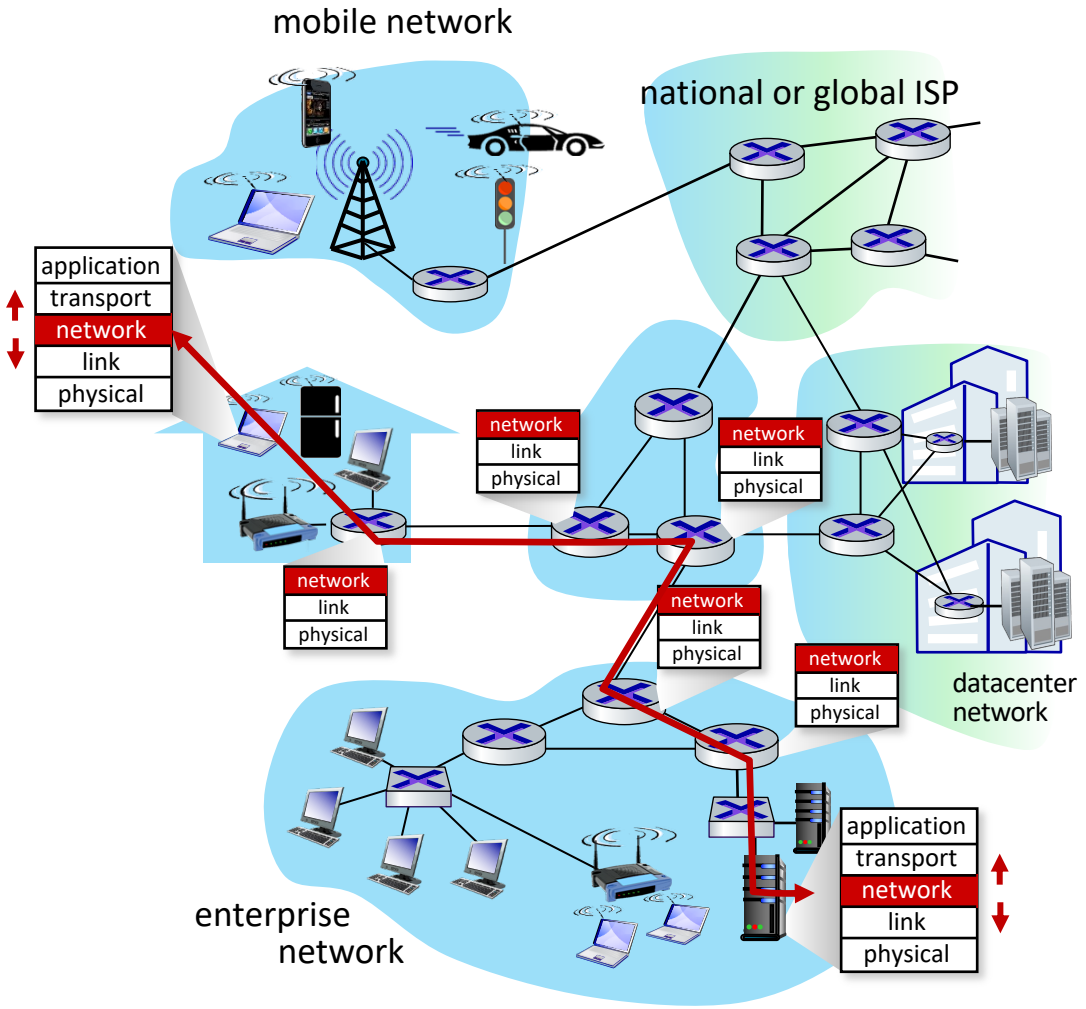
- understand principles behind network control plane:
  - traditional routing algorithms
  - SDN controllers
  - network management, configuration
- instantiation, implementation in the Internet:
  - OSPF, BGP
  - OpenFlow, ODL and ONOS controllers
  - Internet Control Message Protocol: ICMP
  - SNMP, YANG/NETCONF

# Network layer: control plane

- understand principles behind network control plane:
  - traditional routing algorithms
  - SDN controllers
- instantiation, implementation on the Internet:
  - OSPF, BGP
  - OpenFlow and ONOS controllers
  - Internet Control Message Protocol: ICMP

# Network Layer Services and Protocols

- transport segment from sending to receiving host
  - sender: encapsulates segments into datagrams, passes to link layer
  - receiver: delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- routers:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path



# 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)

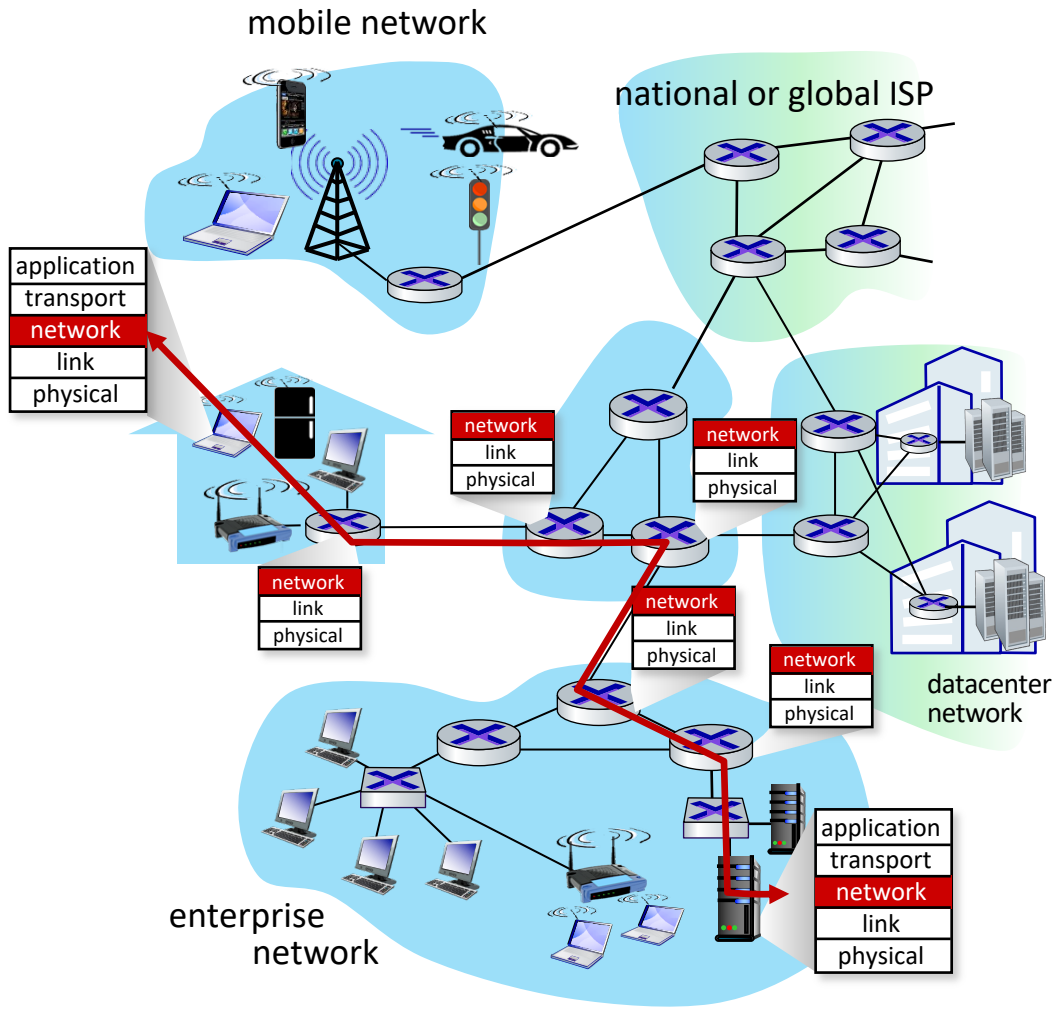
# Control Plane Outline

- Routing Algorithm
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing Among ISPs: BGP
- SDN Control Plane
- Internet Control Message Protocol

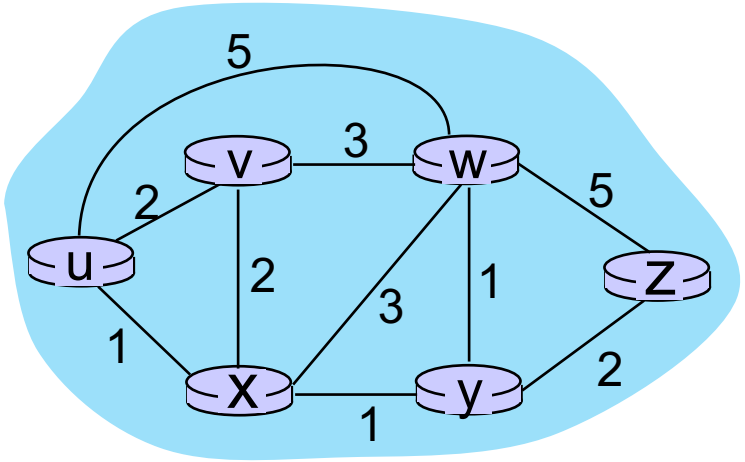
# Routing Protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets traverse from given initial source host to final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!



# Graph Abstraction: link costs




$c_{a,b}$: cost of *direct* link connecting *a* and *b*

*e.g.*, $c_{w,z} = 5$, $c_{u,z} = \infty$

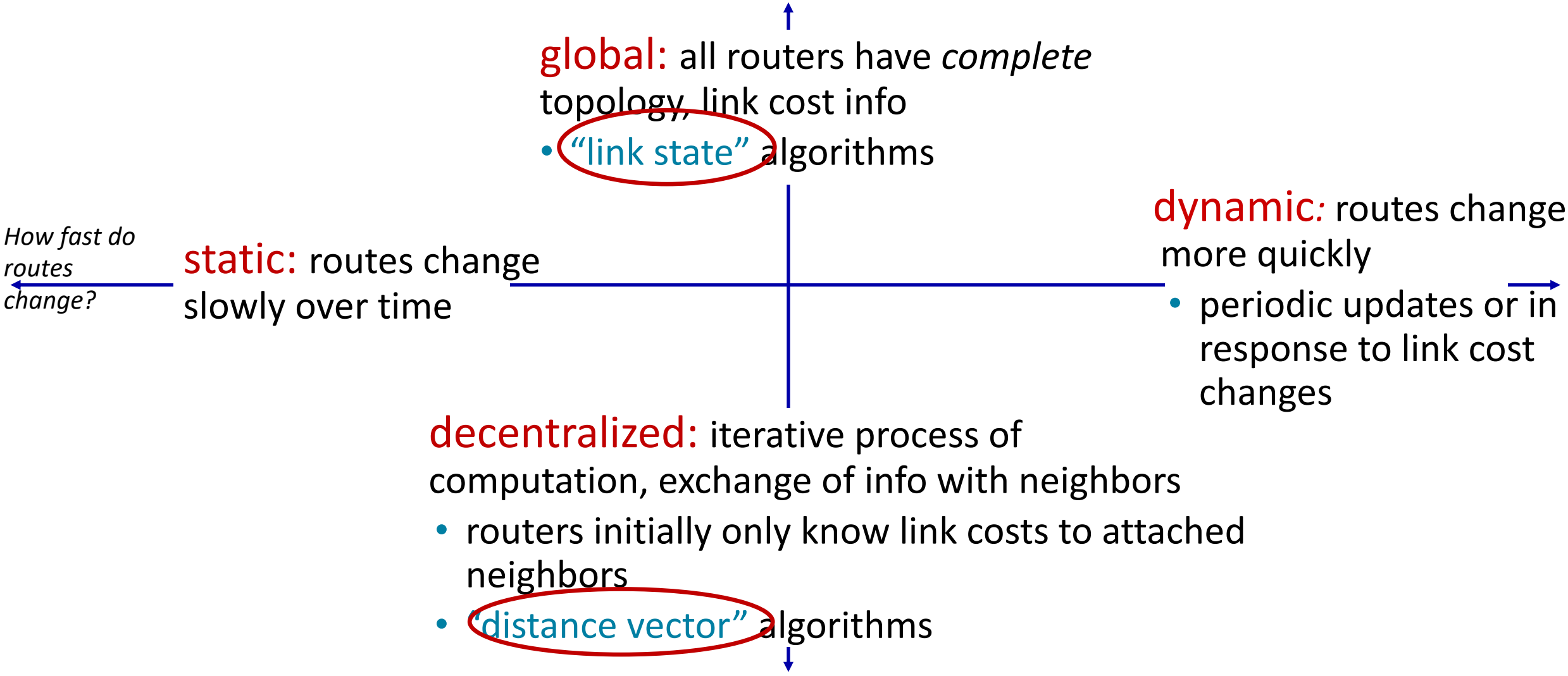
cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

graph: *G = (N,E)*

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

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

# Routing Algorithm Classification



# Dijkstra's Link-state Routing Algorithm

- Dijkstra's Algorithm – The shortest path algorithm
- centralized: network 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* destinations

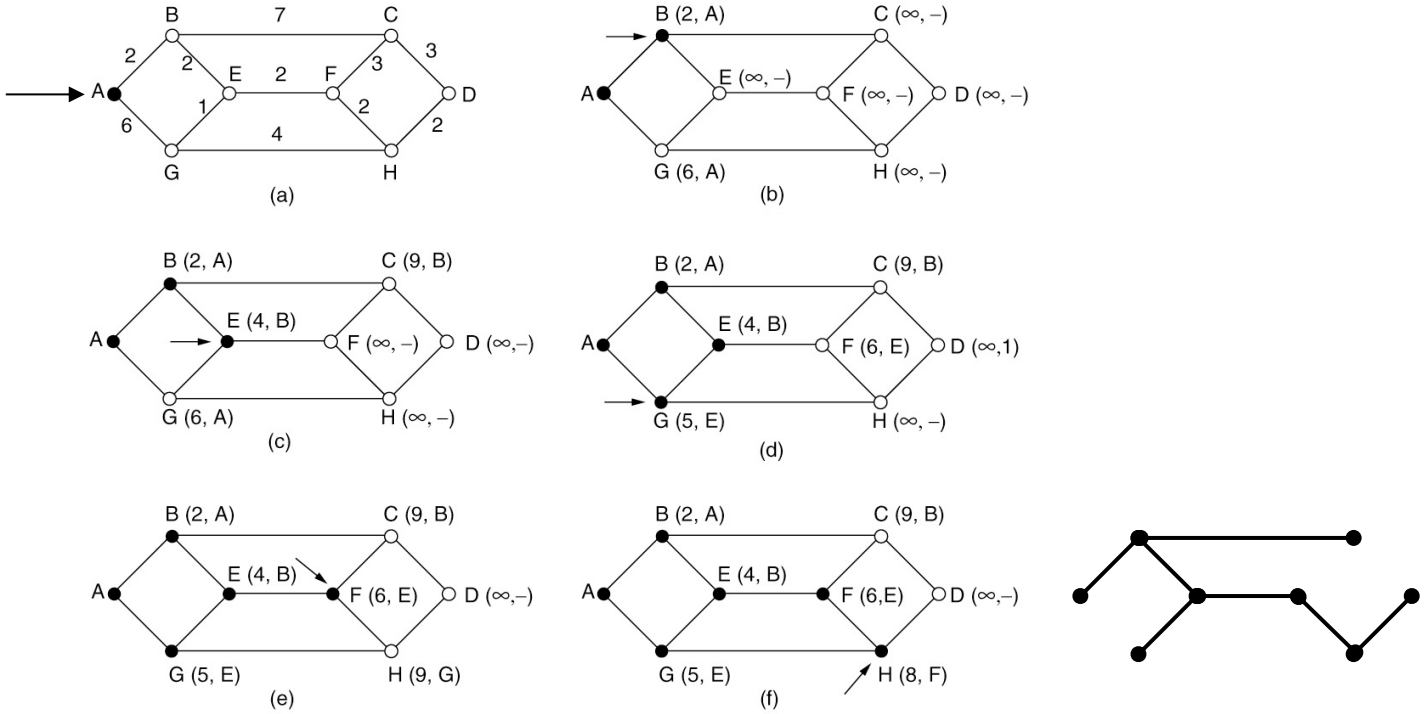
# Dijkstra's Link-state Routing Algorithm

- Shortest path: to choose a route between a given pair of routers, finds the shortest path between them on the graph

Wait! What is a path length!

The total sum of these weights for the edges included in the shortest path

The number indicates the distance/cost/weight for this connection between two routers




The first six steps used in computing the shortest path from *A* to *D*. The arrows indicate the working node.

# Dijkstra's Link-state Routing Algorithm

- Given a connected graph, Dijkstra's algorithm builds a shortest path algorithm rooted at a distinguished node
  1. Mark every node as unscanned and give each node a label of INF
  2. Set the label of the root to 0 and the predecessor of the root to itself. The root will be the only node that is its own predecessor
  3. Loop until you have scanned all the nodes
     1. Find the node n with the smallest label. Since the label represents the distance to the root we call it d_min
     2. Mark the node as scanned
     3. Scan all the *adjacent* nodes m and see if the distance to the root through n is shorter than the distance stored in the label of m. if it is, update the label and update pred [m] = n
     4. $Min_{neighbors}$(dist(root, neighbor) + dist(neighbor, node))
  4. When the loop finishes, we have a tree stored in pred format rooted at the root

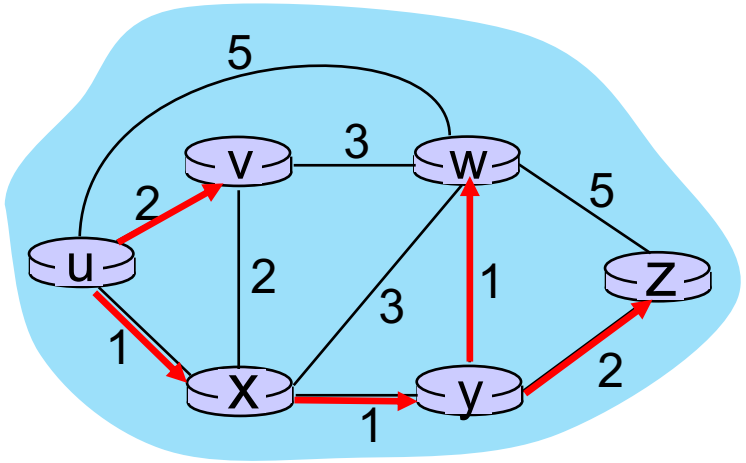
# Dijkstra's Link-state Routing Algorithm (cont.)

```
#define MAX_NODES 1024                          /* maximum number of nodes */
#define INFINITY 1000000000                     /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];              /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                                /* the path being worked on */
      int predecessor;                          /* previous node */
      int length;                               /* length from source to this node */
      enum {permanent, tentative} label;        /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) {     /* initialize state */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                                        /* k is the initial working node */
```

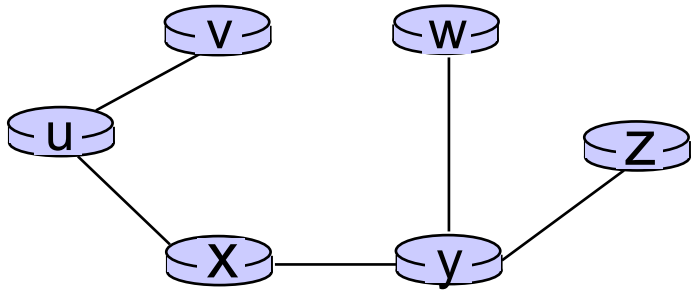
# Dijkstra's Link-state Routing Algorithm (cont.)

```
do {                                                    /* Is there a better path from k? */
    for (i = 0; i < n; i++)                             /* this graph has n nodes */
        if (dist[k][i] != 0 && state[i].label == tentative) {
            if (state[k].length + dist[k][i] < state[i].length) {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }

    /* Find the tentatively labeled node with the smallest label. */
    k = 0; min = INFINITY;
    for (i = 0; i < n; i++)
        if (state[i].label == tentative && state[i].length < min) {
            min = state[i].length;
            k = i;
        }
    state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0;  k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

# Dijkstra's algorithm: an example




resulting least-cost-path tree from u:




resulting forwarding table in u:

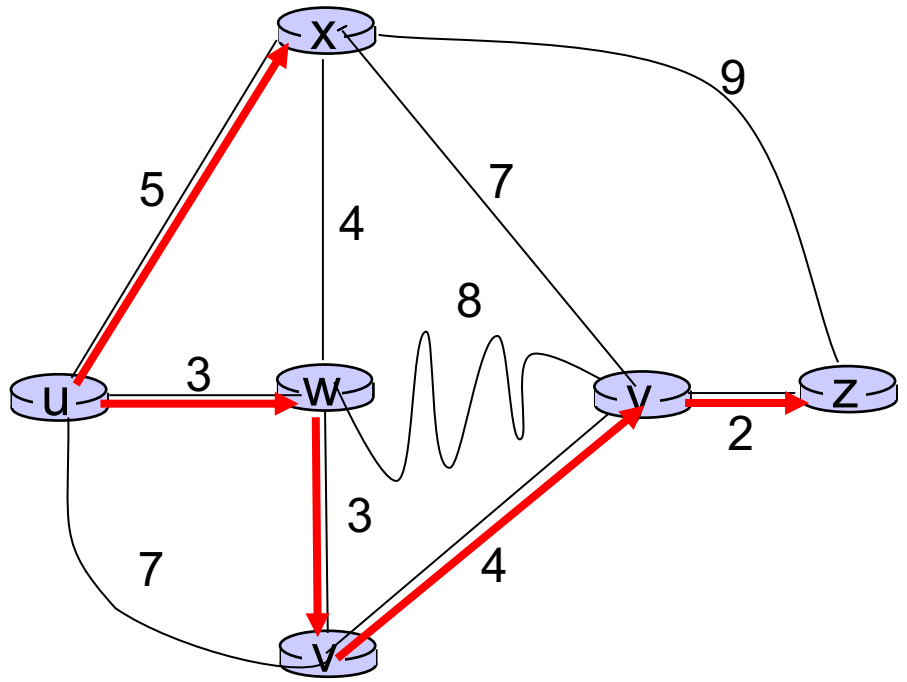
| destination | outgoing link |
|---|---|
| v | (u,v) |
| x | (u,x) |
| y | (u,x) |
| w | (u,x) |
| z | (u,x) |

route from *u* to *v* directly

route from u to all other destinations via *x*

# Dijkstra's algorithm: another example

| Step | N′ | v D(v), p(v) | w D(w), p(w) | x D(x), p(x) | y D(y), p(y) | z 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 least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

# Dijkstra's algorithm: discussion

algorithm complexity: $n$ nodes

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

message complexity:

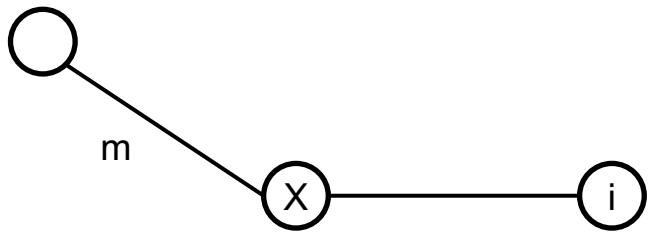
- each router must *broadcast* its link state information to other $n$ routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

# Control Plane Outline

- Routing Algorithm
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing Among ISPs: BGP
- SDN Control Plane
- Internet Control Message Protocol

# Distance Vector Algorithm

- Distance vector algorithm operates by having each router maintain a vector table giving the best known distance to each destination and which line to use to get there. The tables are updated by exchanging information with the neighbors
- Vector table: one entry for each router in the subnet; each entry contains two parts: preferred outgoing line to use for that destination and an estimate of the time or distance to the destination
- The router is assumed to know the distance to each neighbor and update the vector table periodically by changing it with neighbors
  - # hops
  - Delay (ECHO)



# Distance Vector Algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

**Bellman-Ford equation**

Let $D_x(y)$: cost of least-cost path from $x$ to $y$.

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

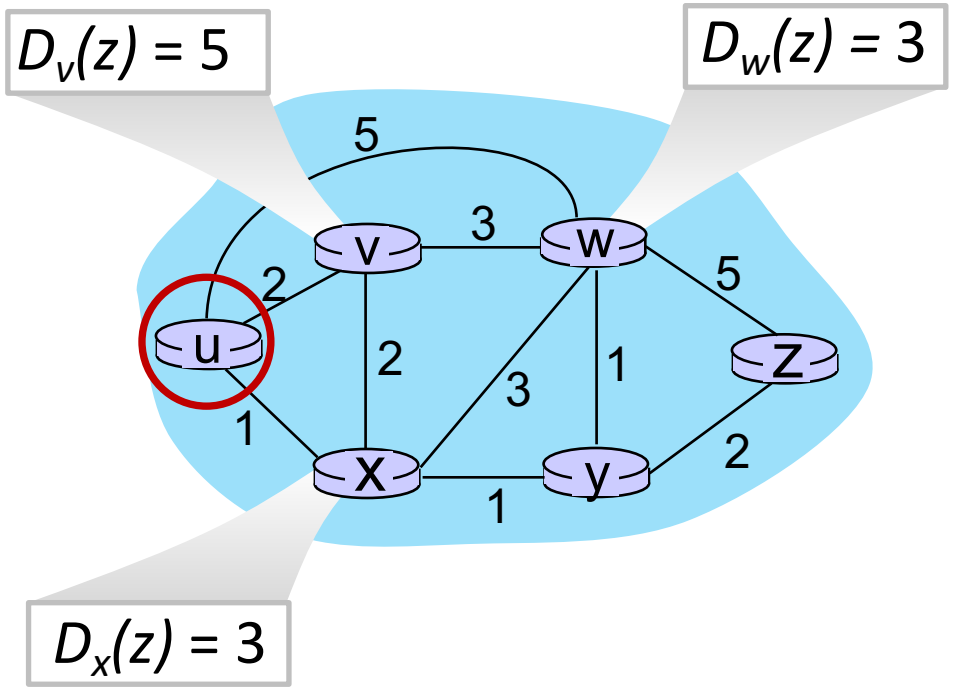
$v$'s estimated least-cost-path cost to $y$

*min* taken over all neighbors $v$ of $x$

direct cost of link from $x$ to $v$

# Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x,v,w,* know that for destination *z*:




Bellman-Ford 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 (x) is next hop on estimated least-cost path to destination (z)*



UCCS

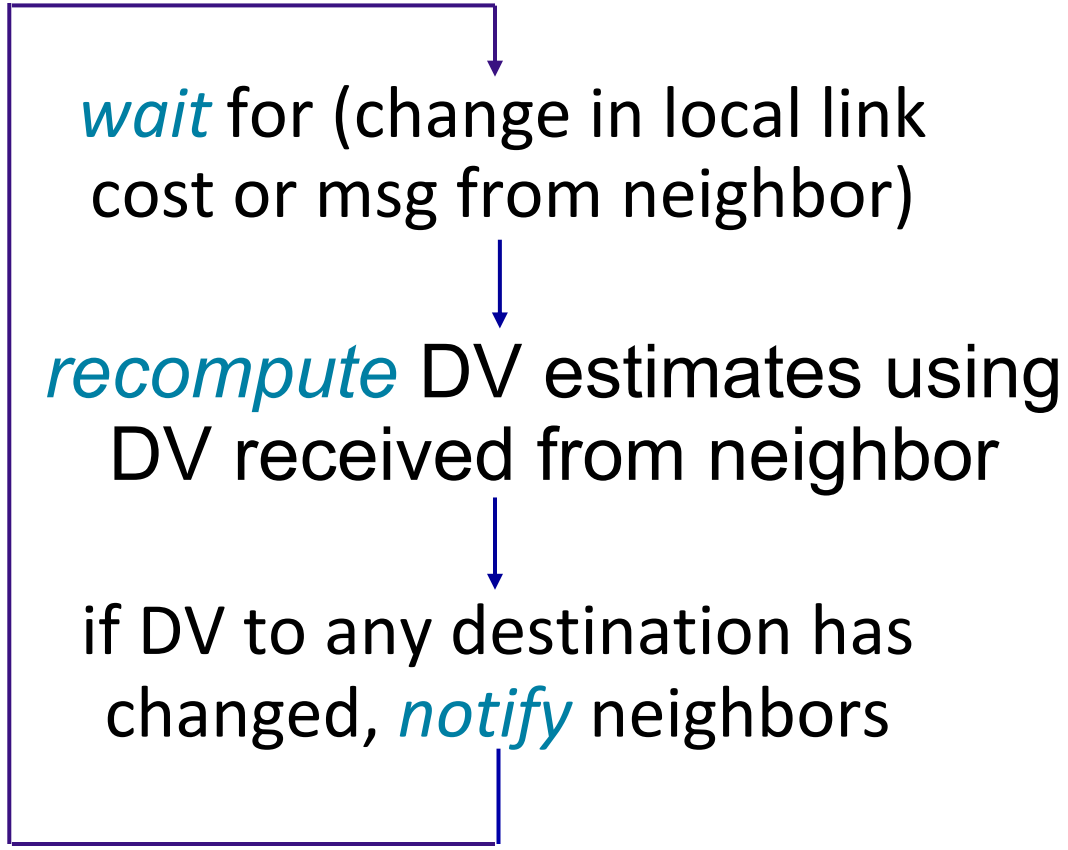
# Distance Vector Algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:
  - $D_x(y) \leftarrow min_v\{c_{x,v} + D_v(y)\}$ 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

each node:




iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

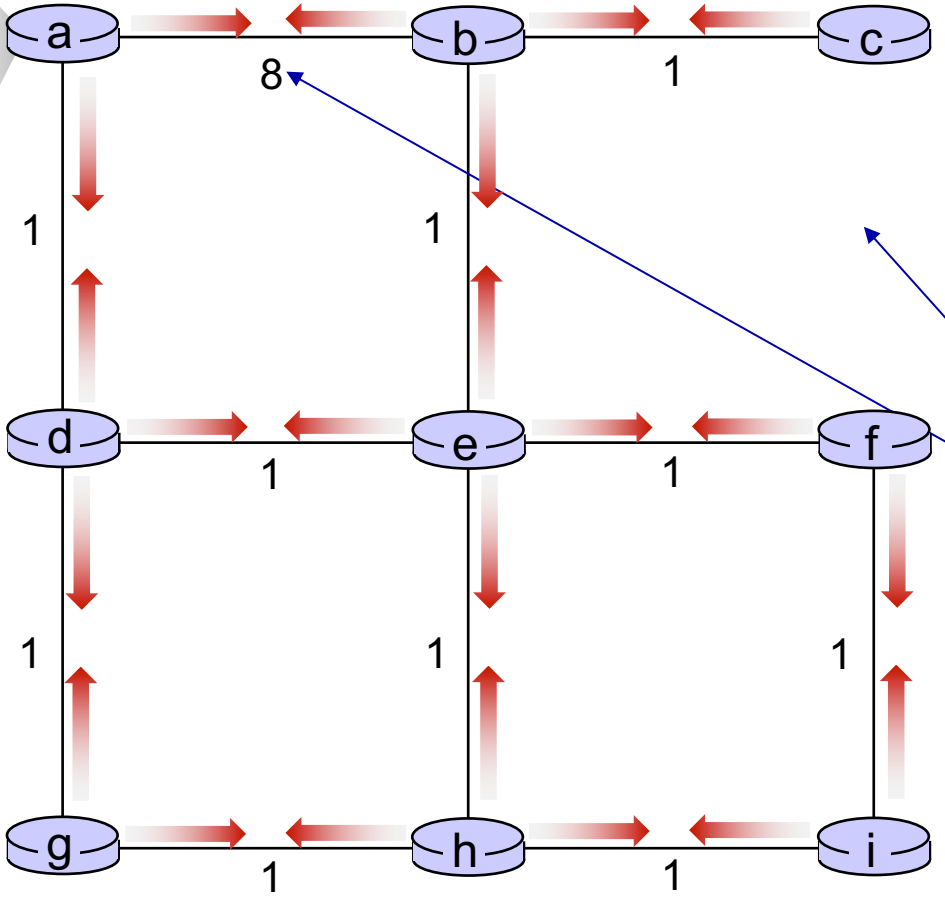
- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

# Distance Vector: example

t=0

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

| DV in a: |
| --- |
| $D_a(a)=0$ |
| $D_a(b) = 8$ |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$ |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



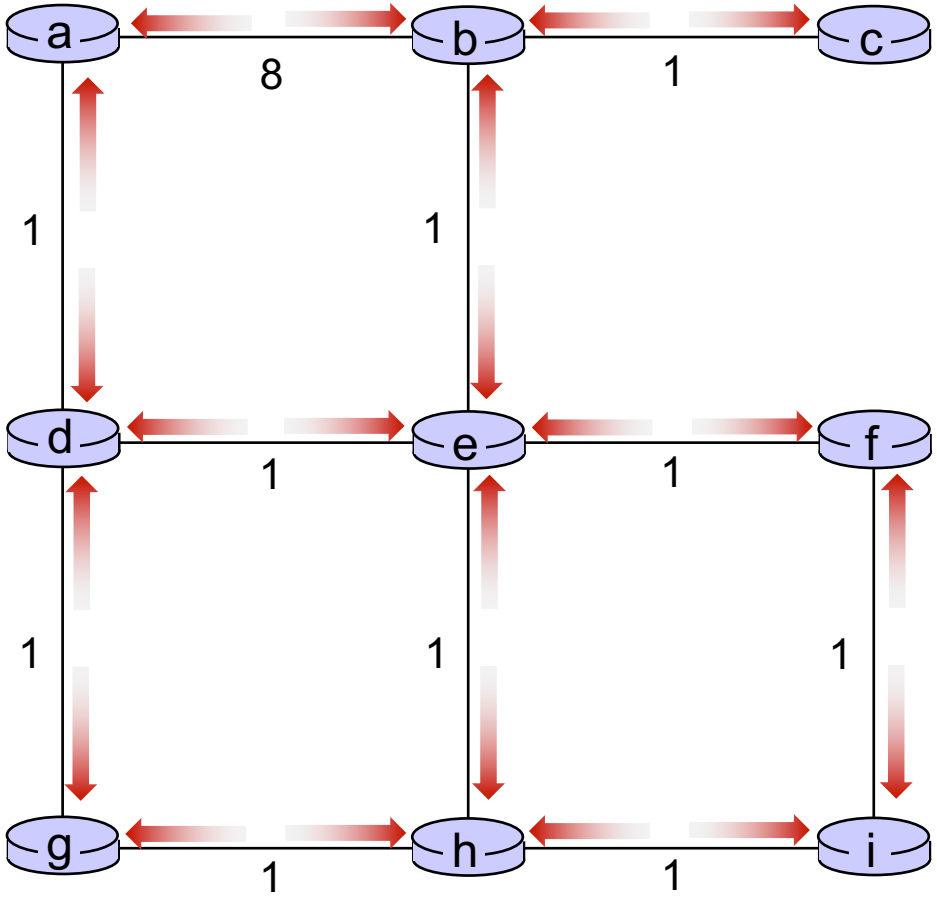

A few asymmetries:
- missing link
- larger cost

# Distance Vector Example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

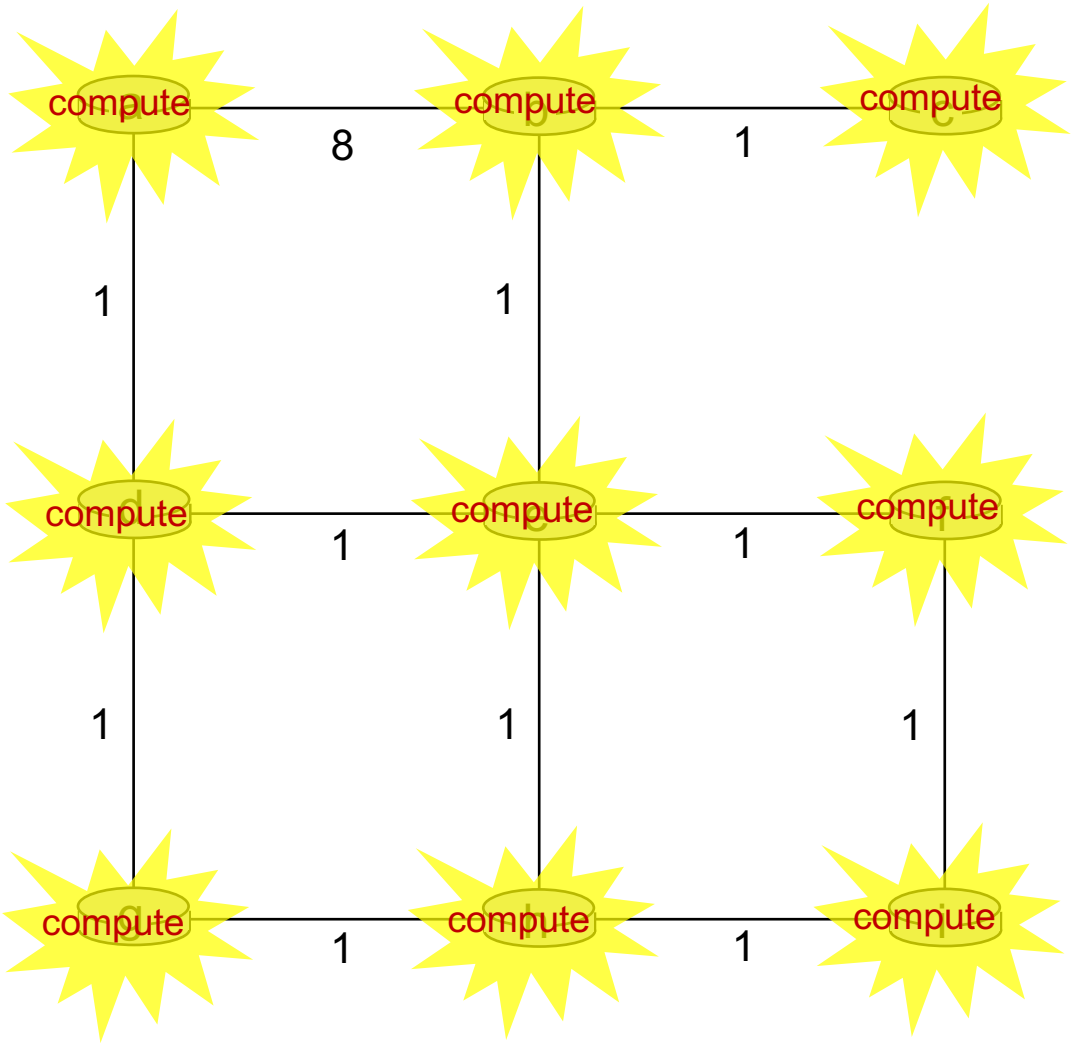


# Distance Vector Example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

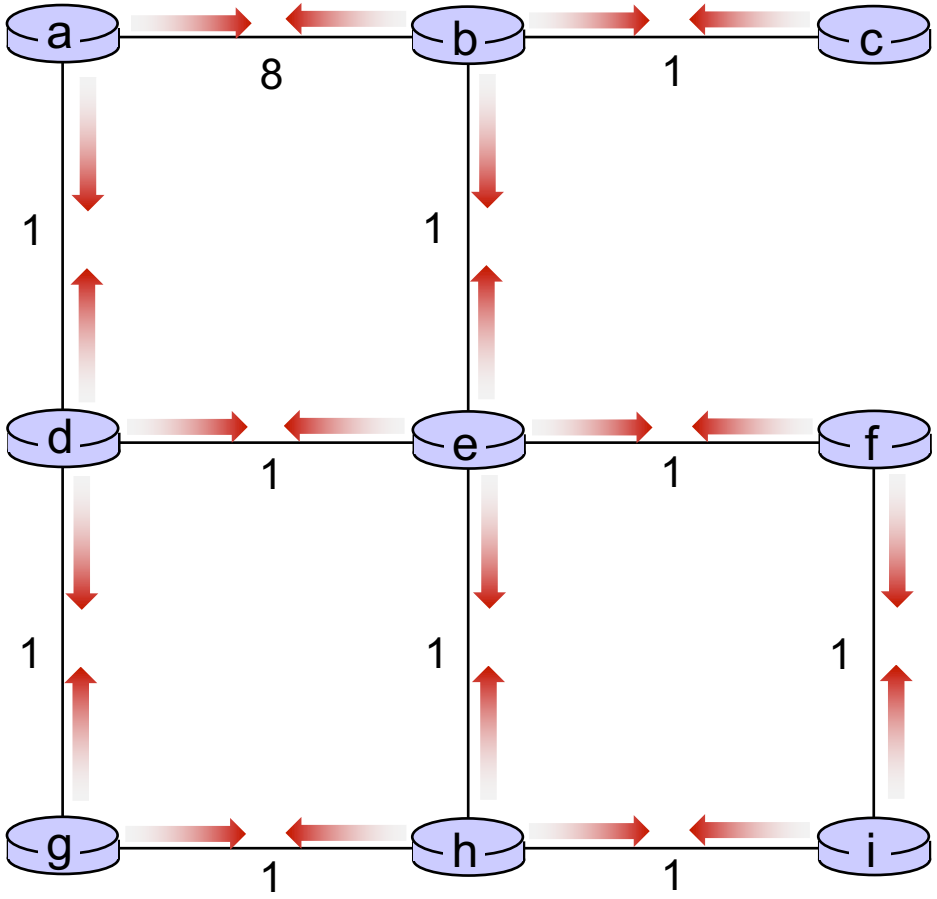


# Distance Vector Example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

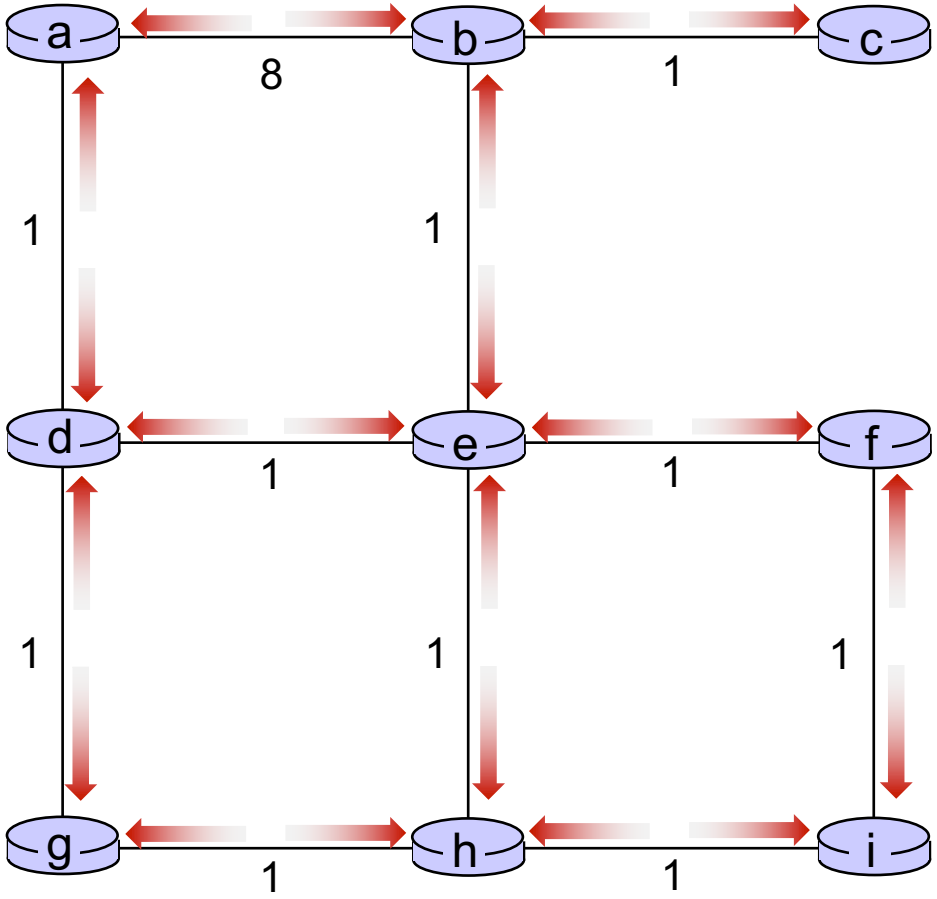


# Distance Vector Example: iteration

t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

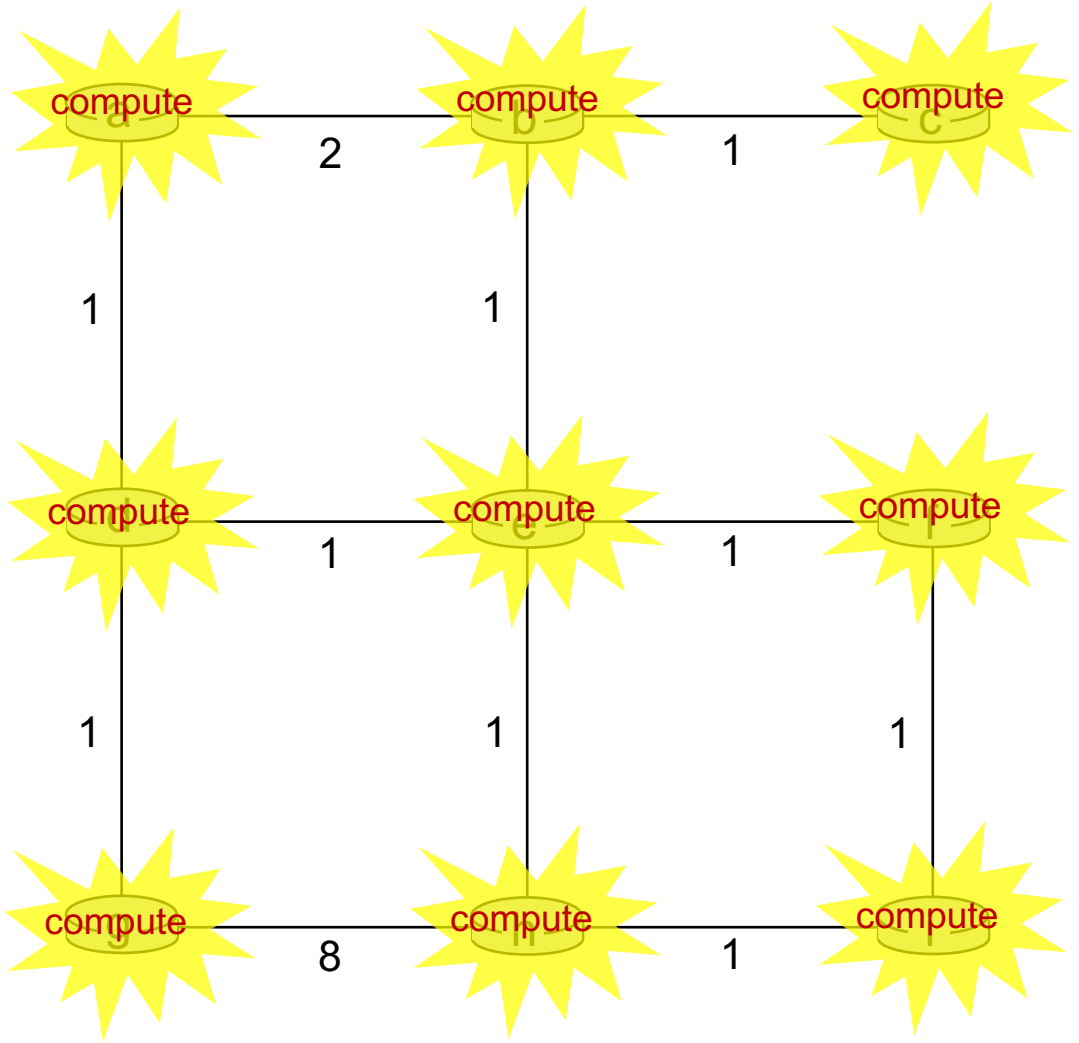


# Distance Vector Example: iteration

t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

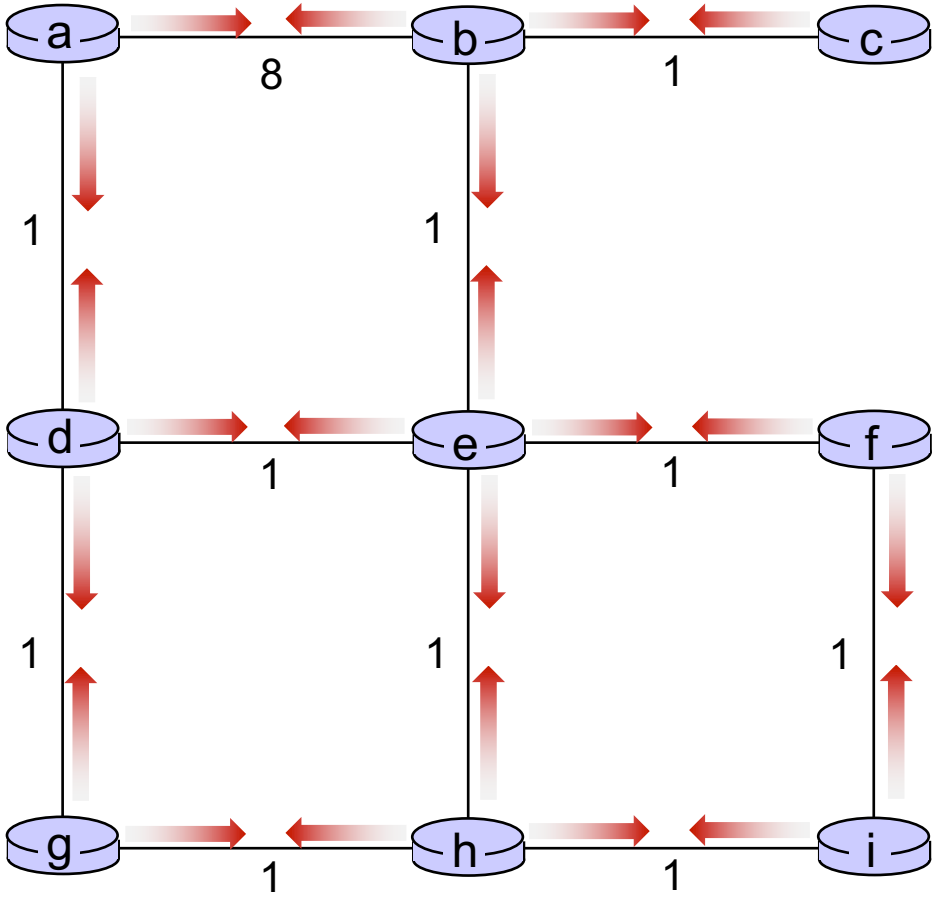


# Distance Vector Example: iteration

t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



# Distance Vector Example: iteration

…. and so on

Let’s next take a look at the iterative *computations* at nodes

# Distance Vector Example: computation

t=1

- b receives DVs from a, c, e

**DV in a:**

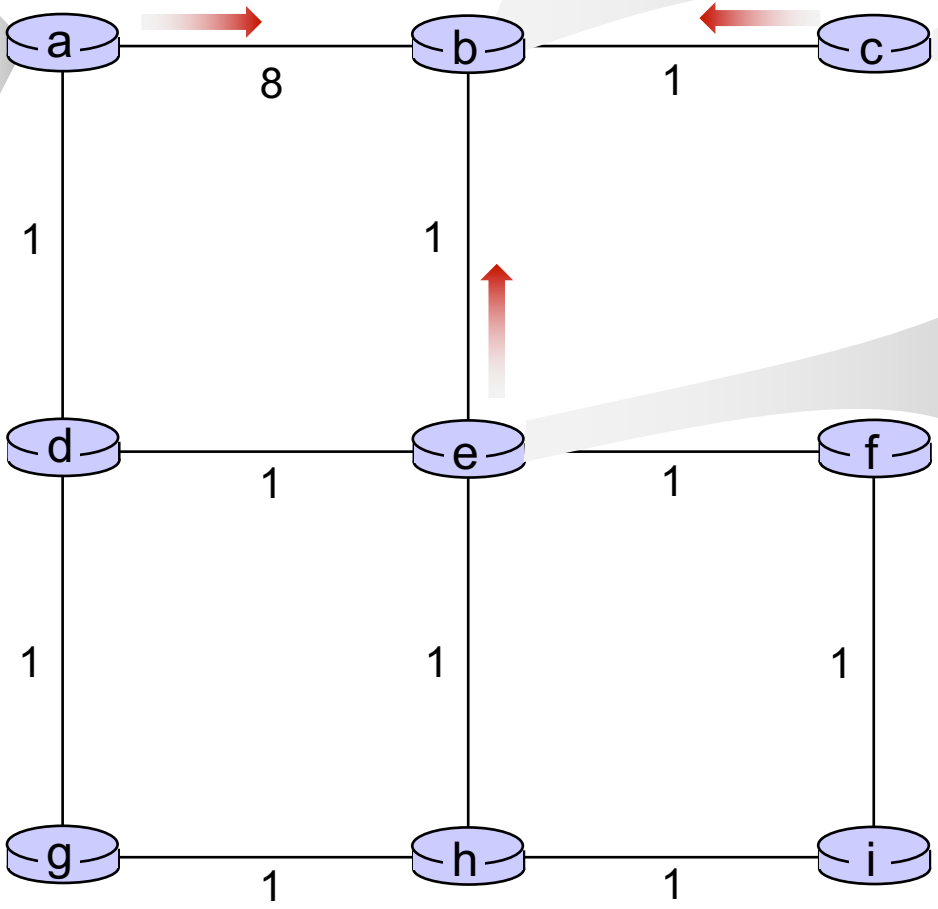
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

**DV in b:**

$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

**DV in c:**

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$



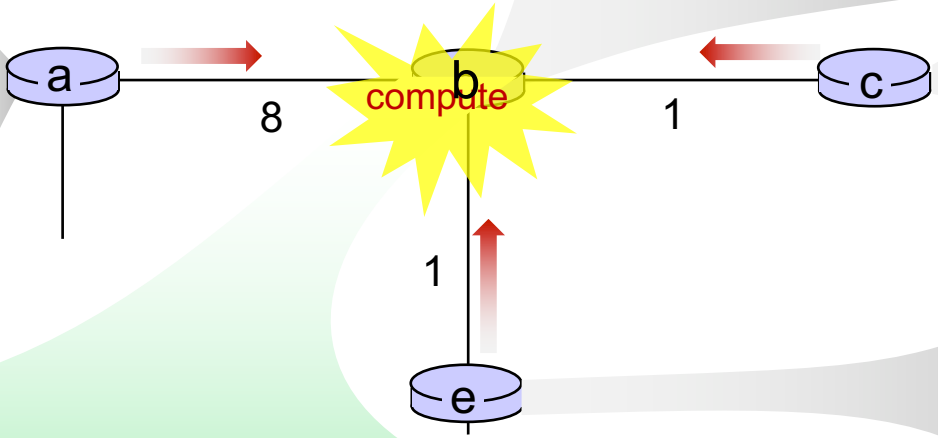

**DV in e:**

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

# Distance Vector Example: computation

t=1

- b receives DVs from a, c, e, computes:




**DV in a:**

$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

**DV in b:**

$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

**DV in c:**

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

**DV in e:**

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

$D_b(a) = \min\{c_{b,a}+D_a(a), c_{b,c}+D_c(a), c_{b,e}+D_e(a)\} = \min\{8,\infty,\infty\} = 8$

$D_b(c) = \min\{c_{b,a}+D_a(c), c_{b,c}+D_c(c), c_{b,e}+D_e(c)\} = \min\{\infty,1,\infty\} = 1$

$D_b(d) = \min\{c_{b,a}+D_a(d), c_{b,c}+D_c(d), c_{b,e}+D_e(d)\} = \min\{9,2,\infty\} = 2$

$D_b(e) = \min\{c_{b,a}+D_a(e), c_{b,c}+D_c(e), c_{b,e}+D_e(e)\} = \min\{\infty,\infty,1\} = 1$

$D_b(f) = \min\{c_{b,a}+D_a(f), c_{b,c}+D_c(f), c_{b,e}+D_e(f)\} = \min\{\infty,\infty,2\} = 2$

$D_b(g) = \min\{c_{b,a}+D_a(g), c_{b,c}+D_c(g), c_{b,e}+D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty$

$D_b(h) = \min\{c_{b,a}+D_a(h), c_{b,c}+D_c(h), c_{b,e}+D_e(h)\} = \min\{\infty, \infty, 2\} = 2$

$D_b(i) = \min\{c_{b,a}+D_a(i), c_{b,c}+D_c(i), c_{b,e}+D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty$

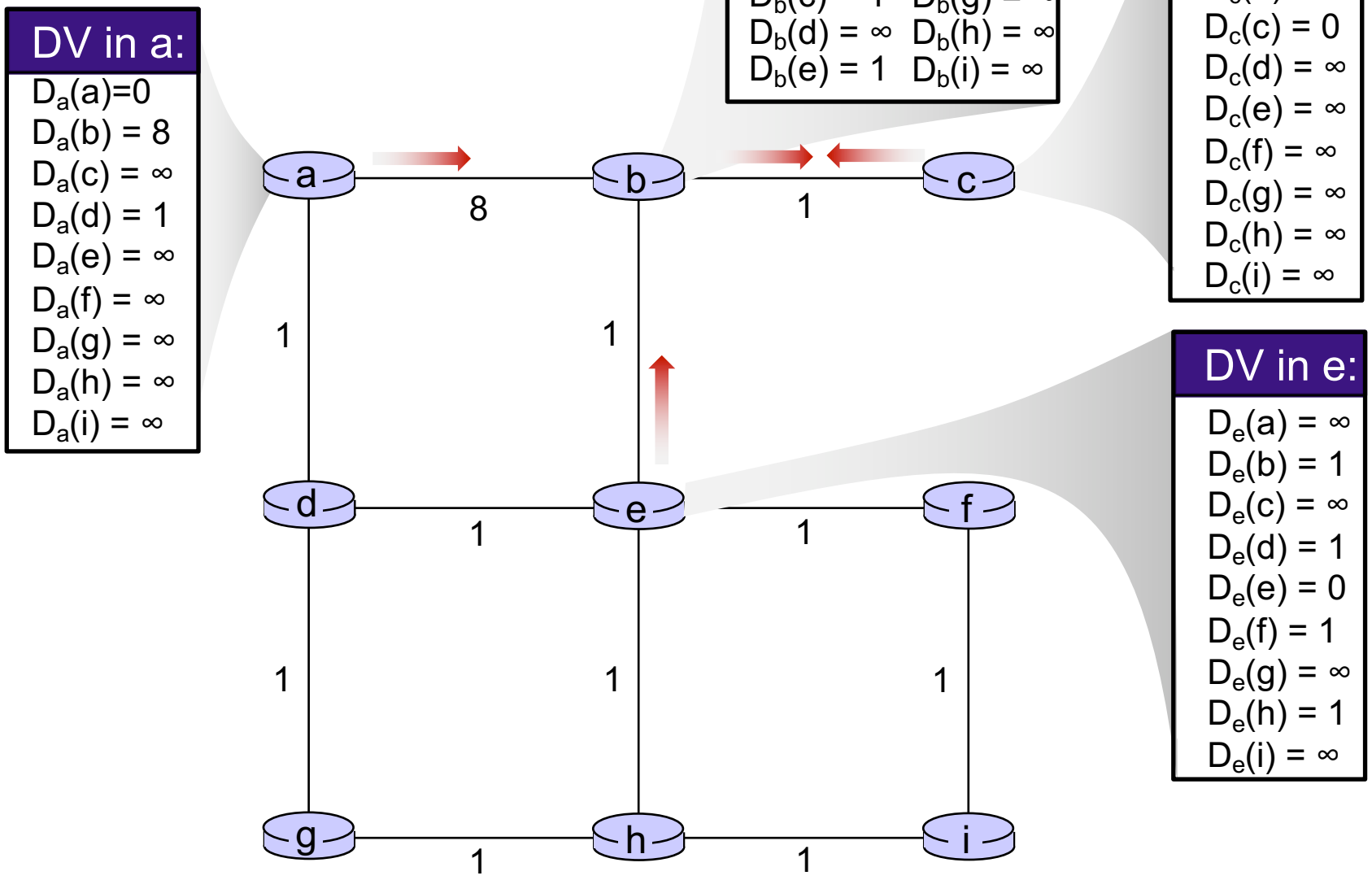
**DV in b:**

$D_b(a) = 8$ $D_b(f) = 2$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = 2$ $D_b(h) = 2$
$D_b(e) = 1$ $D_b(i) = \infty$

# Distance Vector Example: computation

t=1

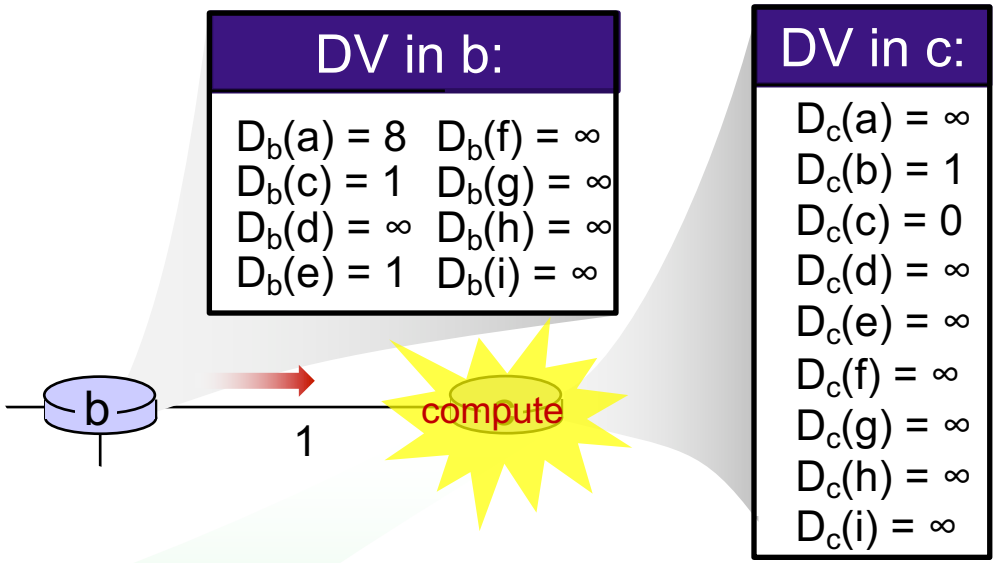
- c receives DVs from b



# Distance Vector Example: computation

t=1

- c receives DVs from b computes:




$D_c(a) = \min\{c_{c,b}+D_b(a)\} = 1 + 8 = 9$

$D_c(b) = \min\{c_{c,b}+D_b(b)\} = 1 + 0 = 1$

$D_c(d) = \min\{c_{c,b}+D_b(d)\} = 1+ \infty = \infty$

$D_c(e) = \min\{c_{c,b}+D_b(e)\} = 1 + 1 = 2$

$D_c(f) = \min\{c_{c,b}+D_b(f)\} = 1+ \infty = \infty$

$D_c(g) = \min\{c_{c,b}+D_b(g)\} = 1+ \infty =$

$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$

$D_c(i) = \min\{c_{c,b}+D_b(i)\} = 1+ \infty = \infty$

**DV in c:**

$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

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

# Distance Vector Example: computation

t=1

- e receives DVs from b, d, f, h

**DV in b:**

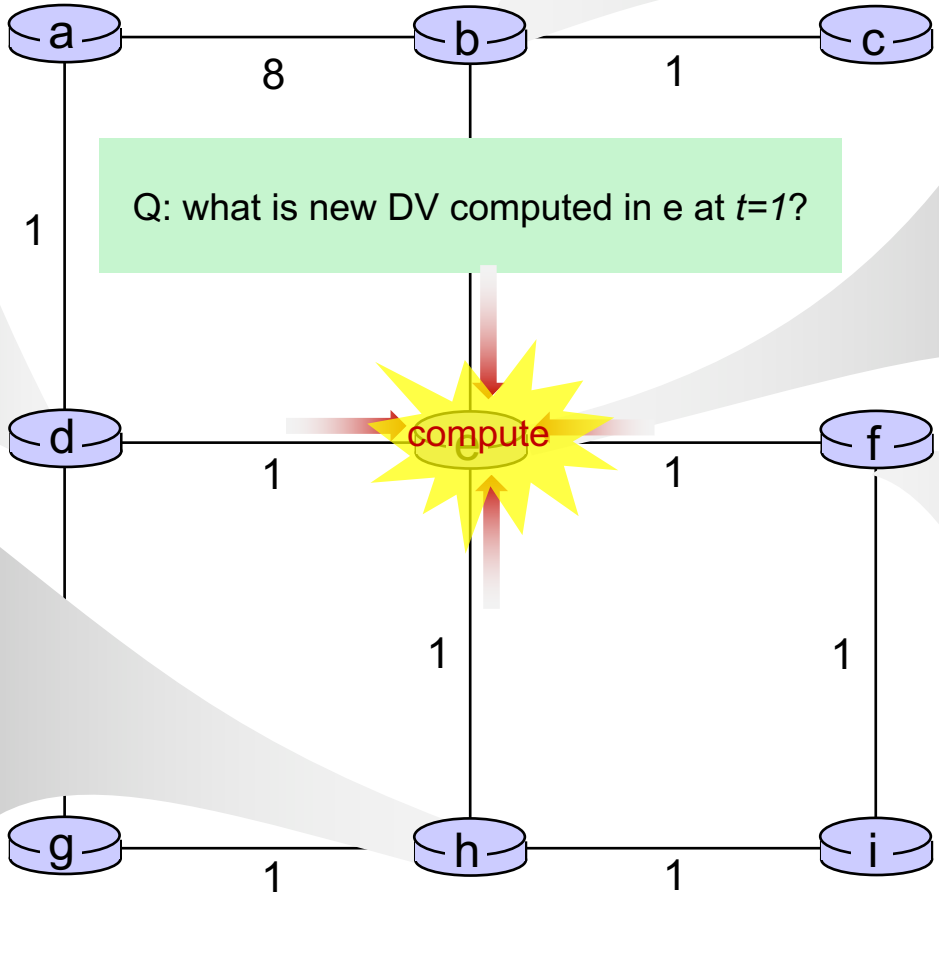
$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

**DV in d:**

$D_c(a) = 1$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = 0$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = \infty$
$D_c(i) = \infty$

**DV in e:**

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$




**DV in h:**

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = 0$
$D_c(i) = 1$

**DV in f:**

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = 0$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = 1$



UCCS

# Distance Vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

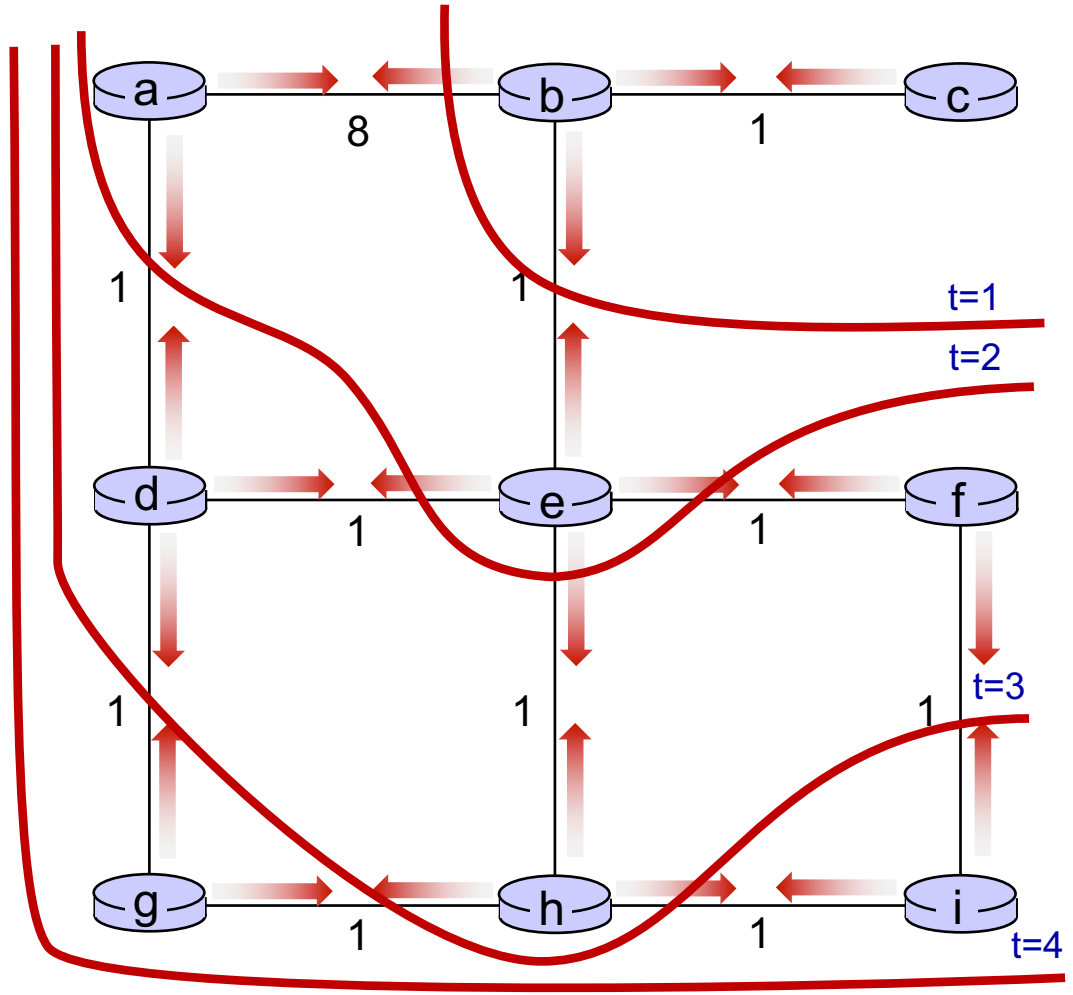
t=0 c's state at t=0 is at c only

t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b

t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well

t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at d, f, h

t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at g, i



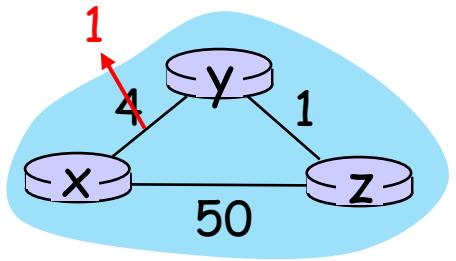



UCCS

# Distance vector: link cost changes

link cost changes:

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



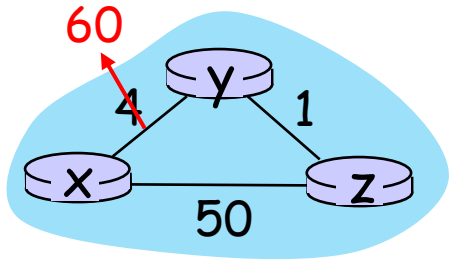

"good news travels fast"

$t_0$ : *y* detects link-cost change, updates its DV, informs its neighbors.

$t_1$ : *z* receives update from *y*, updates its DV, computes new least cost to *x* , sends its neighbors its DV.

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

# Distance vector: link cost changes




link cost changes:

- node detects local link cost change
- "bad news travels slow" – count-to-infinity problem:
  - *y* sees direct link to *x* has new cost 60, but z has said it has a path at cost of 5. So *y* computes "my new cost to x will be 6, via z); notifies *z* of new cost of 6 to *x.*
  - *z* learns that path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y), notifies *y* of new cost of 7 to *x.*
  - *y* learns that path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y), notifies *z* of new cost of 8 to *x.*
  - *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y), notifies *y* of new cost of 9 to *x.*

  …

# Comparison of LS and DV algorithms

## message complexity

LS: $n$ routers, $O(n^2)$ messages sent

DV: exchange between neighbors; convergence time varies

## speed of convergence

LS: $O(n^2)$ algorithm, $O(n^2)$ messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

## robustness: what happens if router malfunctions, or is compromised?

LS:

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

DV:

- DV router can advertise incorrect *path* cost ("I have a *really* low-cost path to everywhere"): *black-holing*
- each router's DV is used by others: error propagate thru network

# Control Plane Outline

- Routing Algorithm
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing Among ISPs: BGP
- SDN Control Plane
- Internet Control Message Protocol

# Making Routing Scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

… not true in practice

scale: billions of destinations:

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

administrative autonomy:

- Internet: a 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")

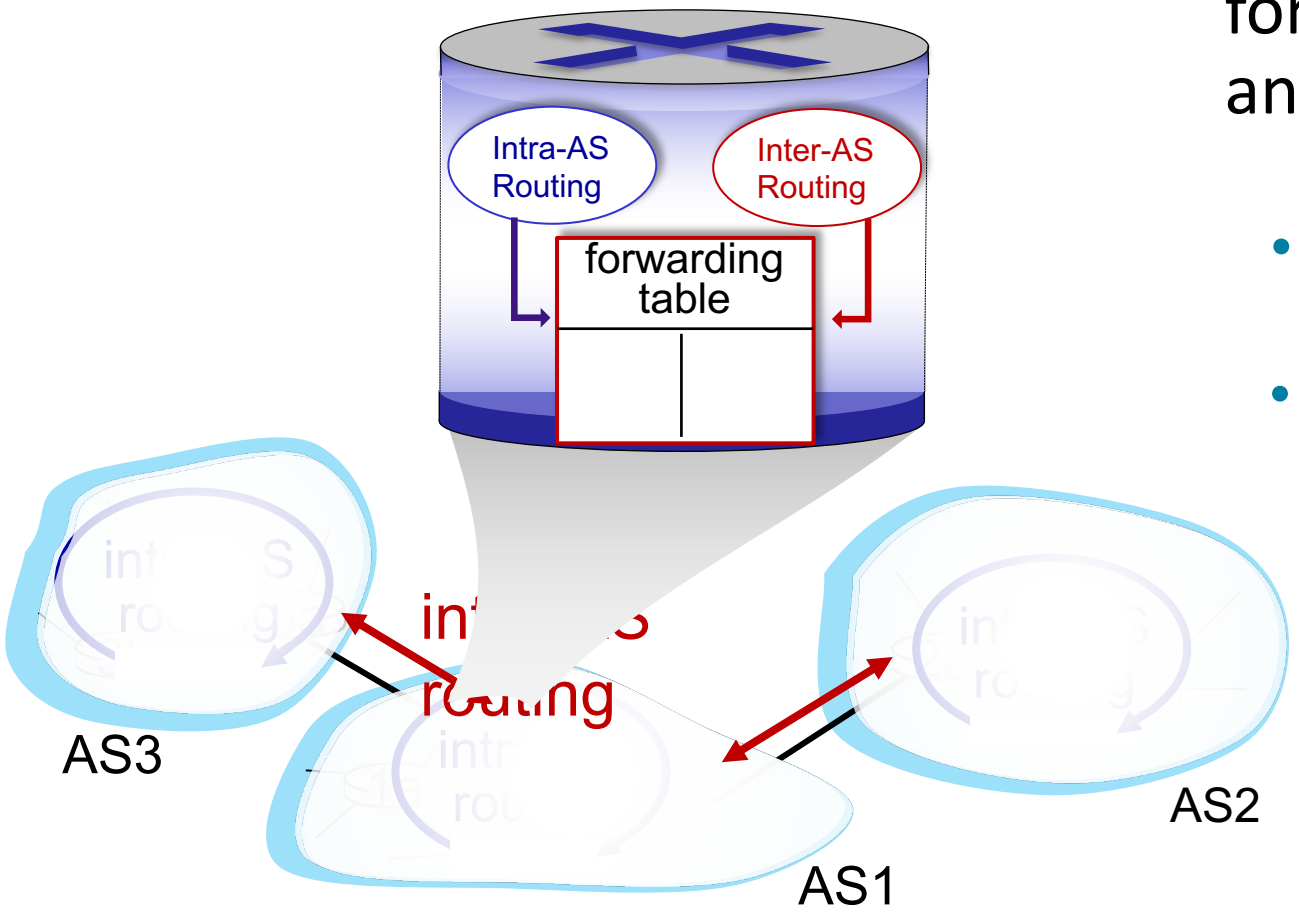
intra-AS (aka "intra-domain"): routing among routers *within same AS ("network")*

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

inter-AS (aka "inter-domain"): routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

# Interconnected ASes




forwarding table configured by intra- and inter-AS routing algorithms

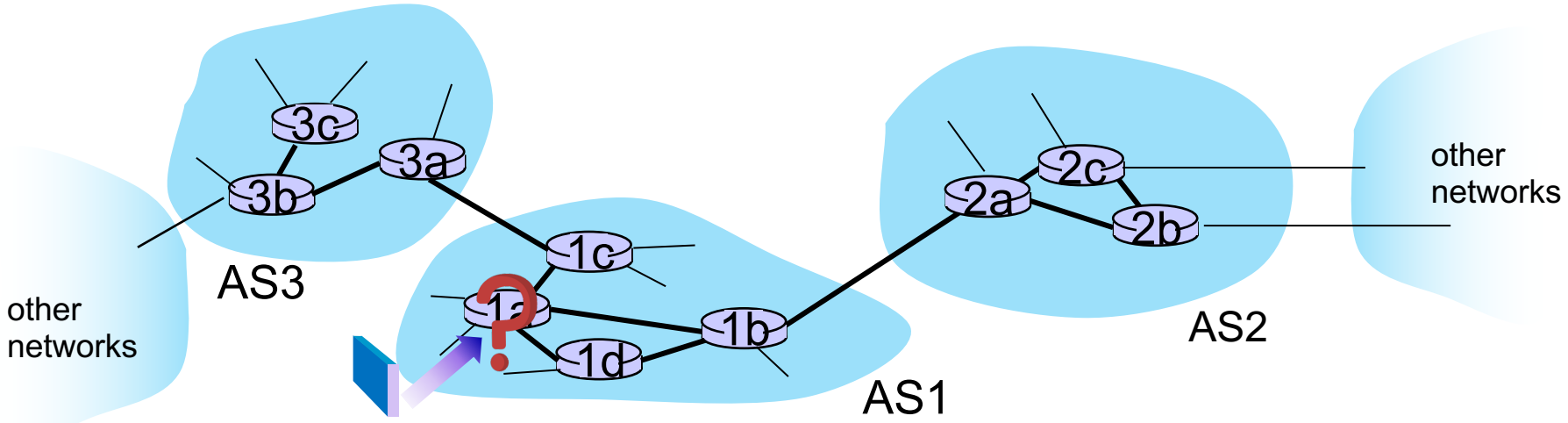
- intra-AS routing determine entries for destinations within AS
- inter-AS & intra-AS determine entries for external destinations

# Inter-AS Routing: a role in intradomain forwarding

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

AS1 inter-domain routing must:

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



# Intra-AS Routing: routing within an AS

most common intra-AS routing protocols:

- RIP: Routing Information Protocol [RFC 1723]
  - classic DV: DVs exchanged every 30 secs
  - no longer widely used
- EIGRP: Enhanced Interior Gateway Routing Protocol
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- OSPF: Open Shortest Path First [RFC 2328]
  - link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

# OSPF (Open Shortest Path First) Routing

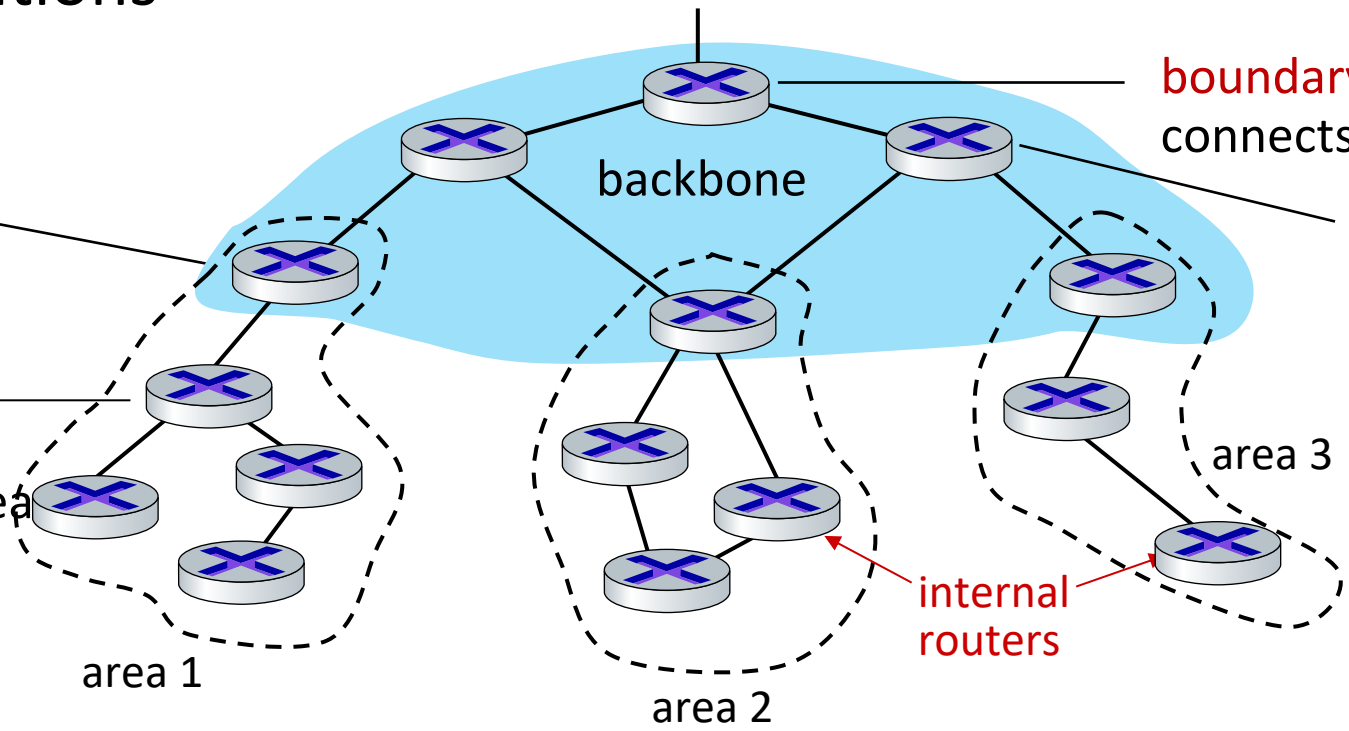
- "open": publicly available
- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra's algorithm to compute forwarding table
- *security:* all OSPF messages authenticated (to prevent malicious intrusion)

# Hierarchical OSPF

- two-level hierarchy: local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations

area border routers:
"summarize" distances to destinations in own area, advertise in backbone

local routers:
- flood LS in area only
- compute routing within area
- forward packets to outside via area border router



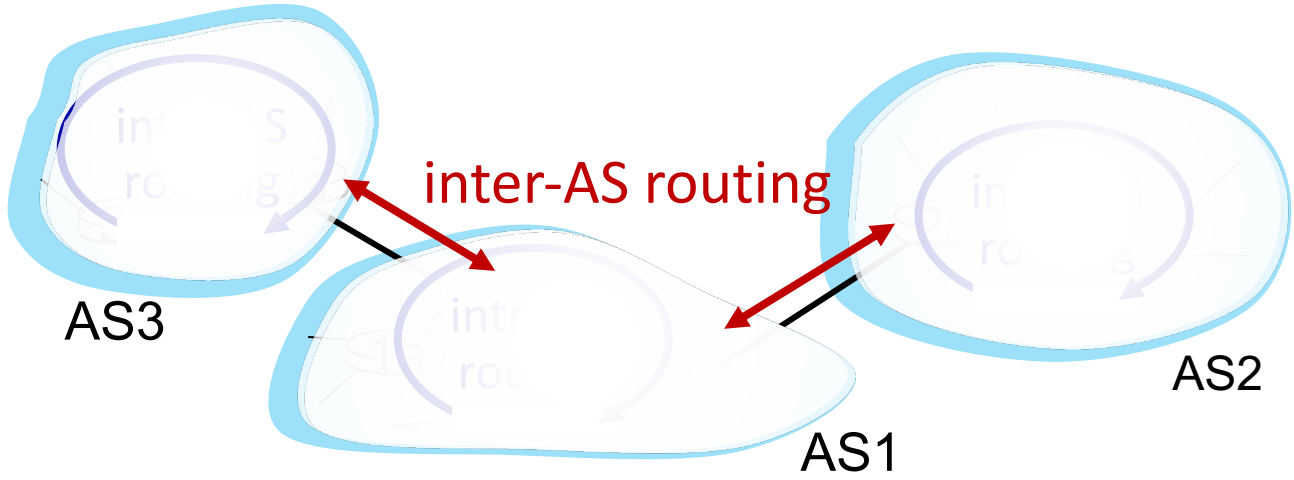

boundary router:
connects to other ASes

backbone router:
runs OSPF limited to backbone

# Control Plane Outline

- Routing Algorithm
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing Among ISPs: BGP
- SDN Control Plane
- Internet Control Message Protocol

# Interconnected ASes



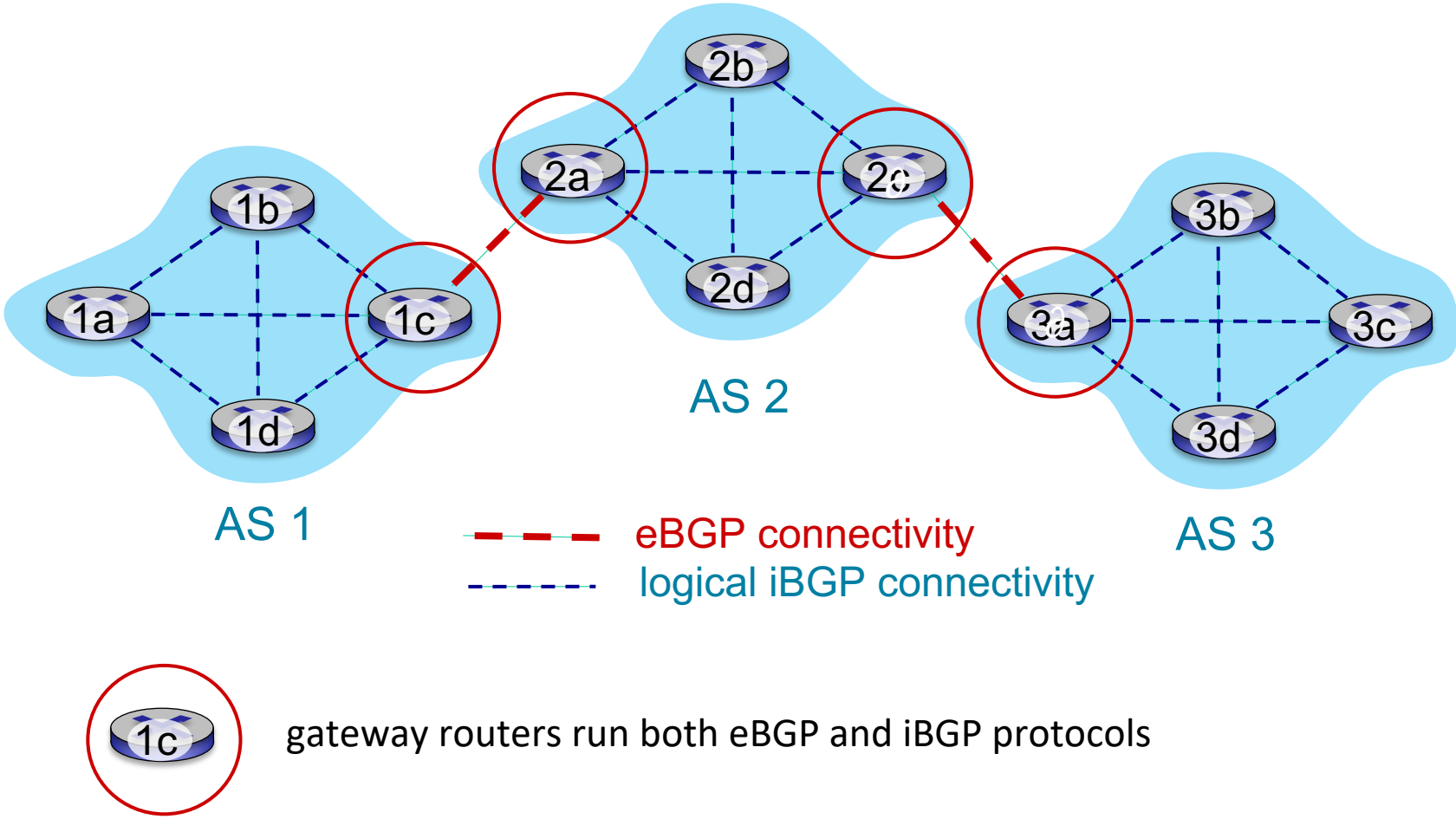

intra-AS (aka "intra-domain"): routing among routers *within same AS ("network")*

inter-AS (aka "inter-domain"): routing *among* AS'es

# Internet Inter-AS Routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *"I am here, here is who I can reach, and how"*
- BGP provides each AS a means to:
  - obtain destination network reachability info from neighboring ASes (eBGP)
  - determine routes to other networks based on reachability information and *policy*
  - propagate reachability information to all AS-internal routers (iBGP)
  - advertise (to neighboring networks) destination reachability info

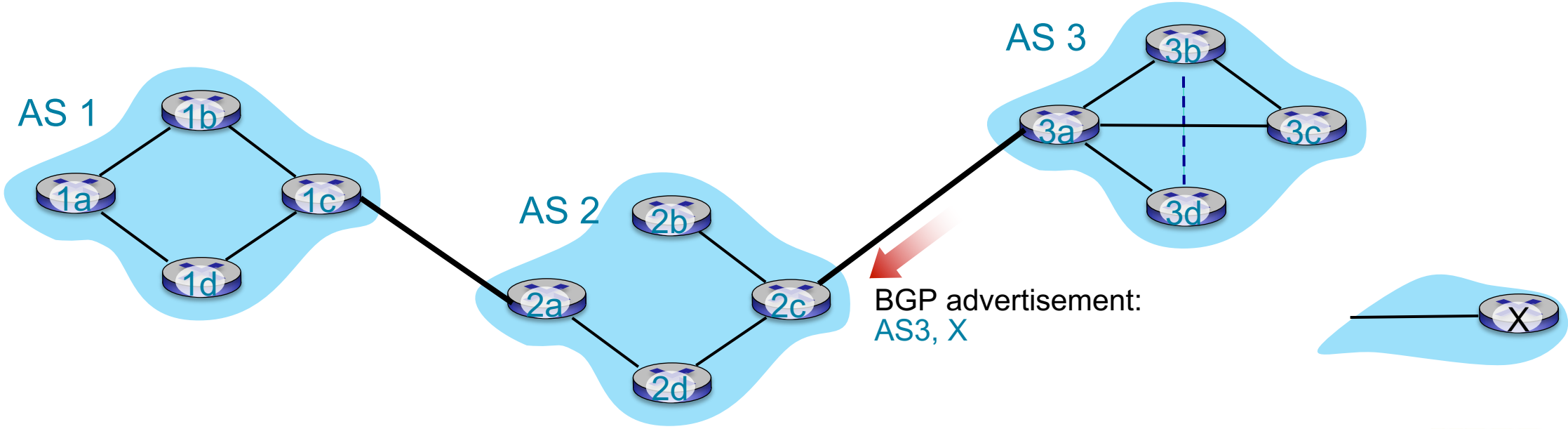
# eBGP, iBGP Connections



# 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 3a advertises path AS3,X to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



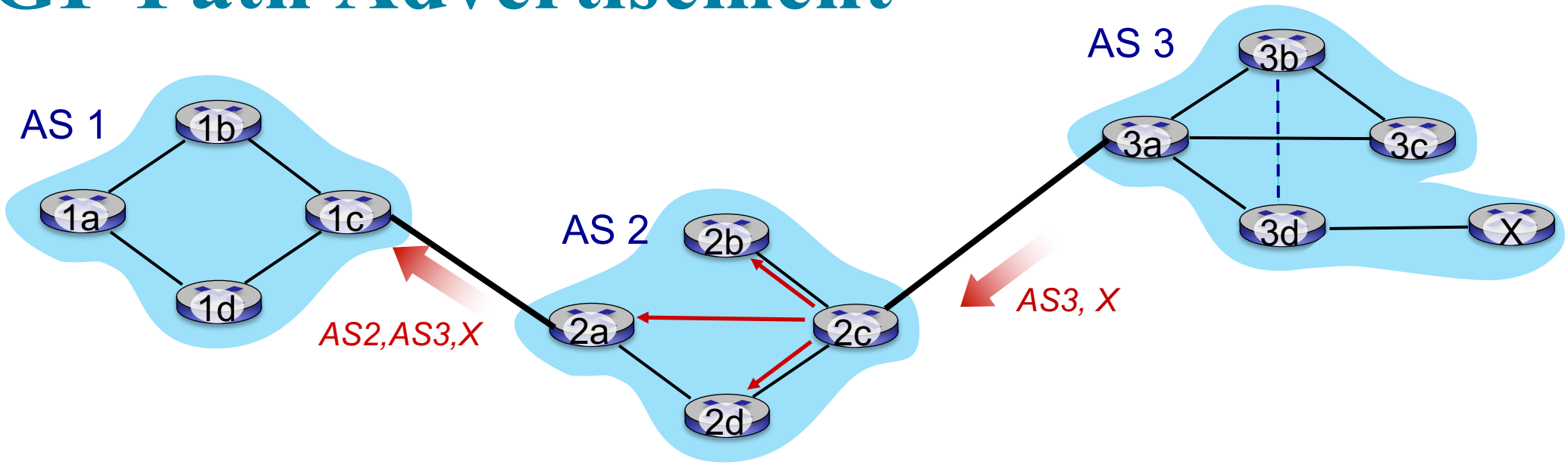
# BGP Protocol Messages

- BGP messages exchanged between peers over TCP connection
- BGP messages [RFC 4371]:
  - 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

# Path Attributes and BGP Routes

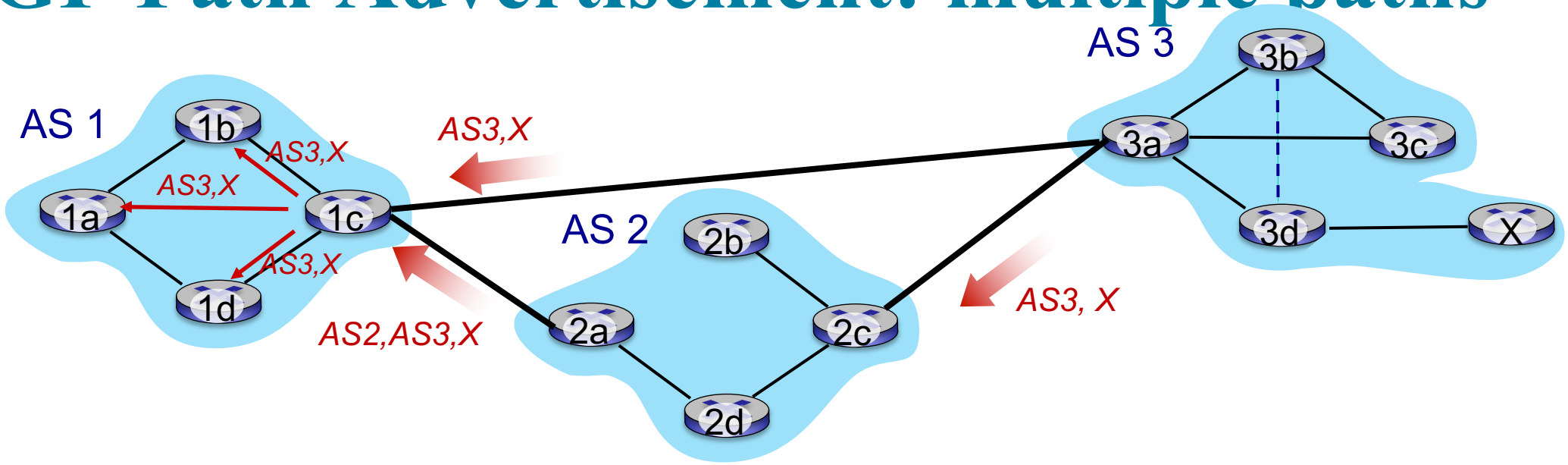
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- policy-based routing:
  - gateway receiving 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

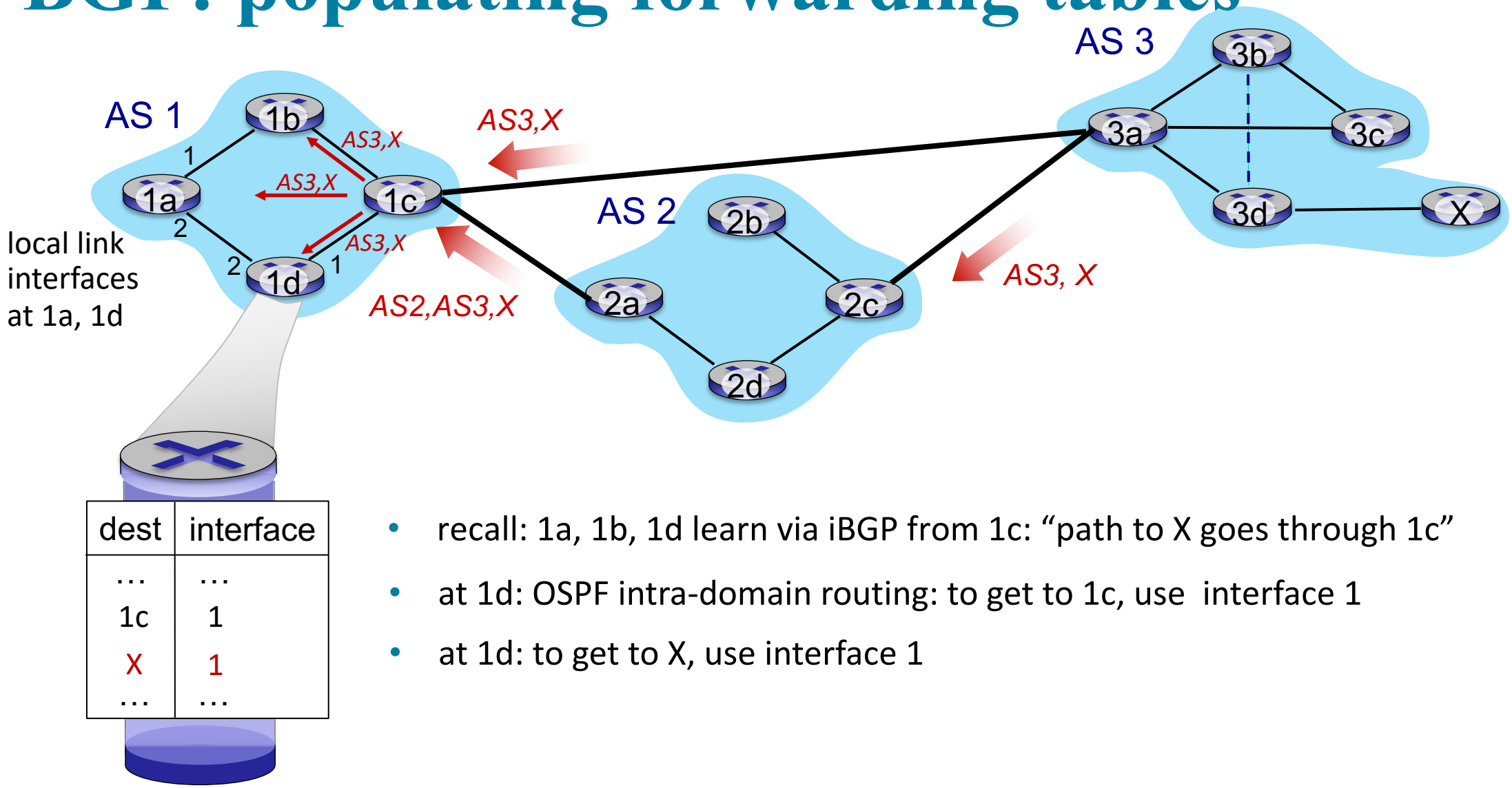
# BGP Path Advertisement: multiple paths




gateway router may learn about multiple paths to destination:

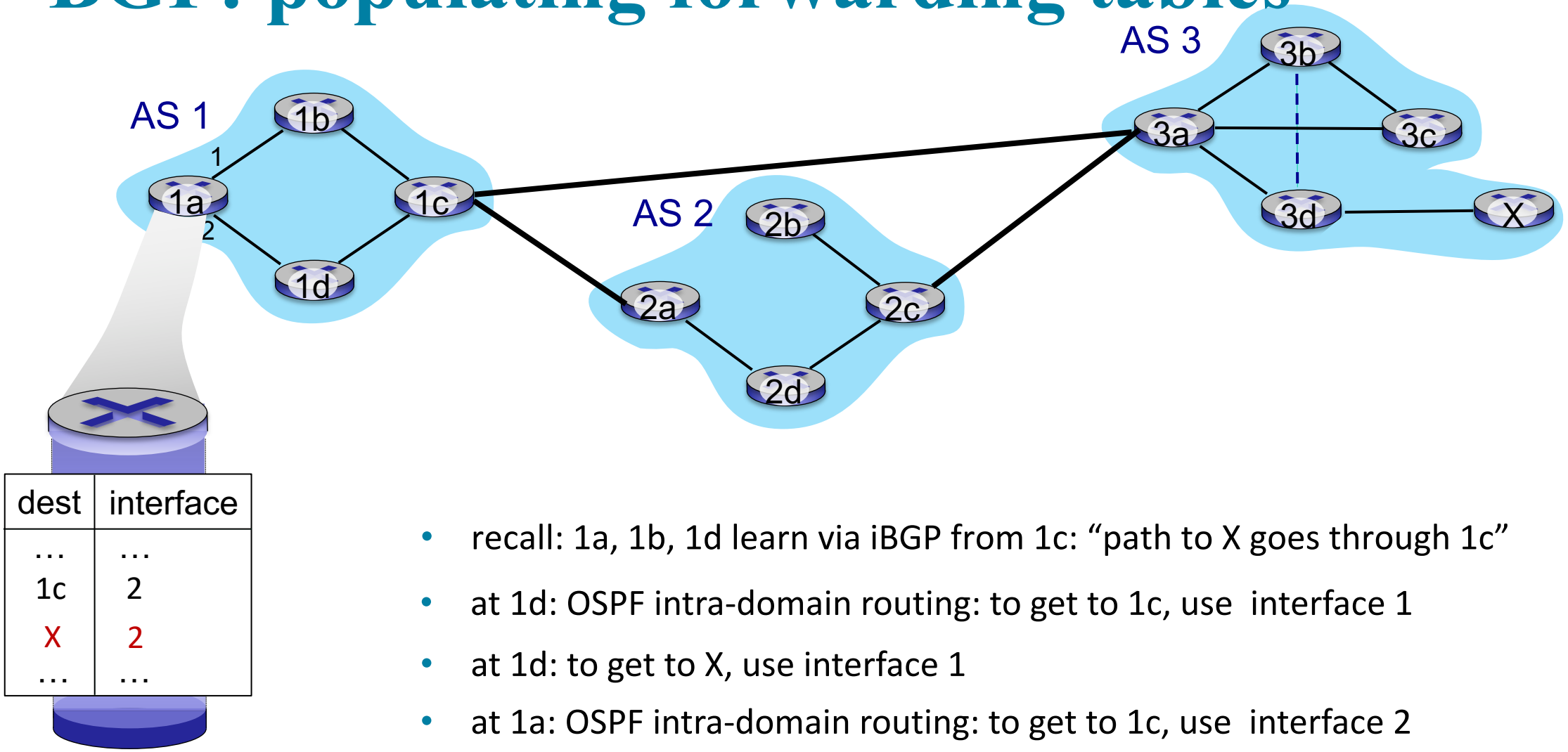
- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- based on *policy,* AS1 gateway router 1c chooses path *AS3,X* and advertises path within AS1 via iBGP

# BGP: populating forwarding tables



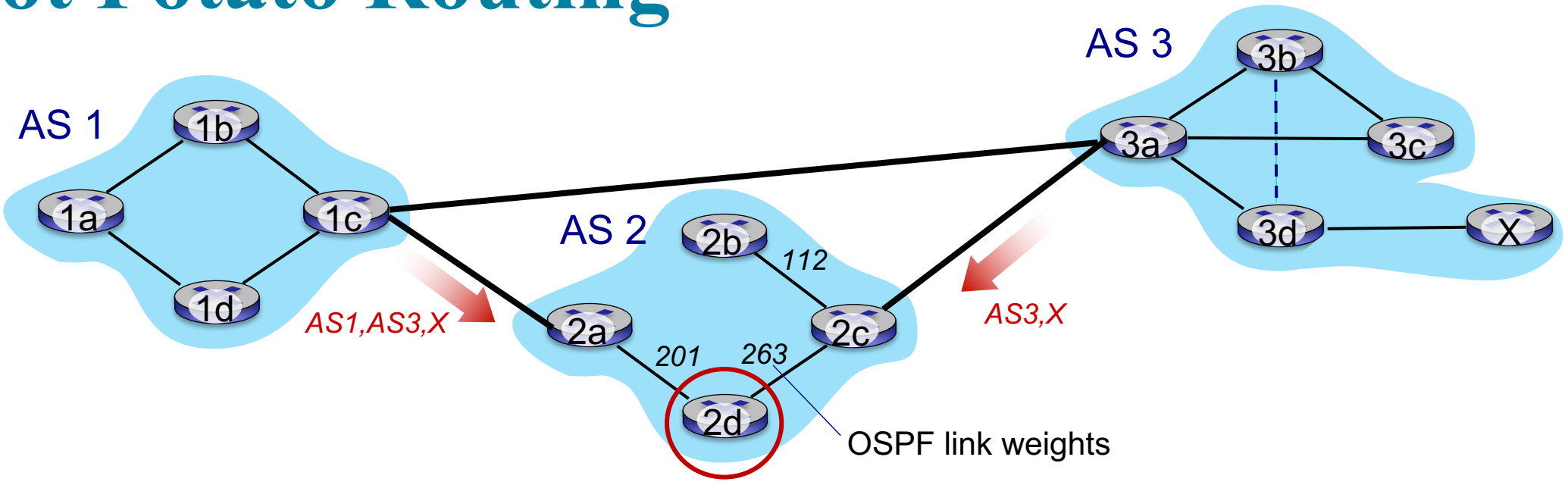

- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

# BGP: populating forwarding tables



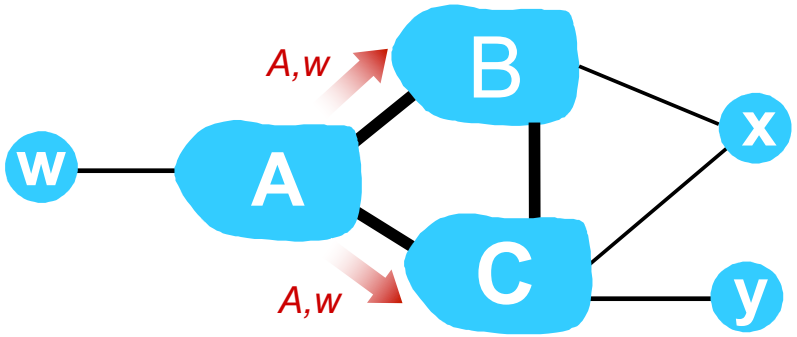

- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2

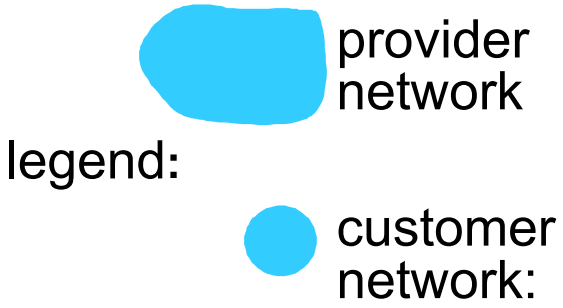
# Hot Potato Routing




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

# BGP: achieving policy via advertisements

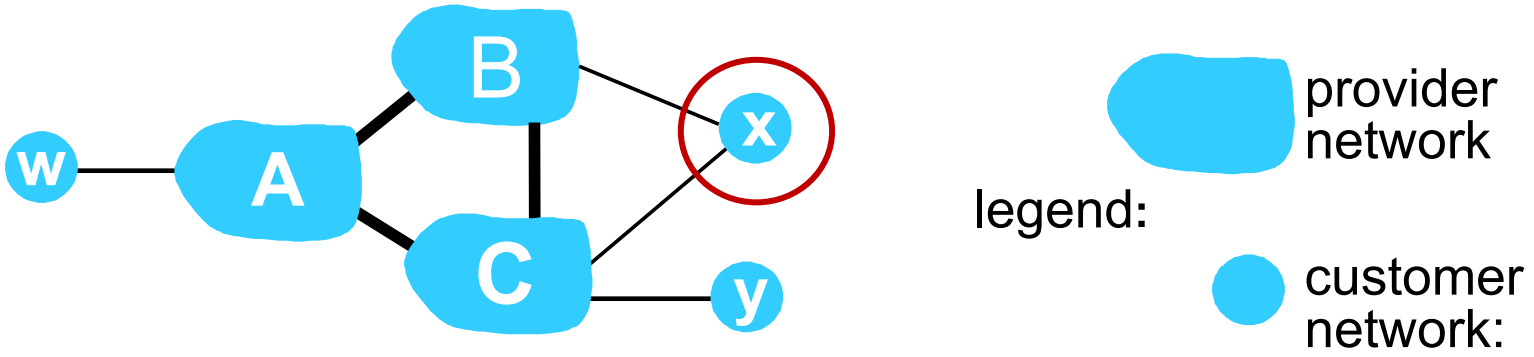



ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical "real world" policy)

- 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 (more)




ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical "real world" policy)

- 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

# 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

# Why Different Intra-, Inter-AS Routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

scale:

- hierarchical routing saves table size, reduced update traffic

performance:

- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

# Control Plane Outline

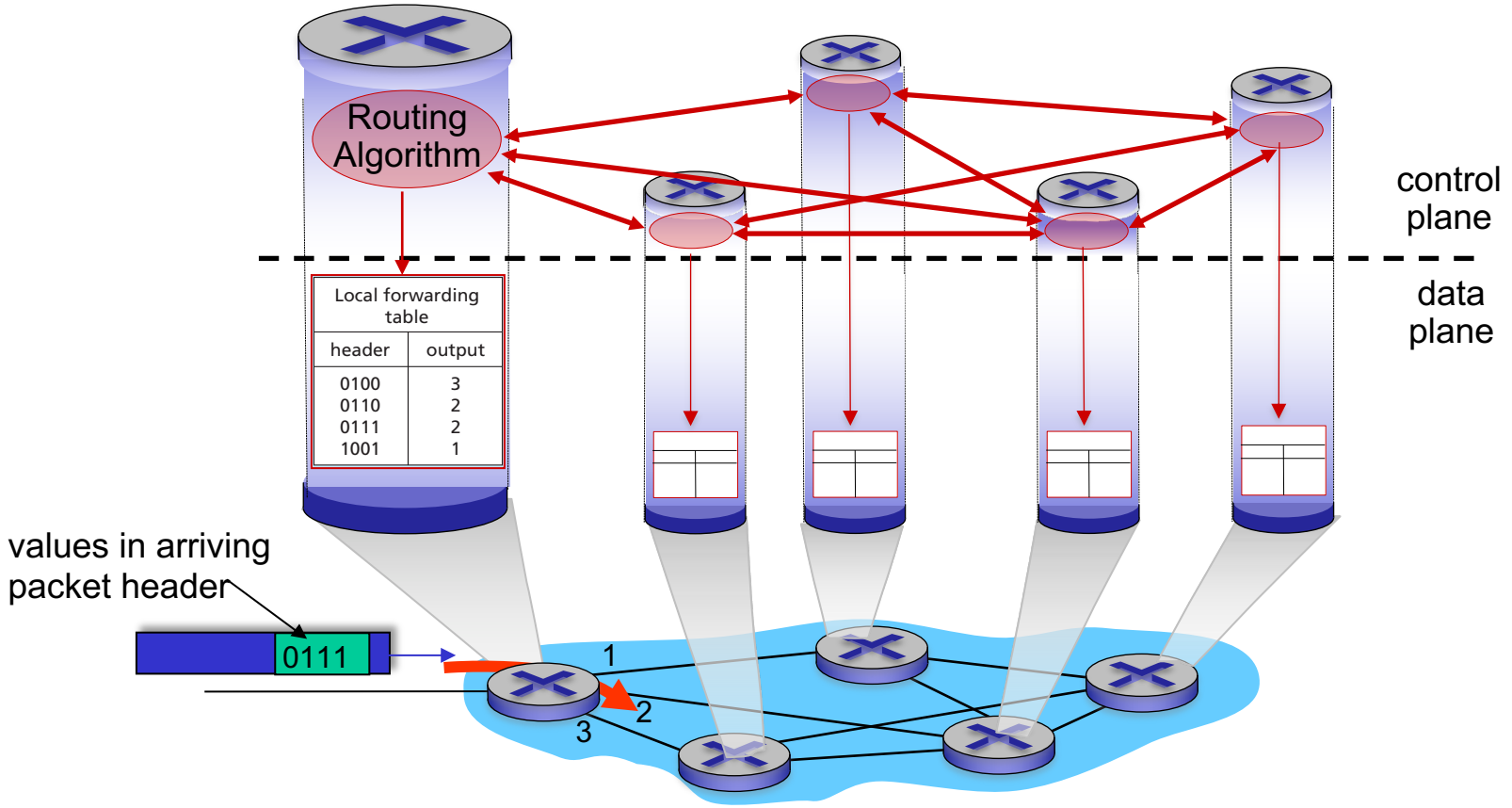
- Routing Algorithm
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing Among ISPs: BGP
- SDN Control Plane
- Internet Control Message Protocol

# Software Defined Networking (SDN)

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

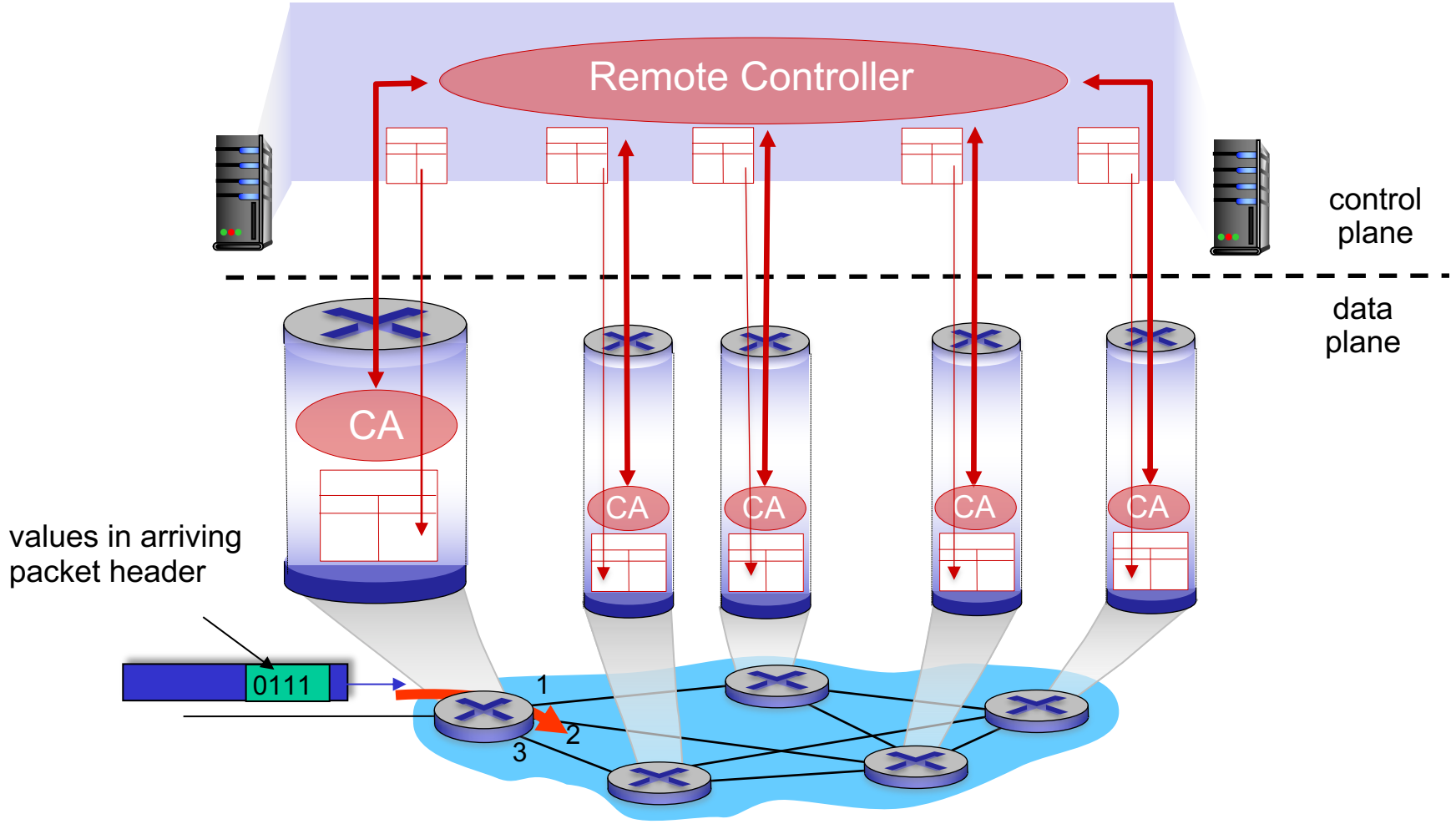
# Per-router Control Plane

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

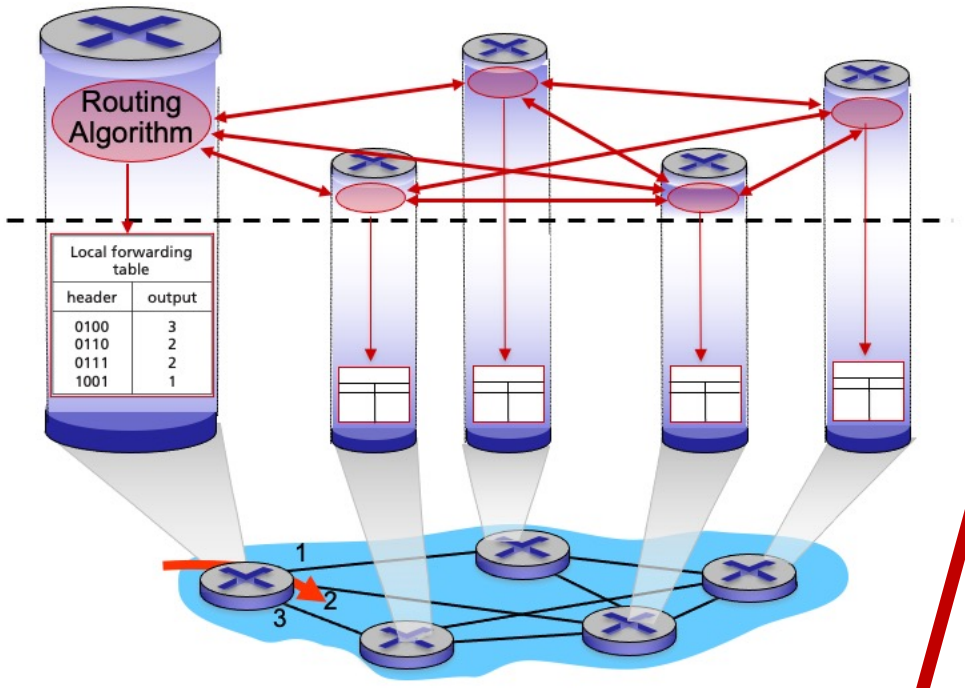


# Software-Defined Networking (SDN) Control Plane

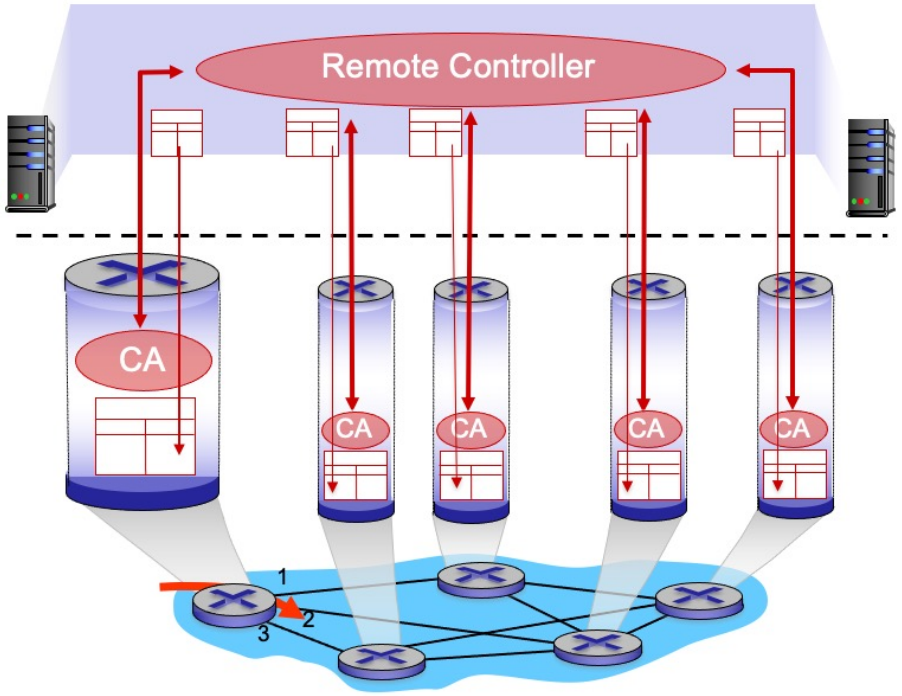
Remote controller computes, installs forwarding tables in routers



# Per-router control plane




# SDN control plane



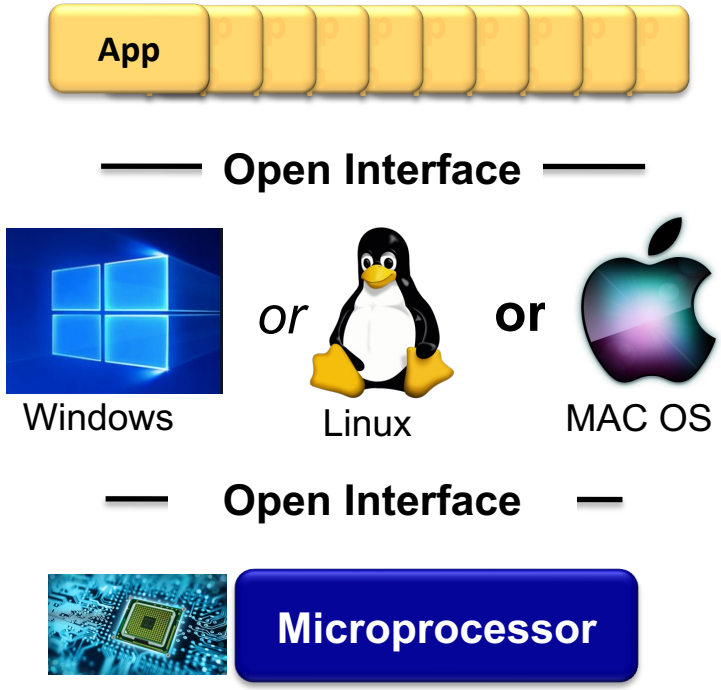
# Software Defined Networking (SDN)

*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
  - foster innovation: let 1000 flowers bloom

# SDN Analogy: mainframe to PC revolution

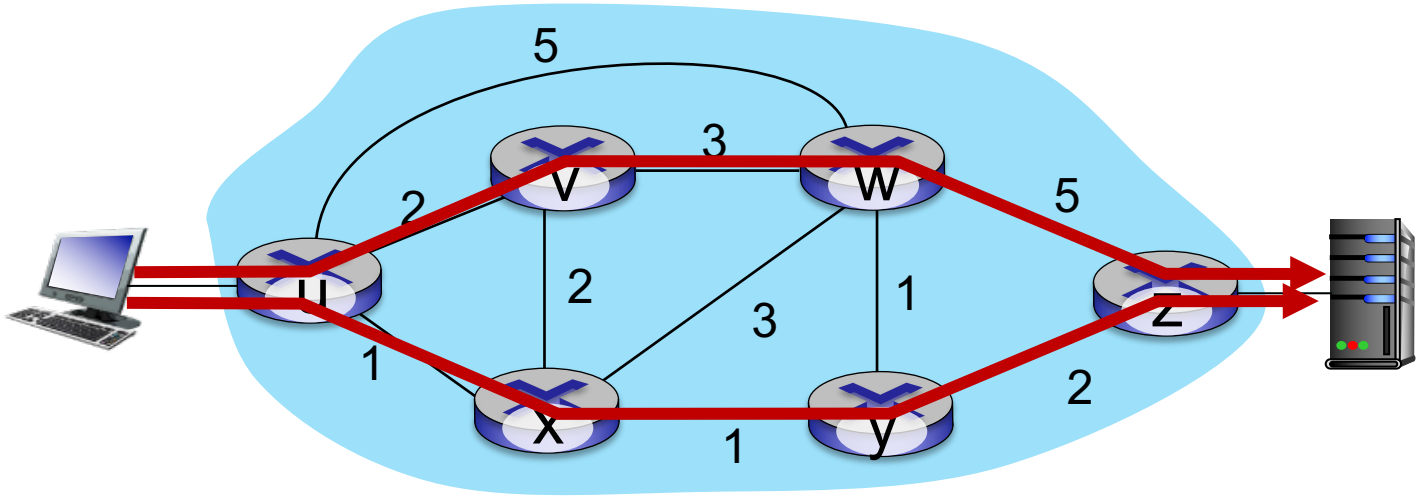

Vertically integrated
Closed, proprietary
Slow innovation
Small industry




Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

# Traffic Engineering: difficult with traditional routing

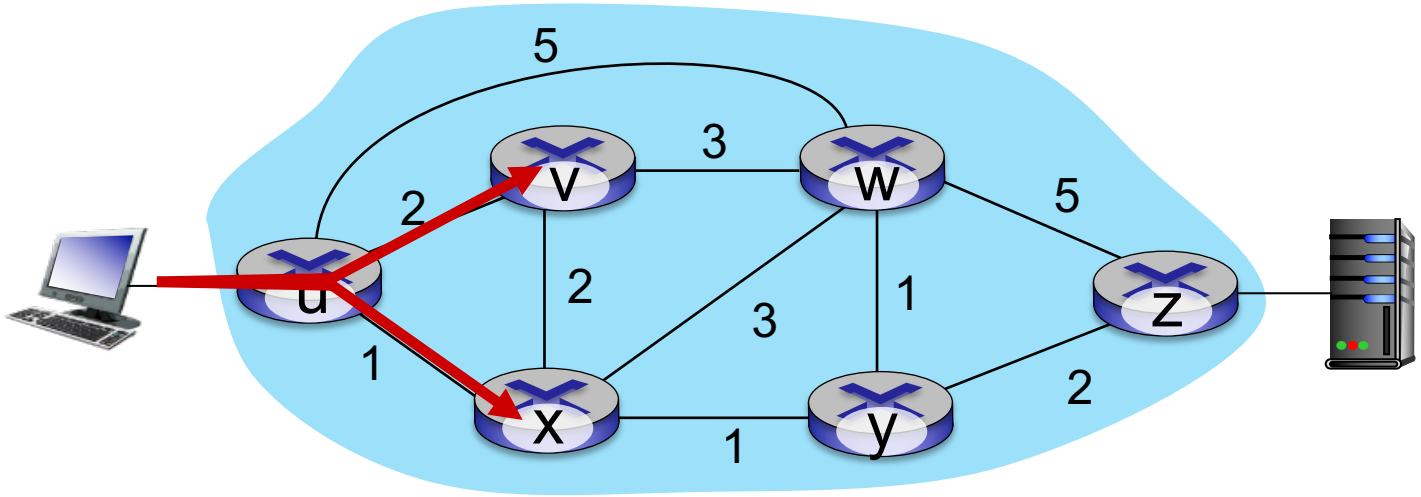



*Q:* what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

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

*link weights are only control "knobs": not much control!*

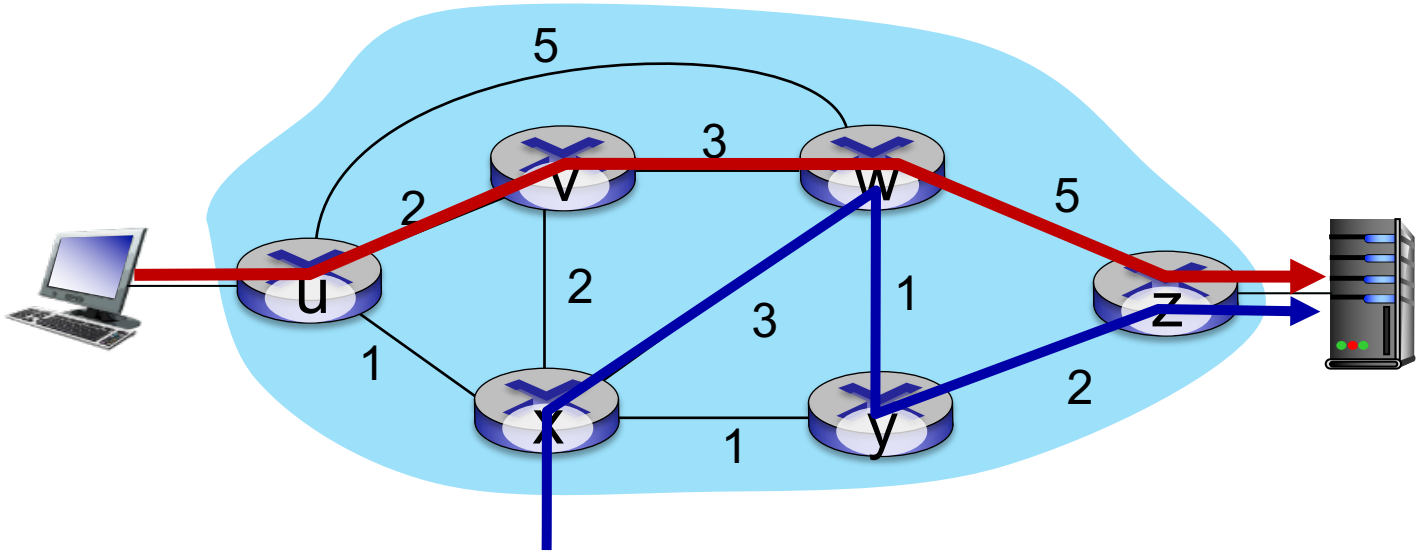
# Traffic Engineering: difficult with traditional routing




*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)

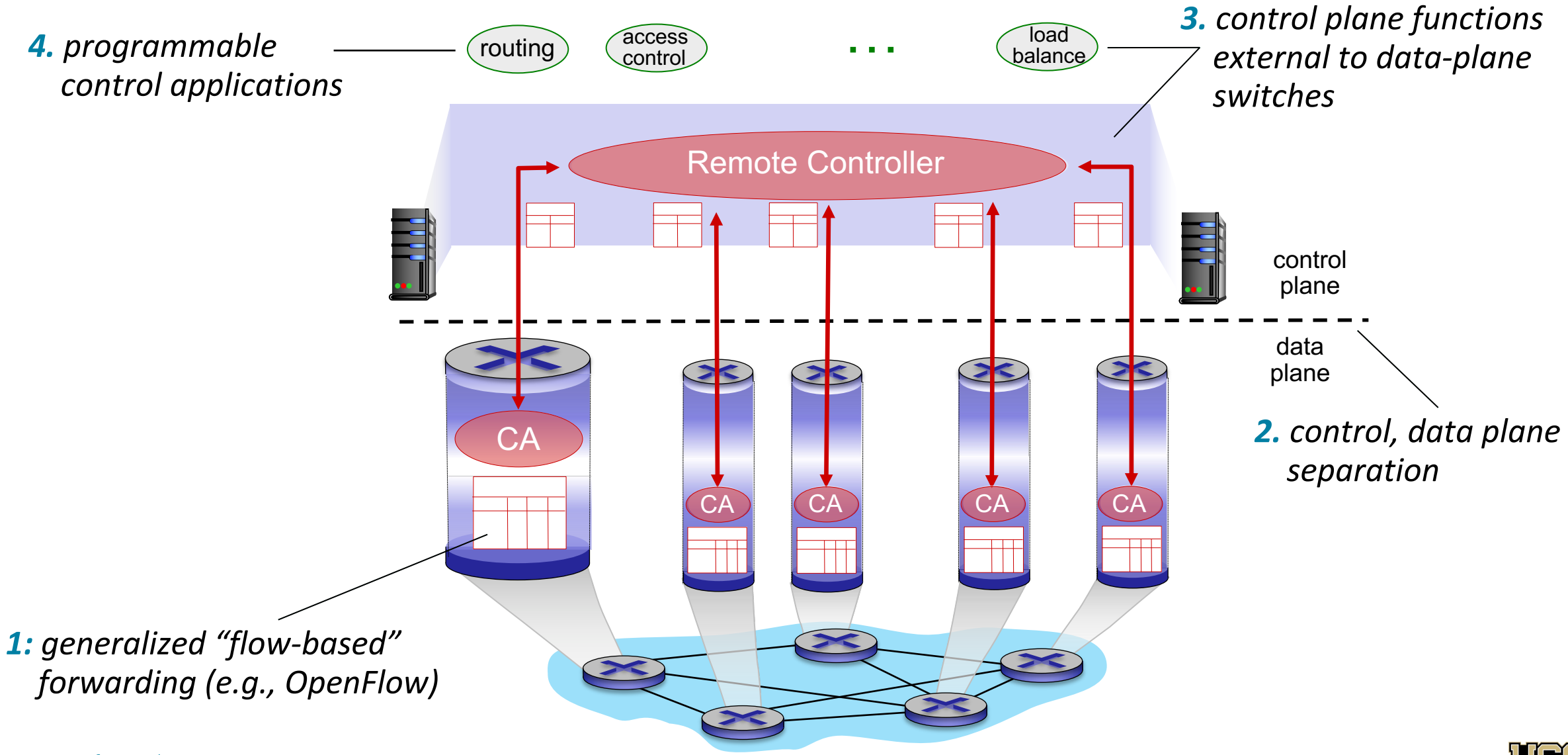
# Traffic Engineering: difficult with traditional routing




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

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

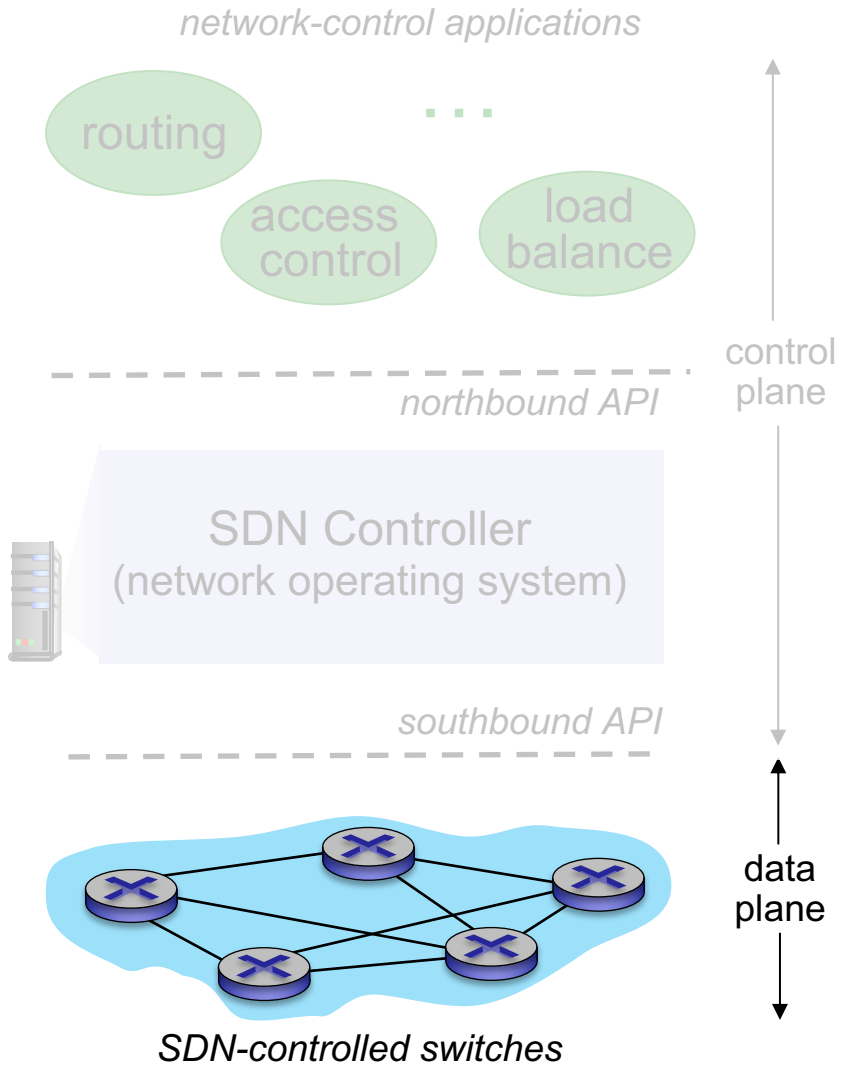
# Software Defined Networking (SDN)



# Software Defined Networking (SDN)

## Data-plane switches:

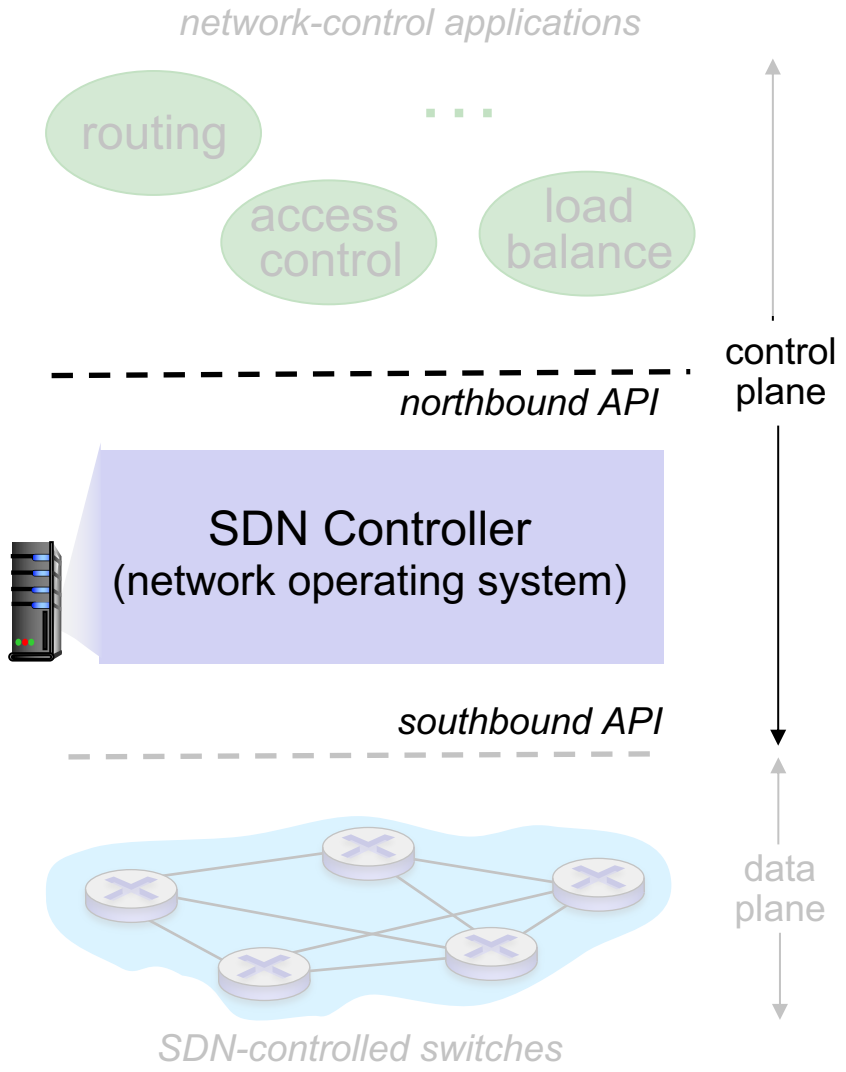
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with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



# Software Defined Networking (SDN)

## SDN controller (network OS):

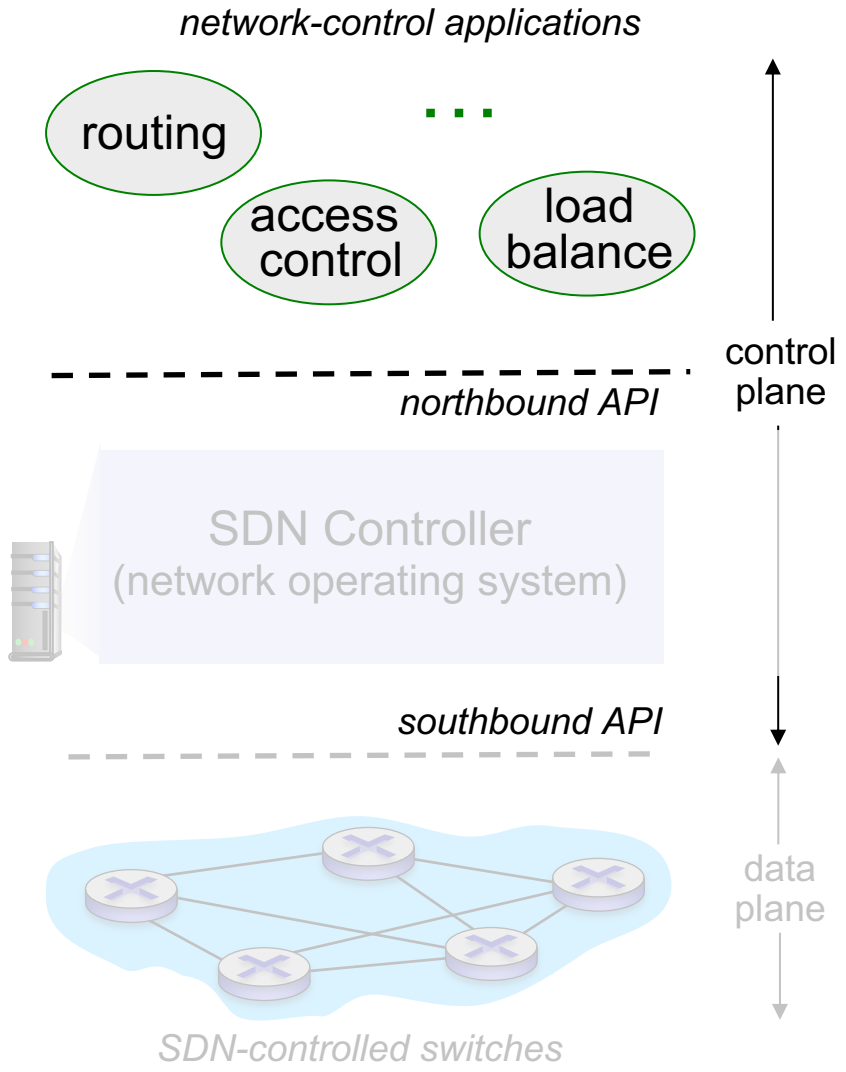
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# Software Defined Networking (SDN)

## 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

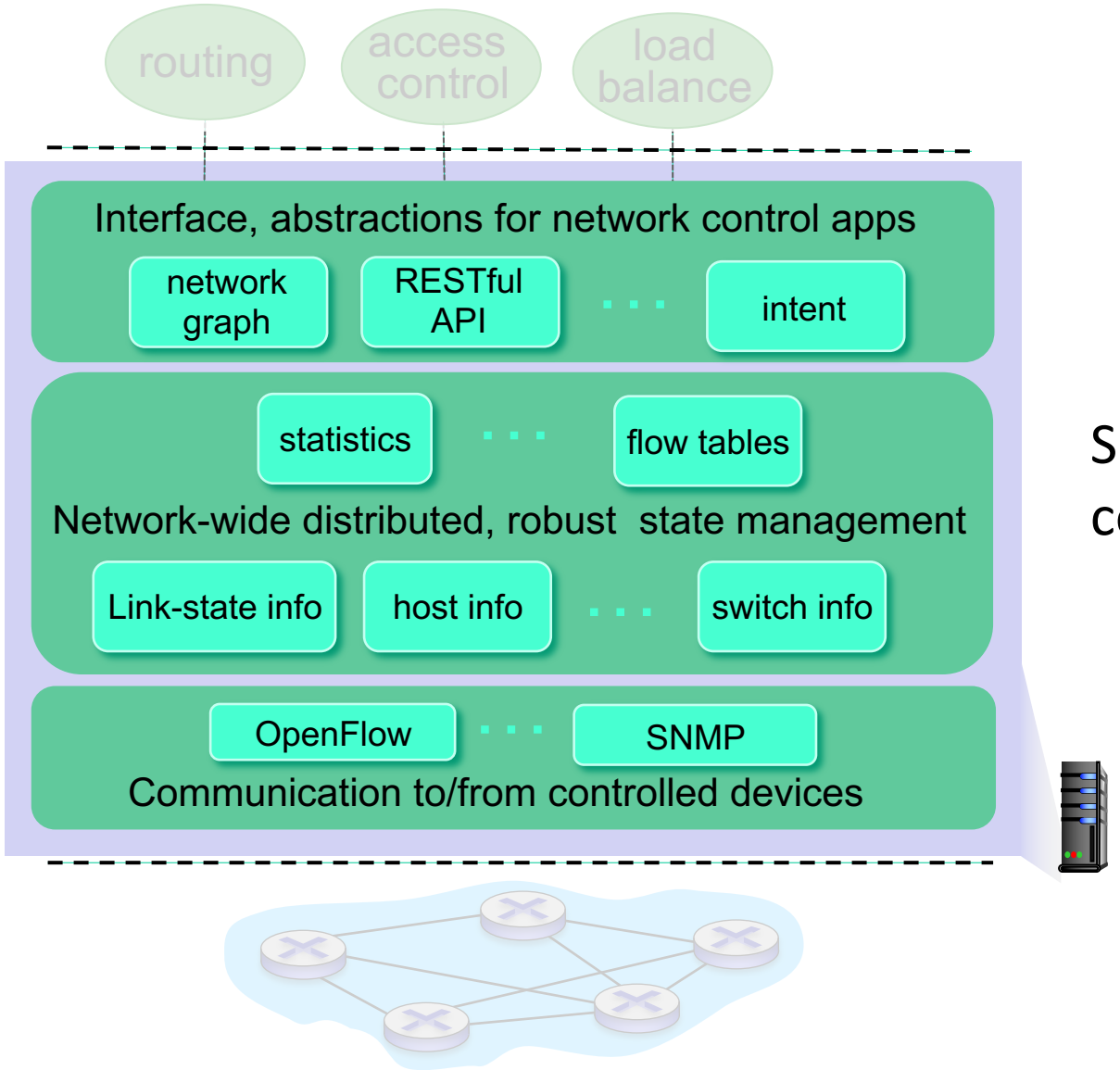


# Components of SDN Controller

interface layer to network control apps: abstractions API

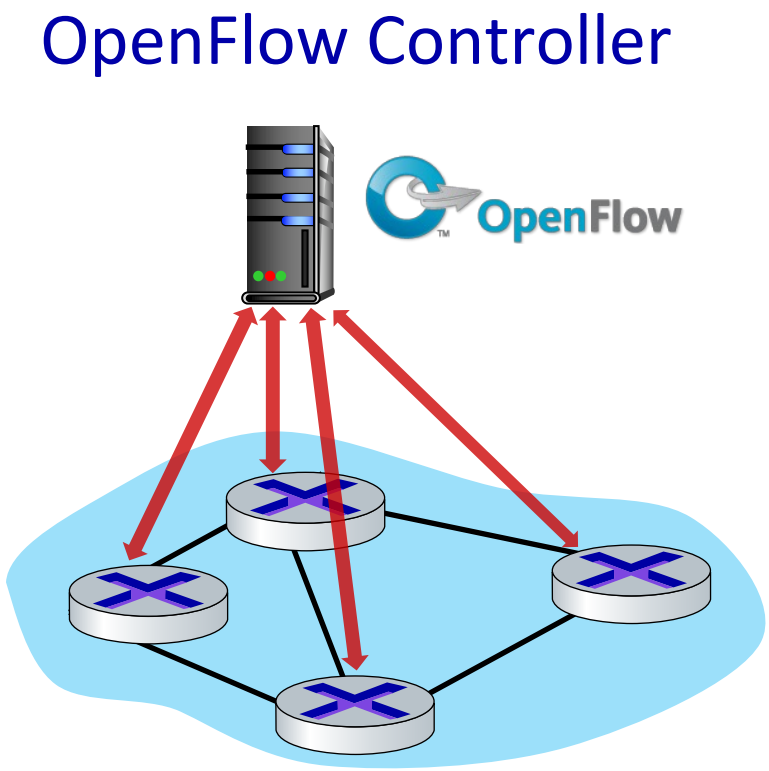
network-wide state management : state of networks links, switches, services: a *distributed database*

*communication*: communicate between SDN controller and controlled switches

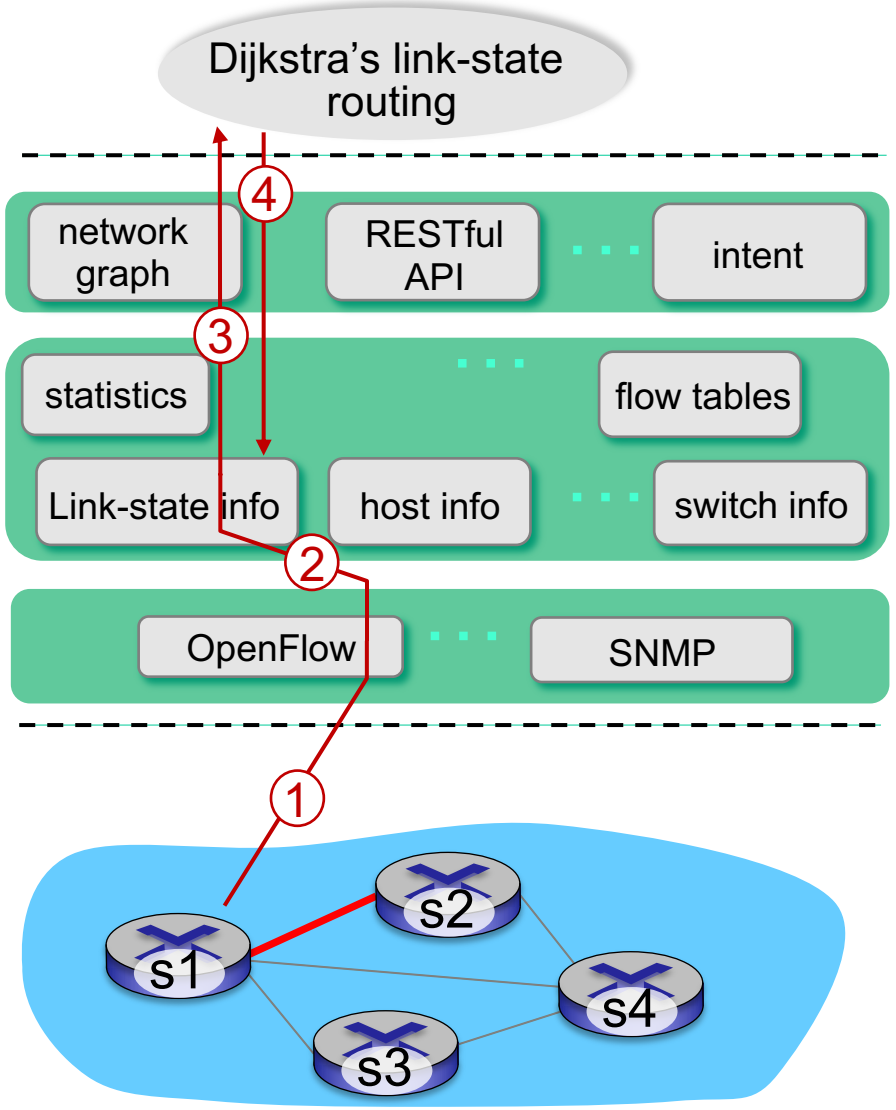


# OpenFlow protocol

- 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.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions



# SDN: control/data plane interaction example



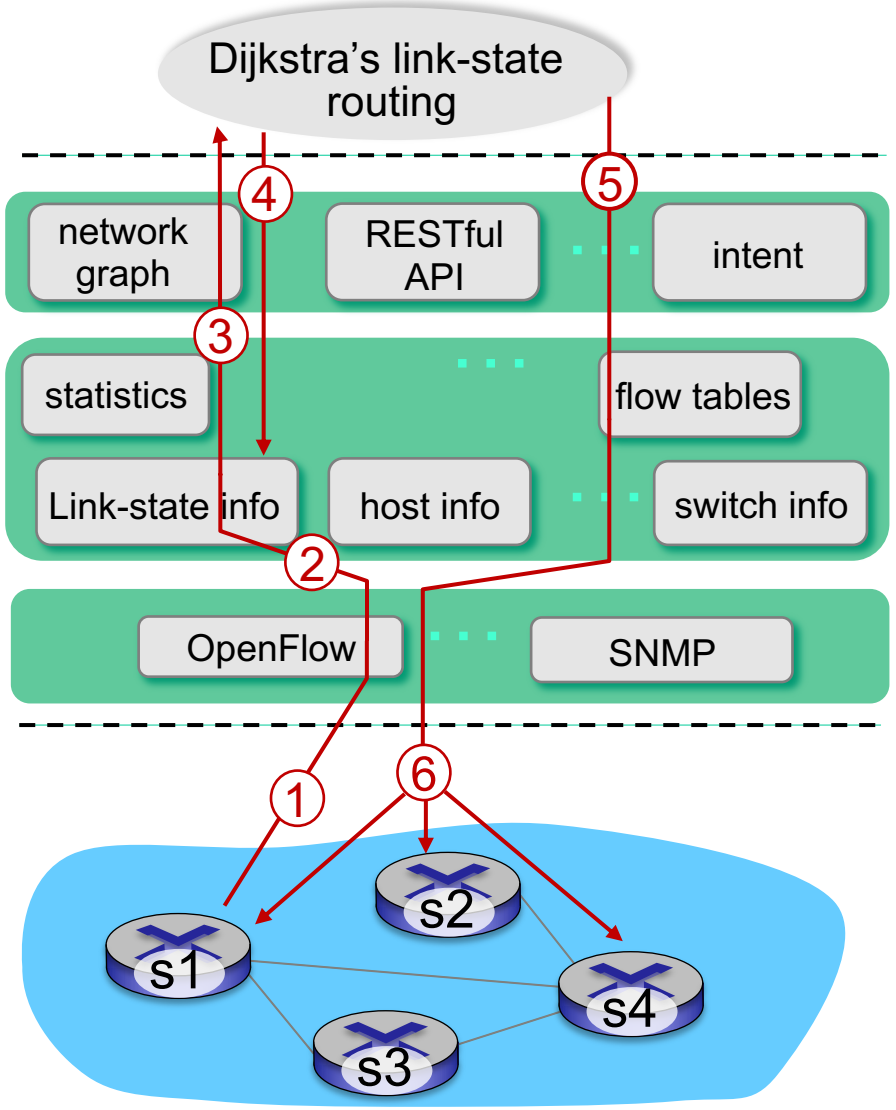

① S1, experiencing link failure uses OpenFlow port status message to notify controller

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

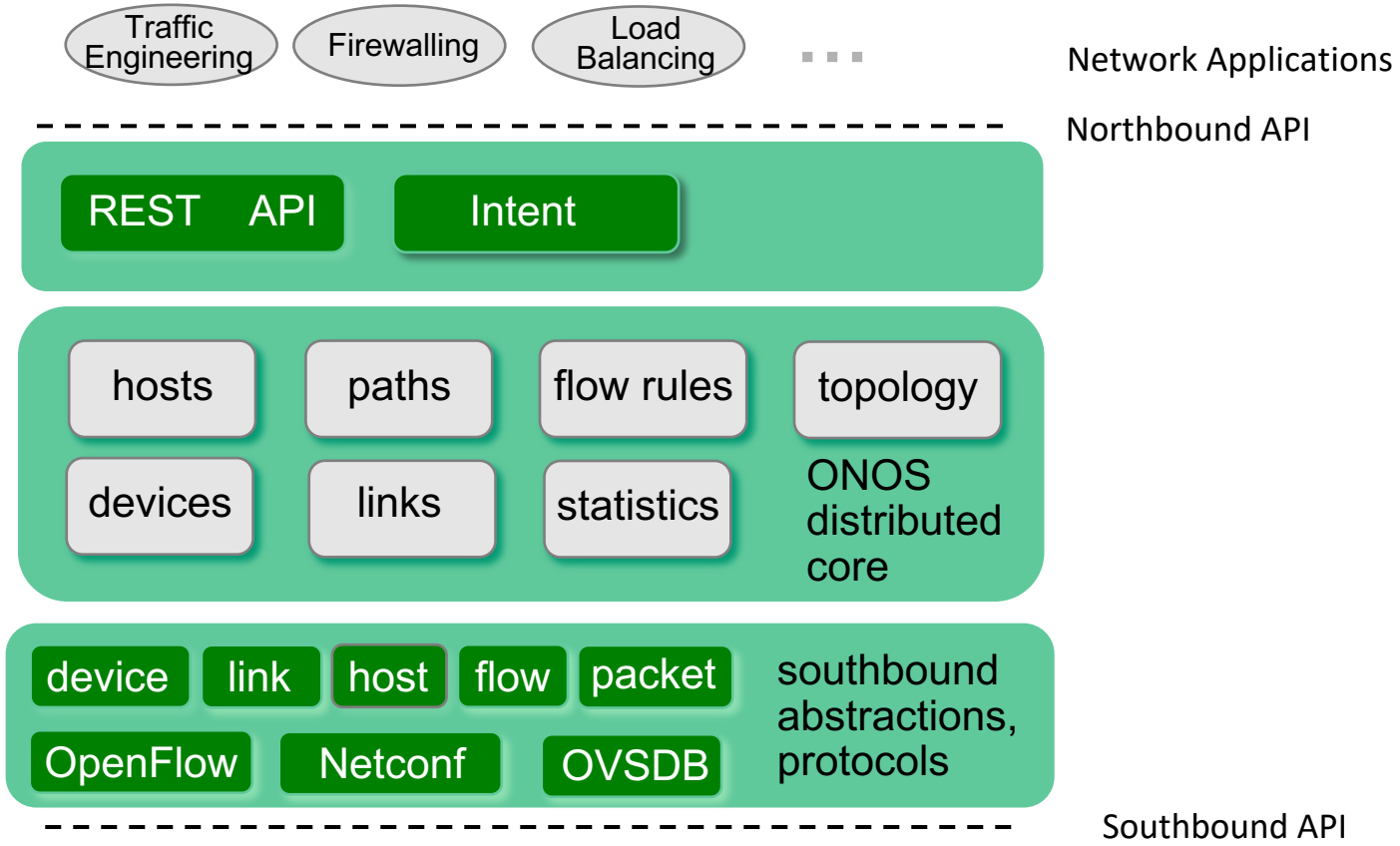
# SDN: control/data plane interaction example




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

# ONOS SDN 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

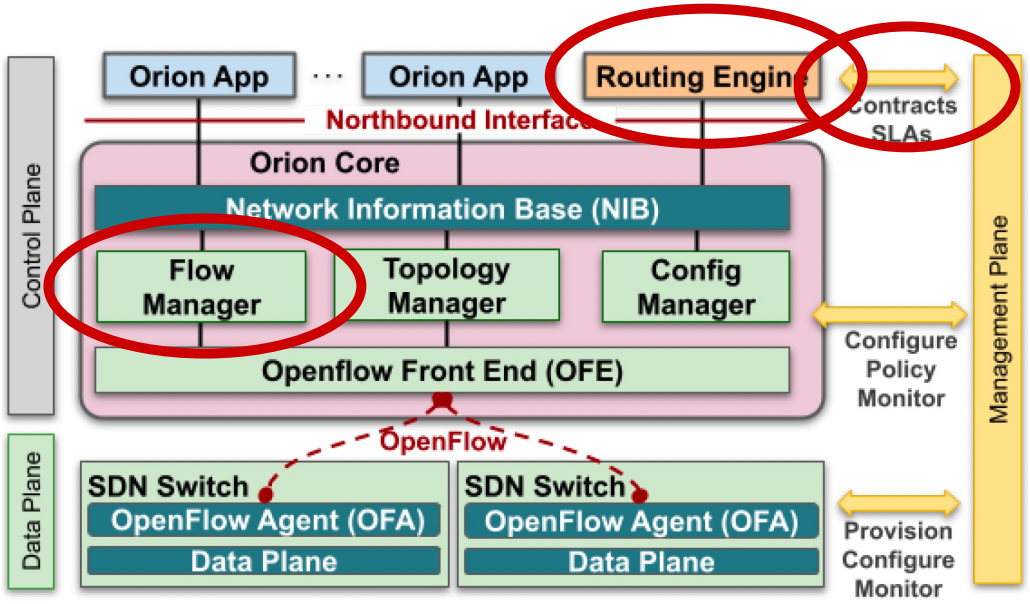
ONOS: early, influential open-source SDN controller

# Google ORION SDN Control Plane

ORION: Google's SDN control plane (*NSDI'21*): control plane for Google's datacenter (Jupiter) and wide area (B4) networks

- routing (intradomain, iBGP), traffic engineering: implemented in *applications* on top of ORION core
- edge-edge flow-based controls (e.g., CoFlow scheduling) to meet contract SLAs
- management: pub-sub distributed microservices in Orion core, OpenFlow for switch signaling/monitoring

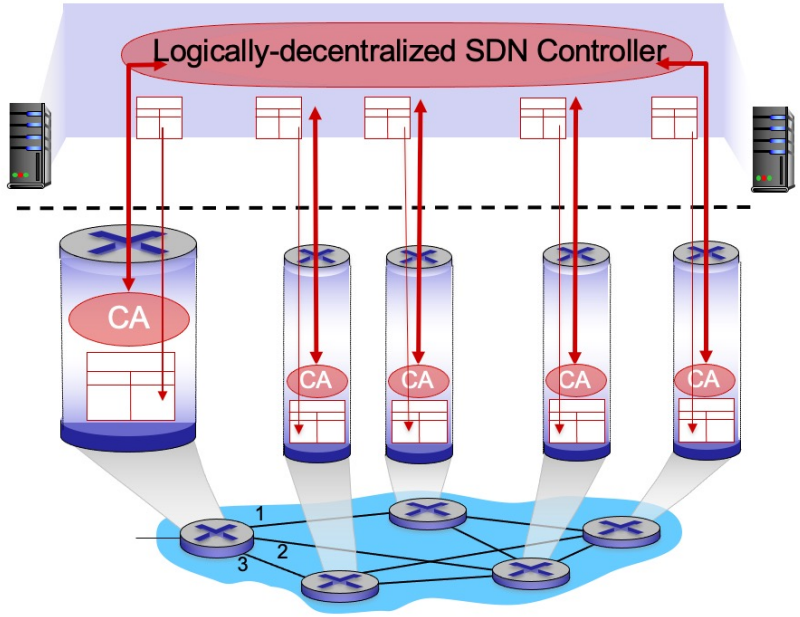
Orion SDN architecture and core apps




Note: ORION provides *intradomain* services within Google's network

# Google, SDN: the value of a logically-centralized abstraction

- new opportunities to more formally, intentionally manage configuration




*" … in a single building or even across a private global WAN such that we[Google] were designing, the location and role of every network element is known ahead of time. So - rather than allowing a node to boot, exchange messages with neighbors, and learn its role in the global network, we knew we could configure the role of all network elements into a logically centralized controller that could then inform all relevant elements of new and changing configurations. This shifted the problem of network management from managing the configuration of individual network elements to managing the configuration of a logical network, dramatically simplifying the problem both from a technical perspective but also from the perspective of the cognitive load on network managers."∂*

*https://www.youtube.com/watch?v=qn7d_iK92TI*

*Amin Vahdat, Engineering Fellow, VP for the Machine Leaning, Systems, and Cloud AI team*

# 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: beyond a single AS
- SDN critical in 5G cellular networks

# SDN and the Future of Traditional Network Protocols

- SDN-computed versus router-computer forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
  - controller sets sender rates based on router-reported (to controller) congestion levels

How will implementation of network functionality (SDN versus protocols) evolve?

# Control Plane Outline

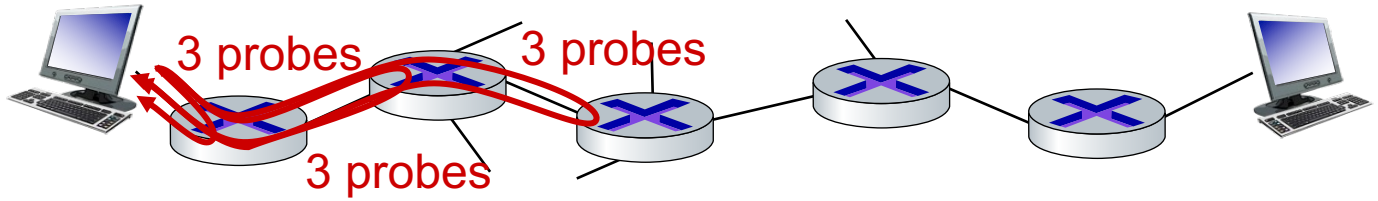
- Routing Algorithm
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing Among ISPs: BGP
- SDN Control Plane
- Internet Control Message Protocol

# ICMP: internet control message protocol

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

| Type | Code | description |
|---|---|---|
| 0 | 0 | echo reply (ping) |
| 3 | 0 | dest. network unreachable |
| 3 | 1 | dest host unreachable |
| 3 | 2 | dest protocol unreachable |
| 3 | 3 | dest port unreachable |
| 3 | 6 | dest network unknown |
| 3 | 7 | dest host unknown |
| 4 | 0 | source quench (congestion control - not used) |
| 8 | 0 | echo request (ping) |
| 9 | 0 | route advertisement |
| 10 | 0 | router discovery |
| 11 | 0 | TTL expired |
| 12 | 0 | bad IP header |

# Traceroute and ICMP




- source sends sets of UDP segments to destination
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops

# Network Layer: 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: ONOS
- ✓Internet Control Message Protocol: ICMP

*next stop: link layer!*