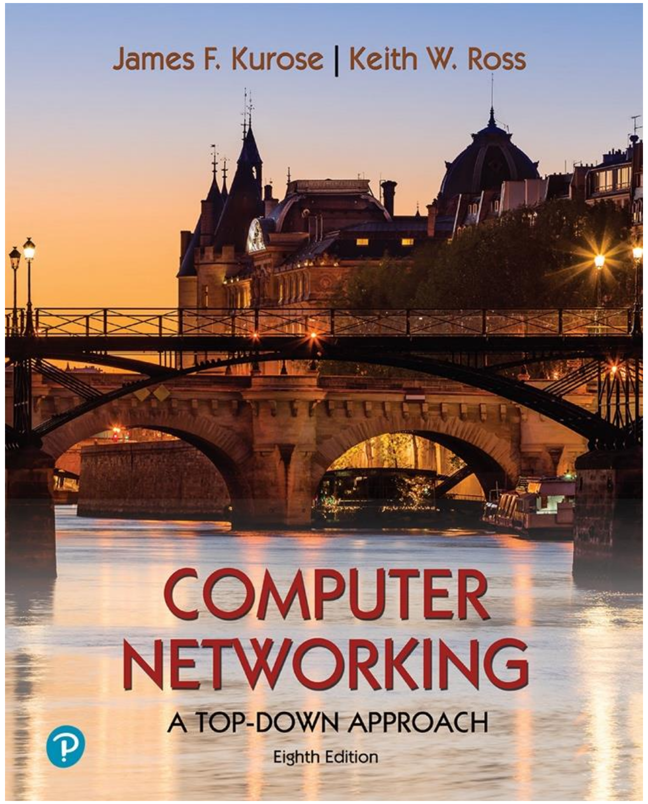
Chapter 5
Network Layer: Control Plane

Yaxiong Xie

Department of Computer Science and Engineering
University at Buffalo, SUNY

Adapted from the slides of the book's authors



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

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” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Network-layer functions

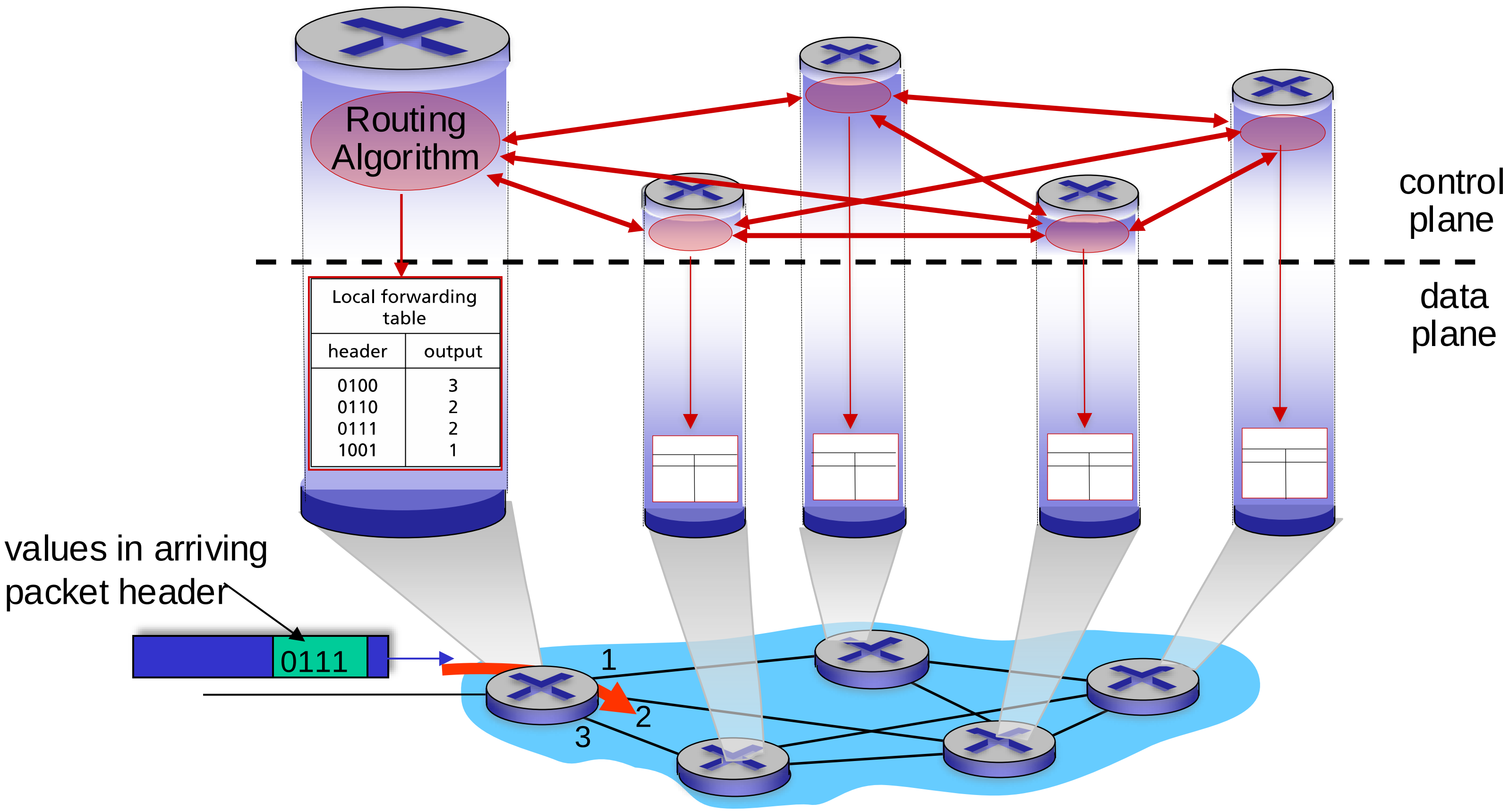
- forwarding: move packets from router's input to appropriate router output — *data plane*
- routing: determine route taken by packets from source to destination — *control plane*

Two approaches to structuring network control plane:

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

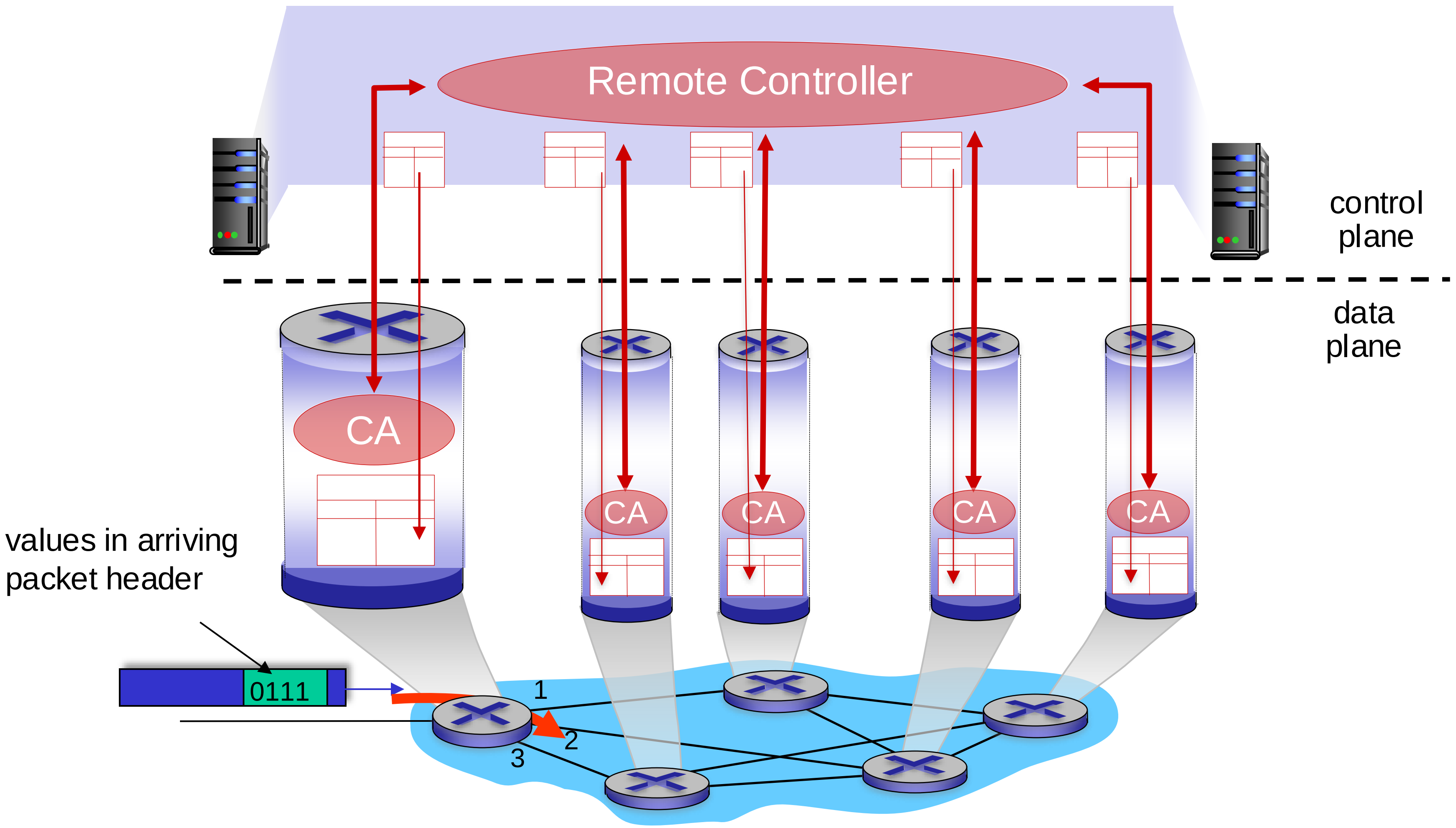
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane

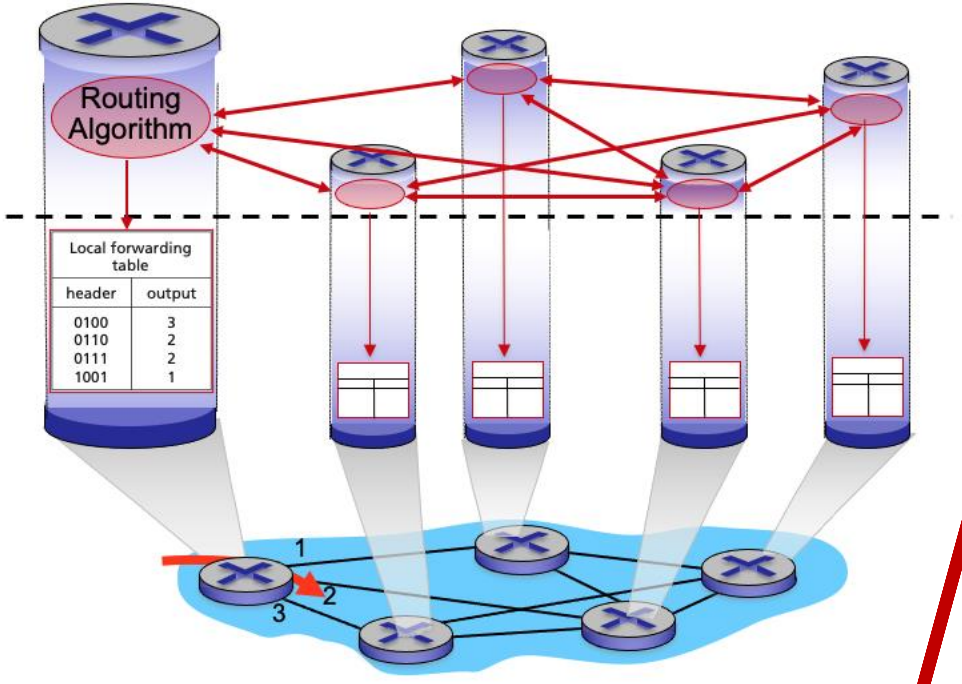


Software-Defined Networking (SDN) control plane

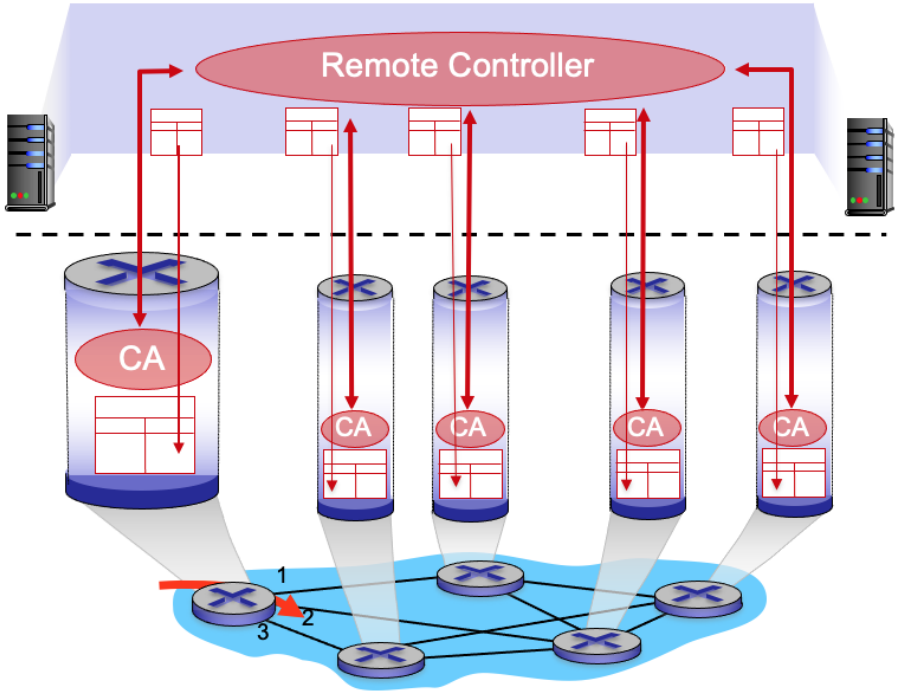
Remote controller computes, installs forwarding tables in routers



Per-router control plane



SDN control plane



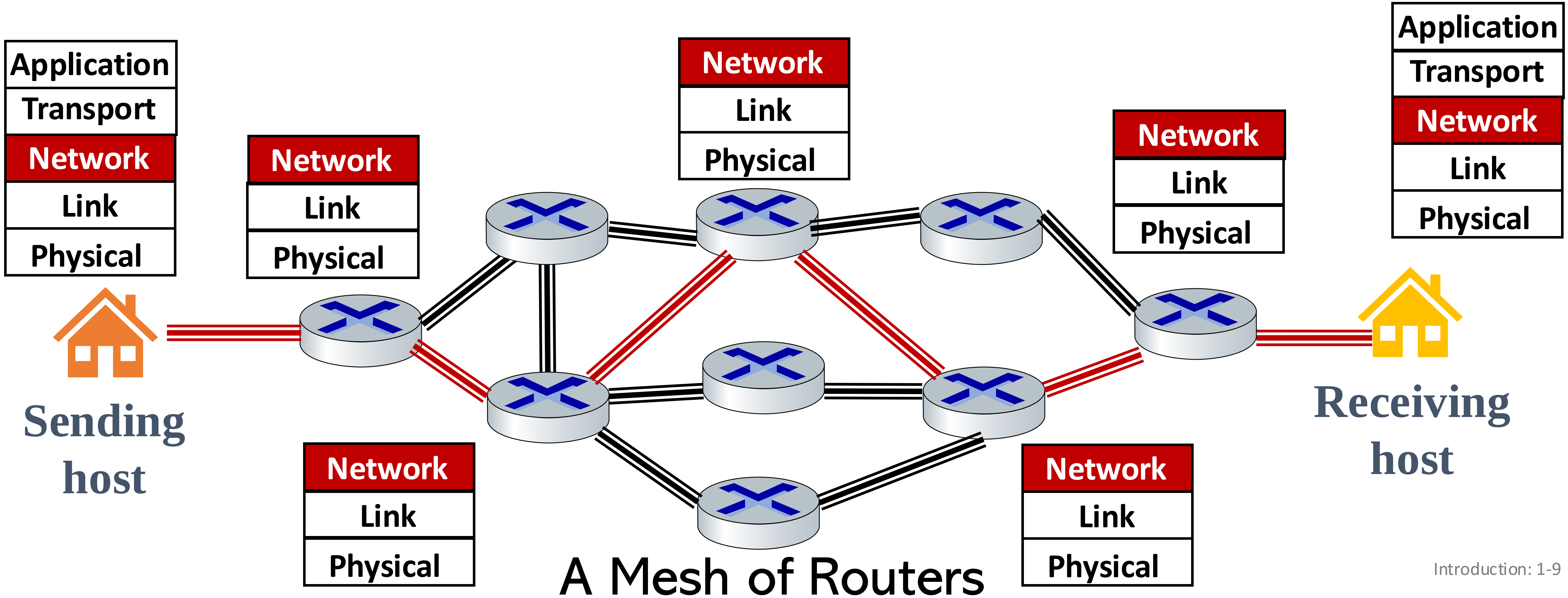
Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

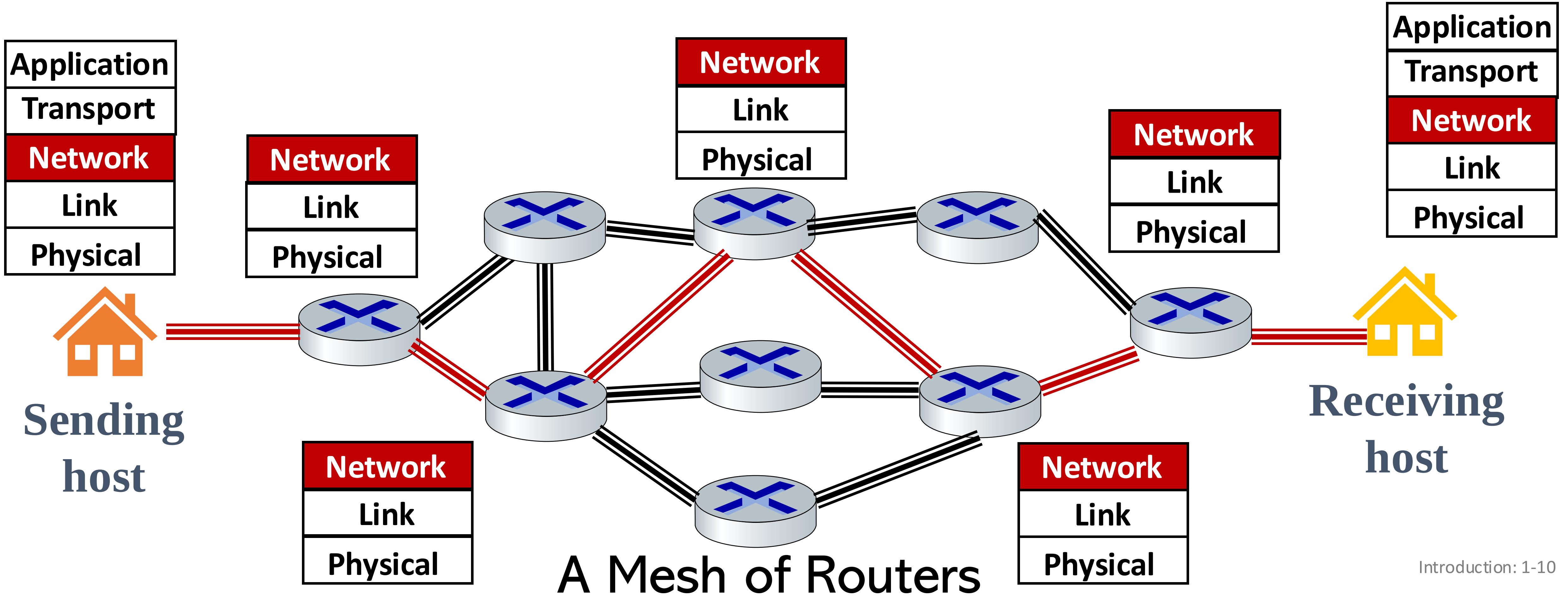
Goal of routing protocols

Determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers



Goal of routing protocols

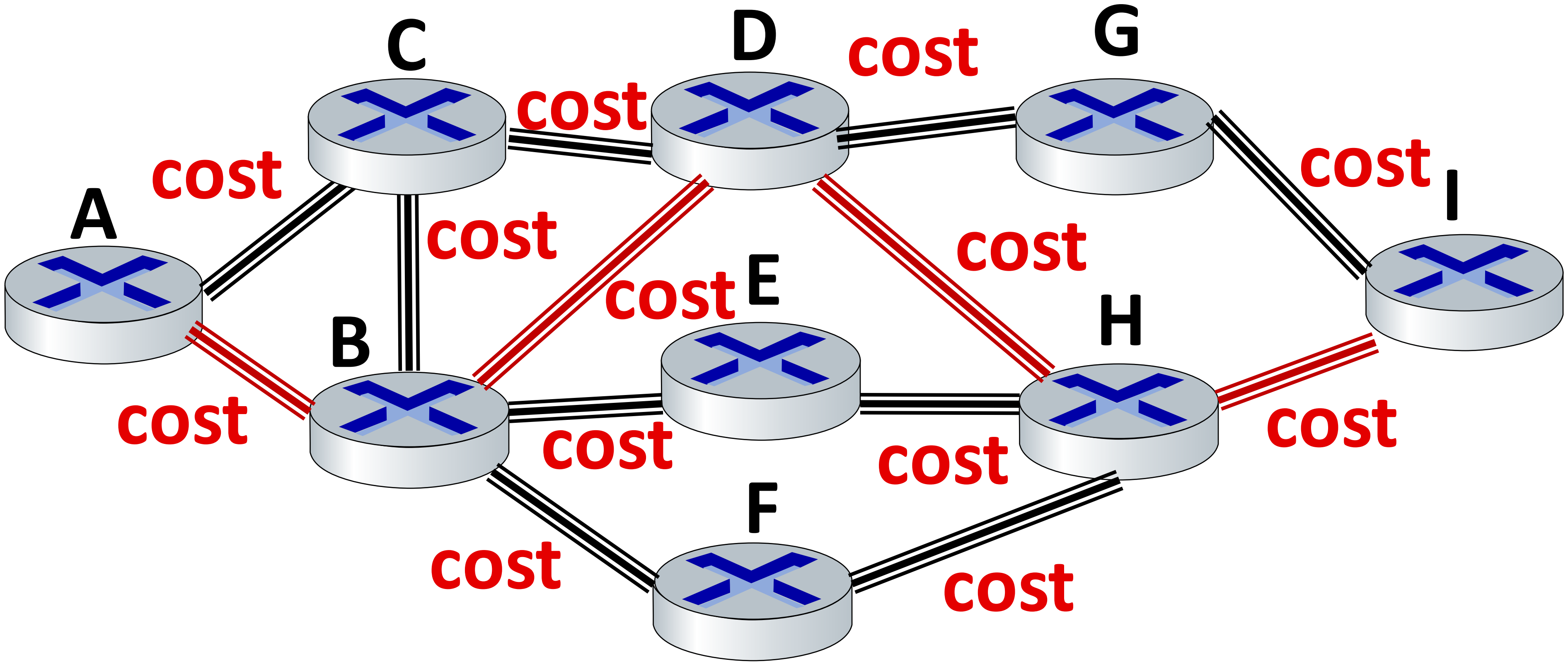
What is a "good" path?



Goal of routing protocols

What is a "good" path? **Path with the smallest cost**

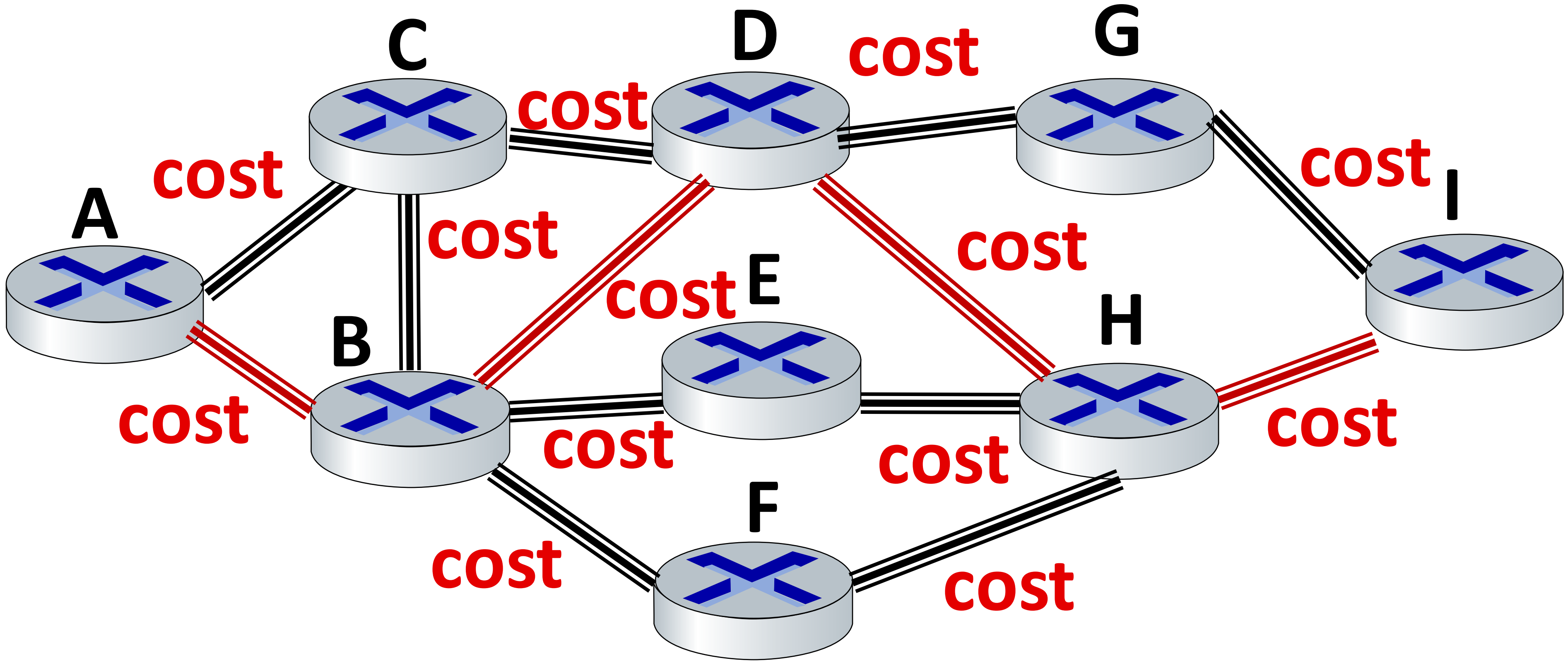
cost = 1 -> **Path with the smallest number of hops**



Goal of routing protocols

What is a "good" path? **Path with the smallest cost**

$$\text{cost} = \frac{1}{bandwidth}$$ **-> Path with the highest speed**



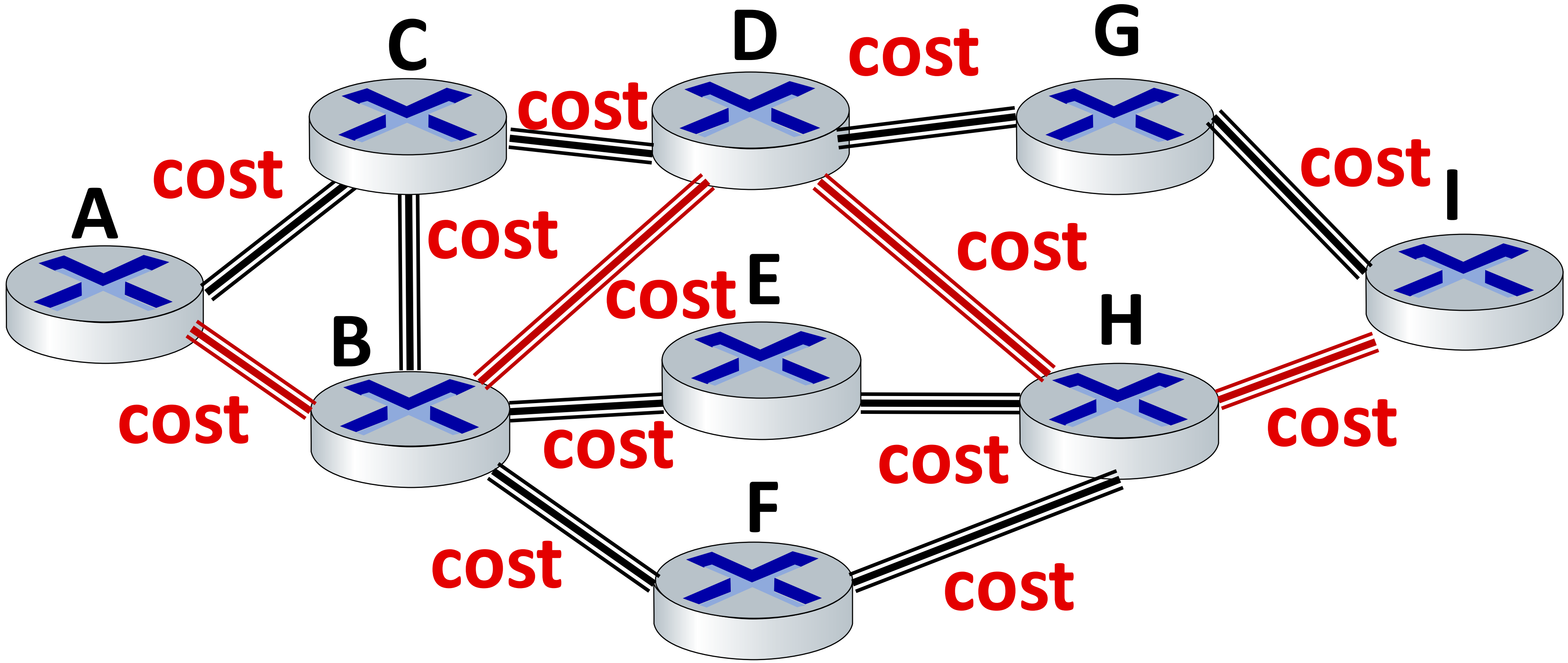
Goal of routing protocols

What is a "good" path? **Path with the smallest cost**

Example: A -> I

A->C->D->G->I A->B->D->G->I A->B->E->H->I

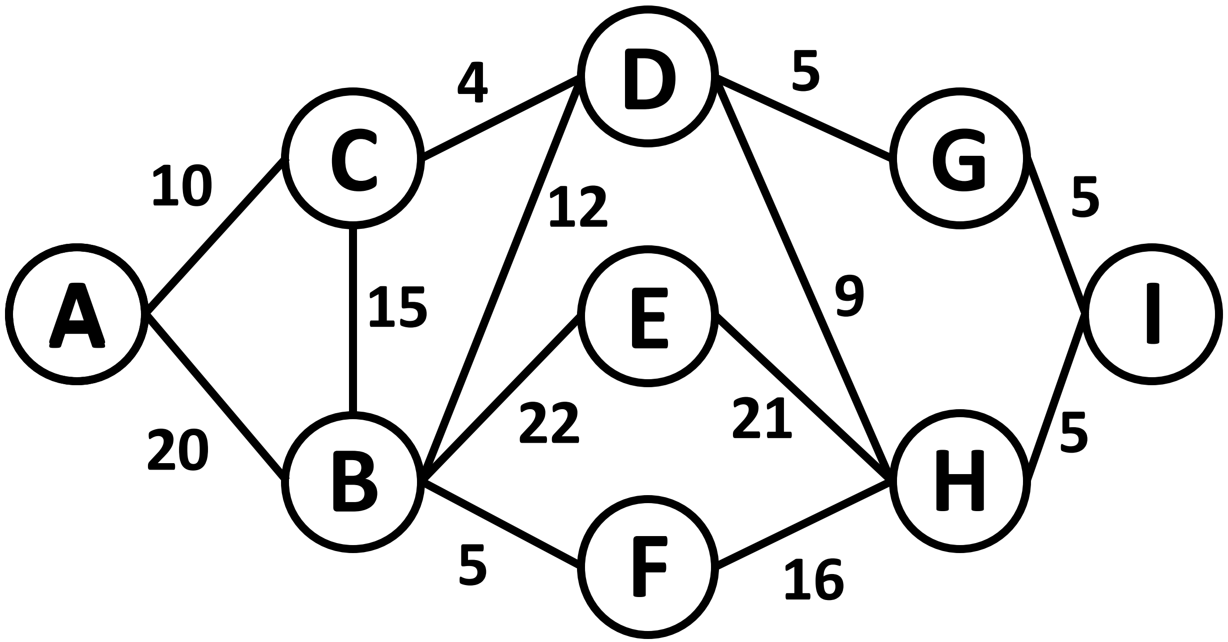
A->C->B->E->H->I A->B->F->H->I A->B->D->H->I



Goal of routing protocols

What is a "good" path?

Path with the smallest cost



Mathematical Problem:

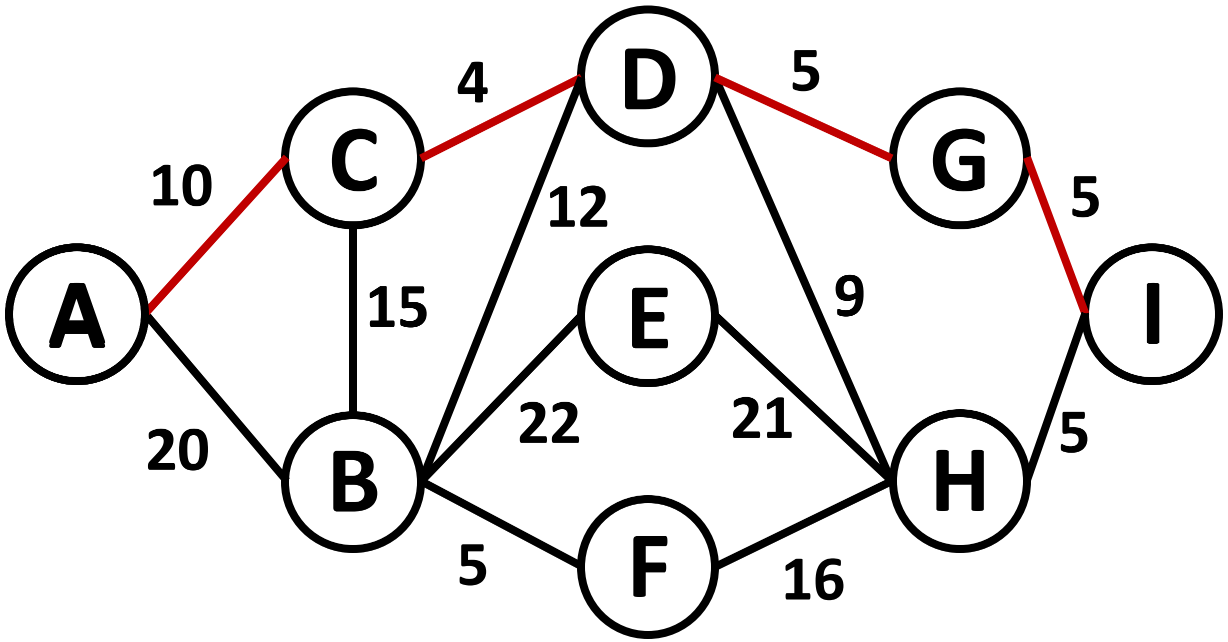
Find the shortest path in this graph

Example: Router A:

Find the shortest path to all the other routers inside the network

Goal of routing protocols

What is a "good" path?



Path with the smallest cost

Router A->I: A->C->D->G->I

IP Address Range	Interface
200.23.16.0/23	B
200.23.18.0/23	B
IP of Router I	C

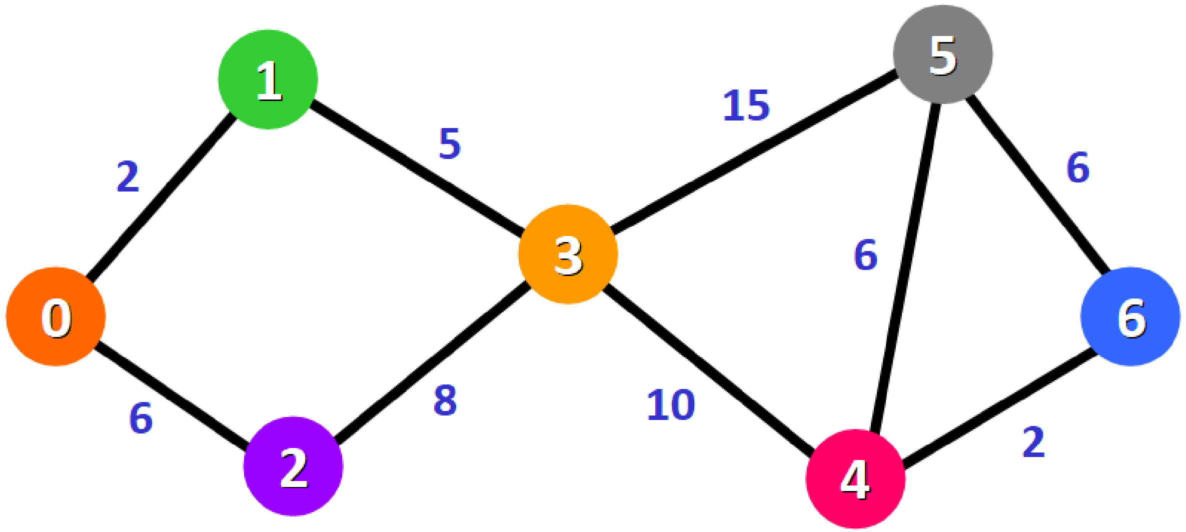
Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

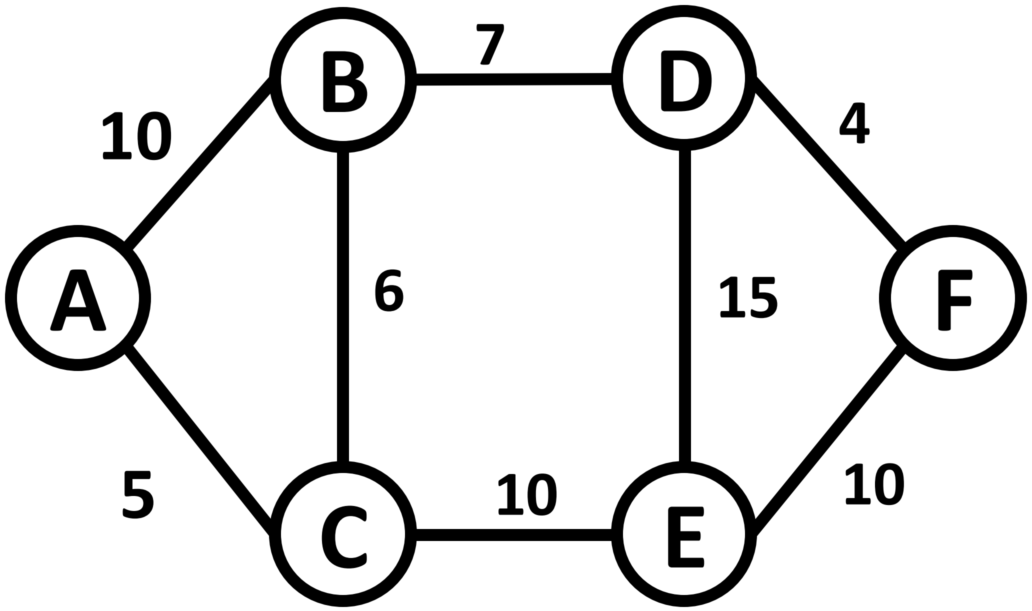
- network management, configuration
  - SNMP
  - NETCONF/YANG

Dijkstra's link-state routing 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



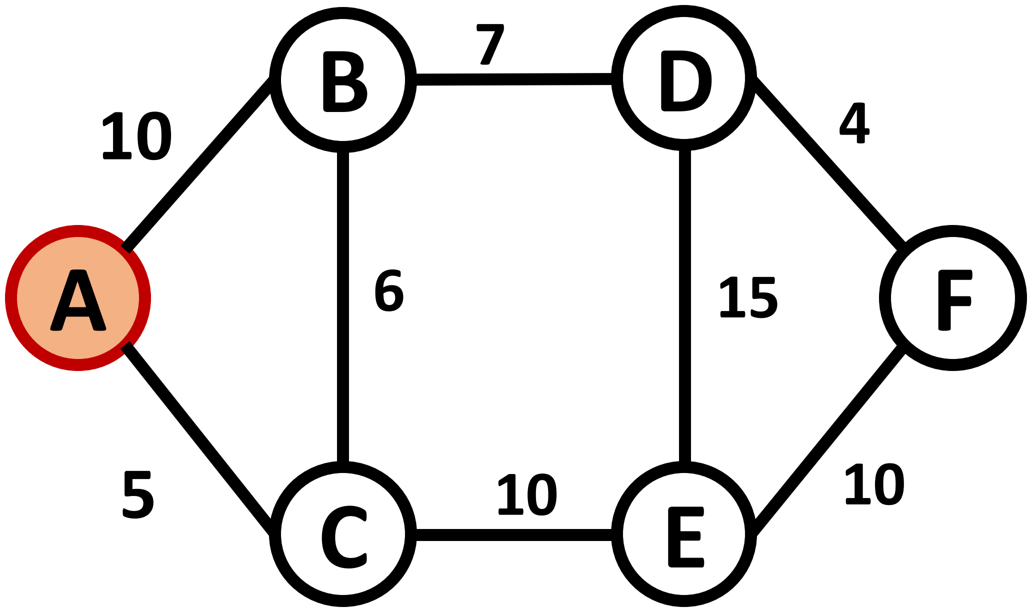
Dijkstra’s link-state routing algorithm



Initialization

Visited Node	Shorted Distance
	∞
	∞
	∞
	∞
	∞
	∞

Dijkstra's link-state routing algorithm



Initialization

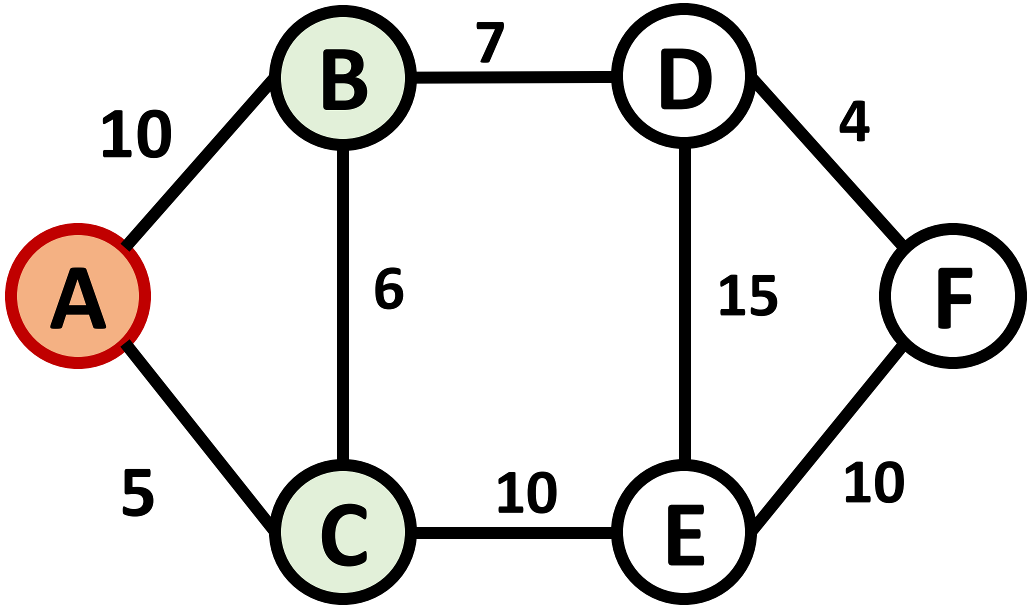
Visited Node

A

Shorted Distance

A	0
B	∞
C	∞
D	∞
E	∞
F	∞

Dijkstra's link-state routing algorithm



Initialization

Visited Node

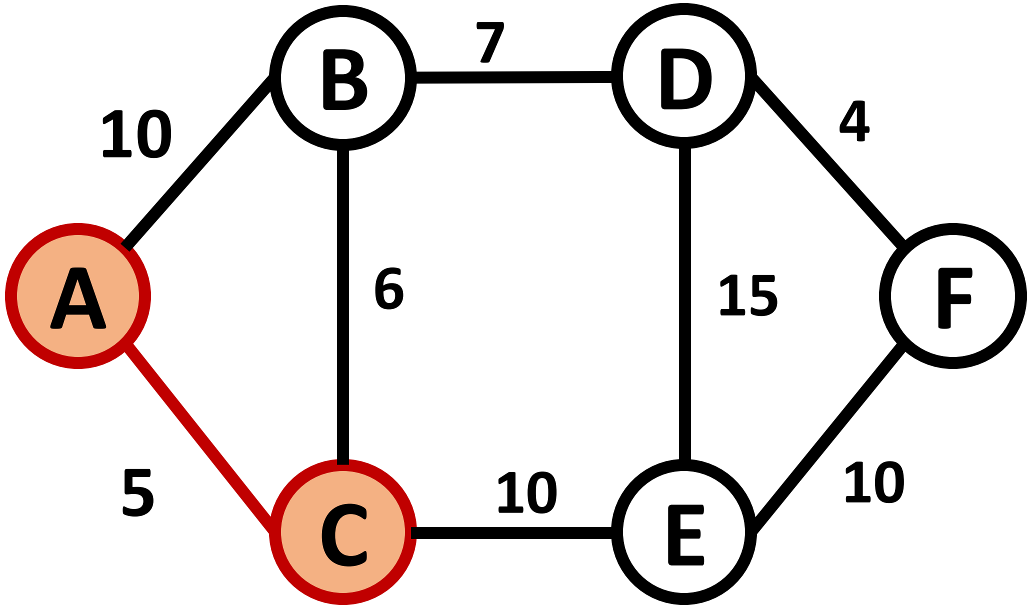
A

Shorted Distance

A	0
B	10
C	5
D	∞
E	∞
F	∞

Check the distance between the source and all its neighbors

Dijkstra’s link-state routing algorithm



Initialization

Visited Node

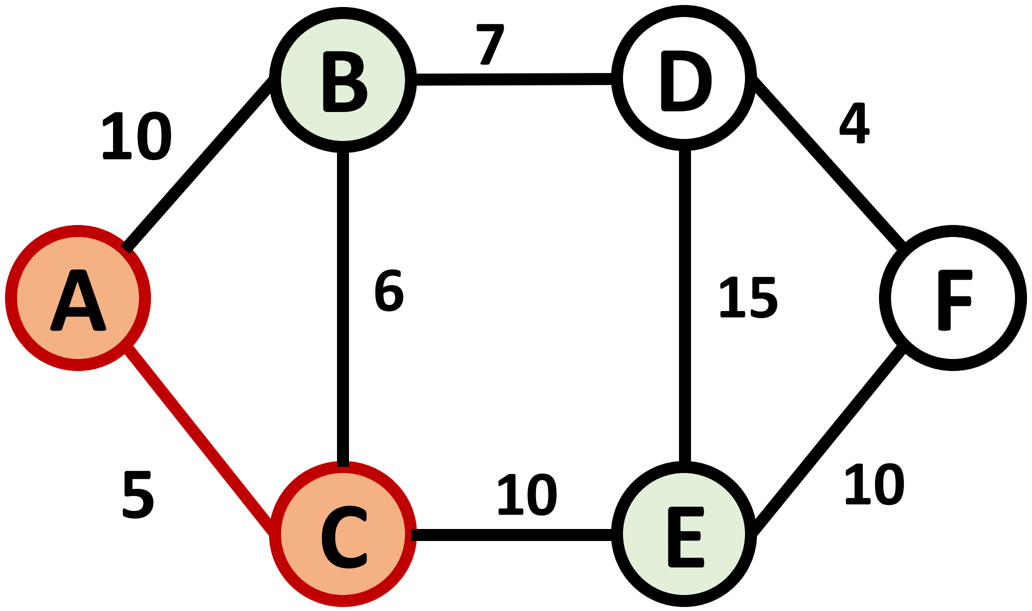
A
C

Shorted Distance

A	0
B	10
C	5
D	∞
E	∞
F	∞

Mark the selected neighbor (C) as visited

Dijkstra's link-state routing algorithm



A->B **10**

A->C->B **11**

A->C->E **15**

Initialization

Visited Node

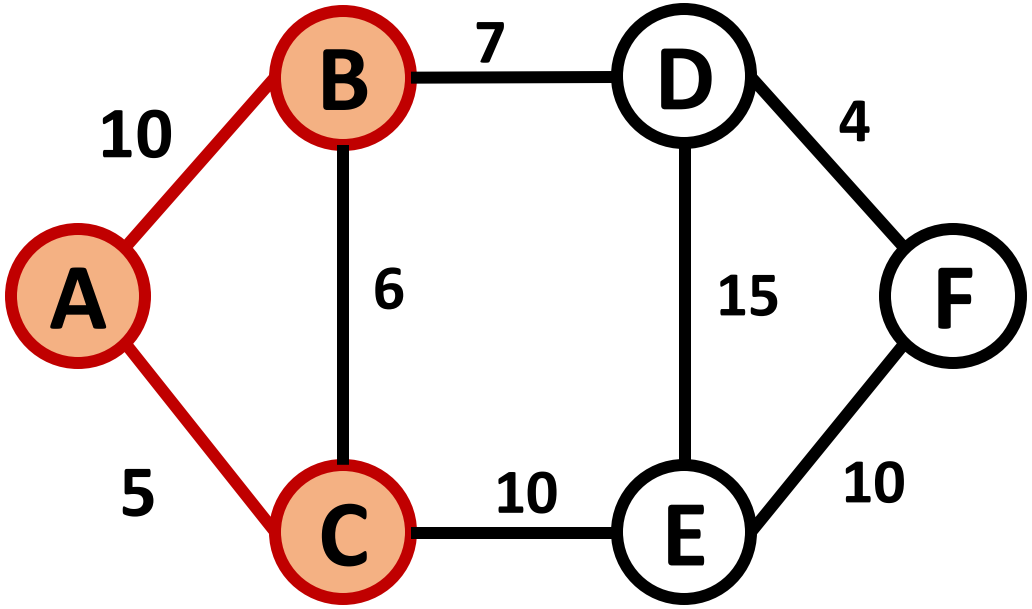
A
C

Shorted Distance

A	0
B	10
C	5
D	∞
E	∞
F	∞

Check the distance between source and all visited nodes' neighbors

Dijkstra's link-state routing algorithm



A->B	10
A->C->B	11
A->C->E	15

Initialization

Visited Node

A
B
C

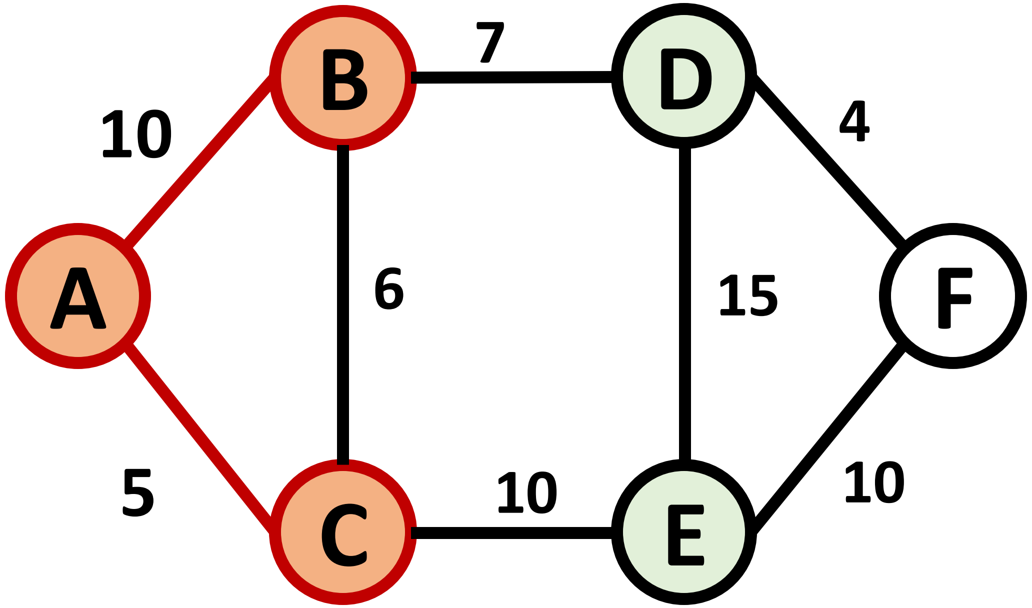
Shorted Distance

A	0
B	10
C	5
D	∞
E	∞
F	∞

Mark the selected neighbor (B) as visited

Dijkstra’s link-state routing algorithm

Initialization



A->B->D 17

A->C->E 15

Visited Node

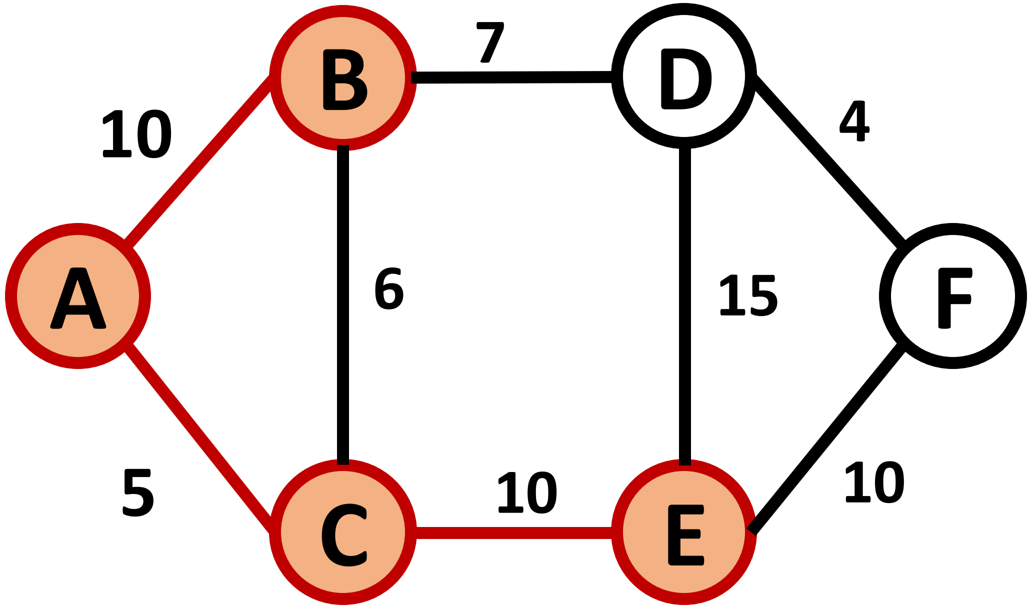
A
B
C

Shorted Distance

A	0
B	10
C	5
D	∞
E	∞
F	∞

Check the distance between source and all visited nodes’ neighbors

Dijkstra’s link-state routing algorithm



A->B->D 17

A->C->E 15

Initialization

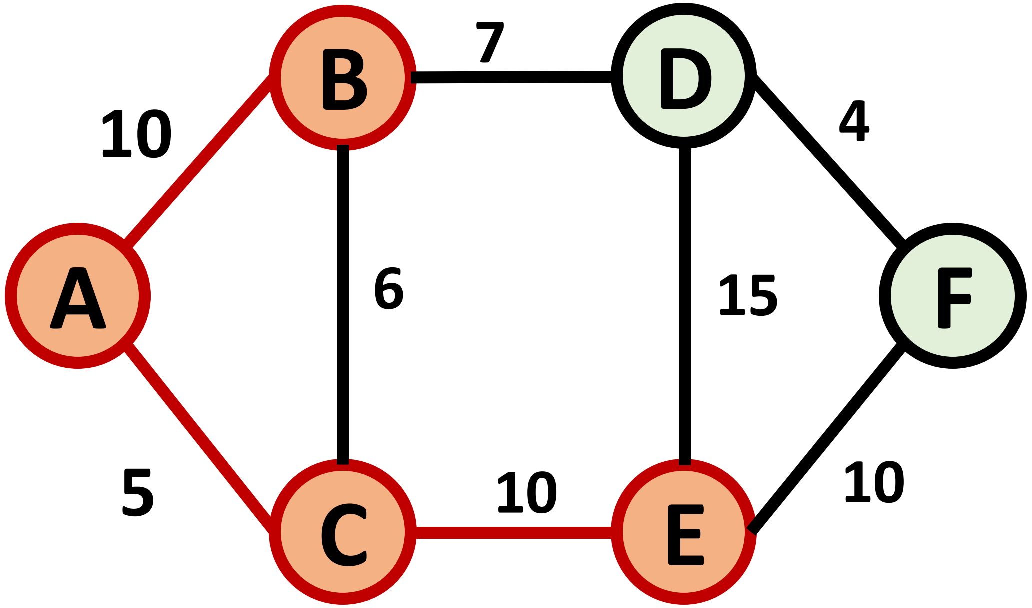
Visited Node
A
B
C
E

Shorted Distance

A	0
B	10
C	5
D	17
E	15
F	∞

Mark the selected neighbor (E) as visited

Dijkstra’s link-state routing algorithm



A->B->D **17**

A->C->E->D **30**

A->C->E->F **25**

Initialization

Visited Node

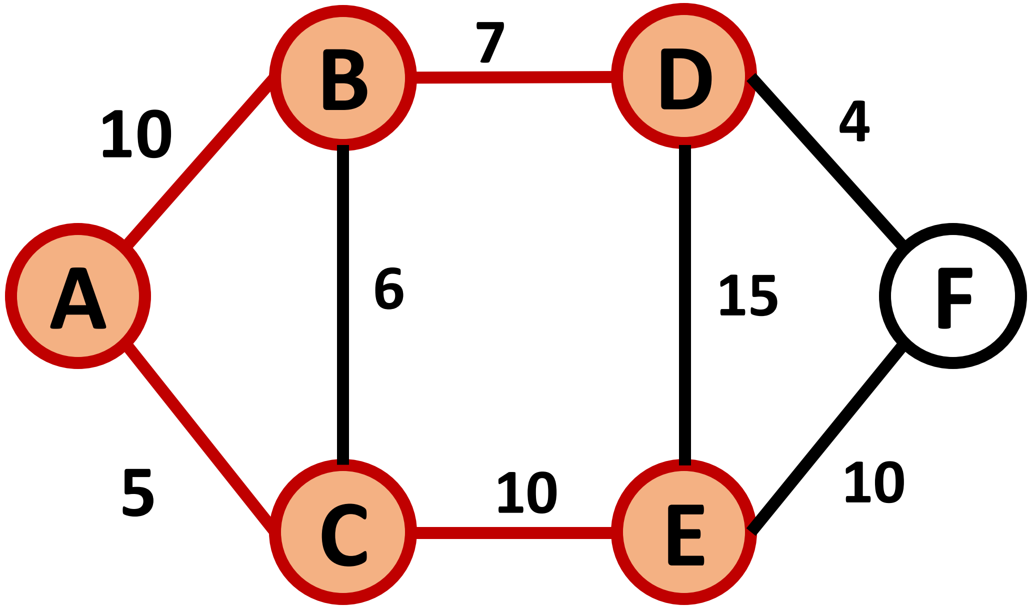
A
B
C
E

Shorted Distance

A	0
B	10
C	5
D	17
E	15
F	25

Check the distance between source and all visited nodes’ neighbors

Dijkstra's link-state routing algorithm



Initialization

A->B->D 17

A->C->E->D 30

A->C->E->F 25

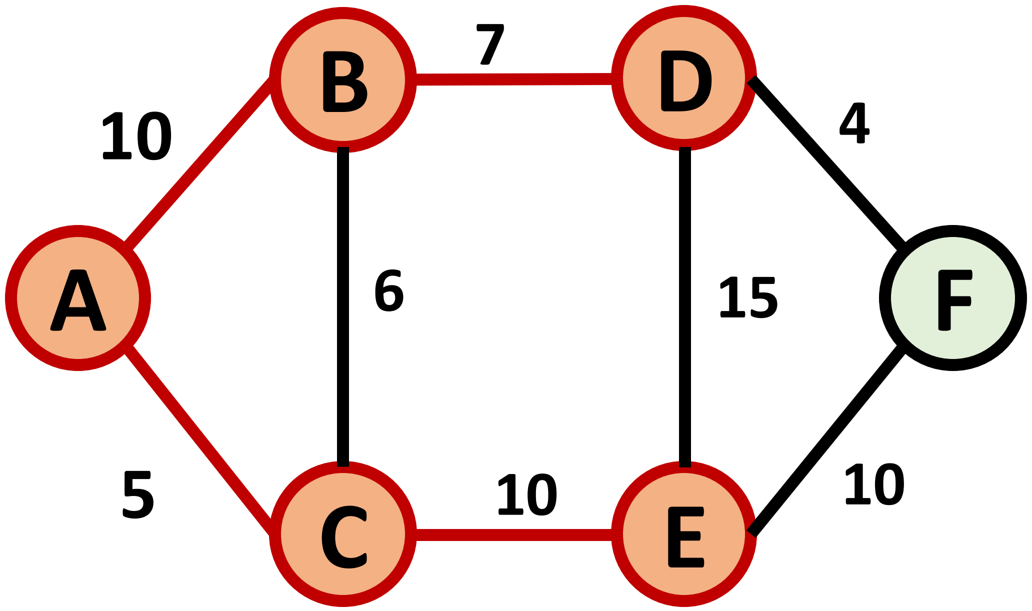
Visited Node
A
B
C
D
E

Shorted Distance

A	0
B	10
C	5
D	17
E	15
F	25

Mark the selected neighbor (D) as visited

Dijkstra's link-state routing algorithm



A->B->D->F 21

A->C->E->F 25

Initialization

Visited Node

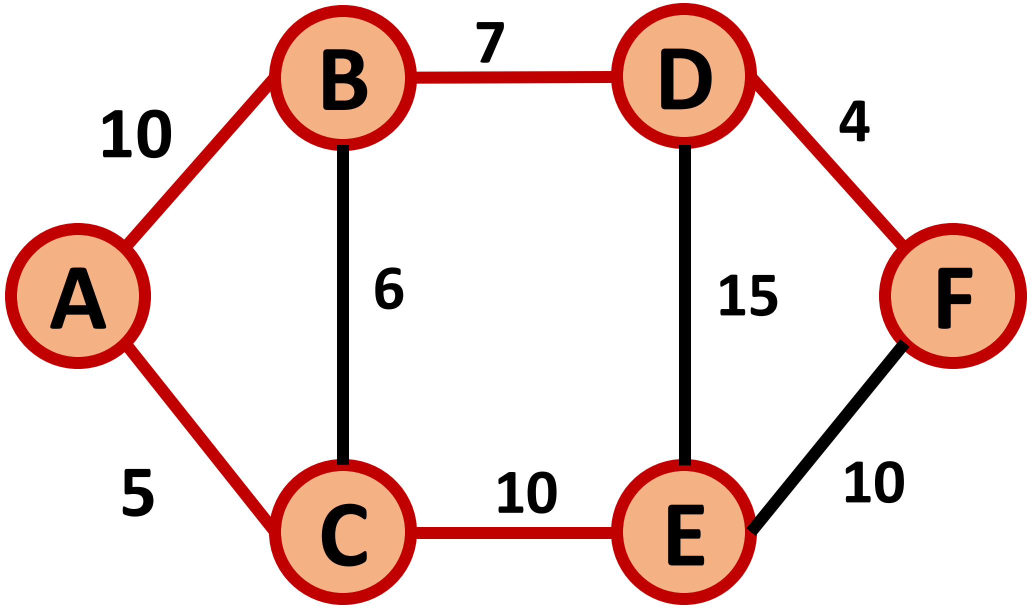
A
B
C
D
E

Shorted Distance

A	0
B	10
C	5
D	17
E	15
F	25

Check the distance between source and all visited nodes' neighbors

Dijkstra’s link-state routing algorithm



A->B->D->F 21

A->C->E->F 25

Initialization

Visited Node

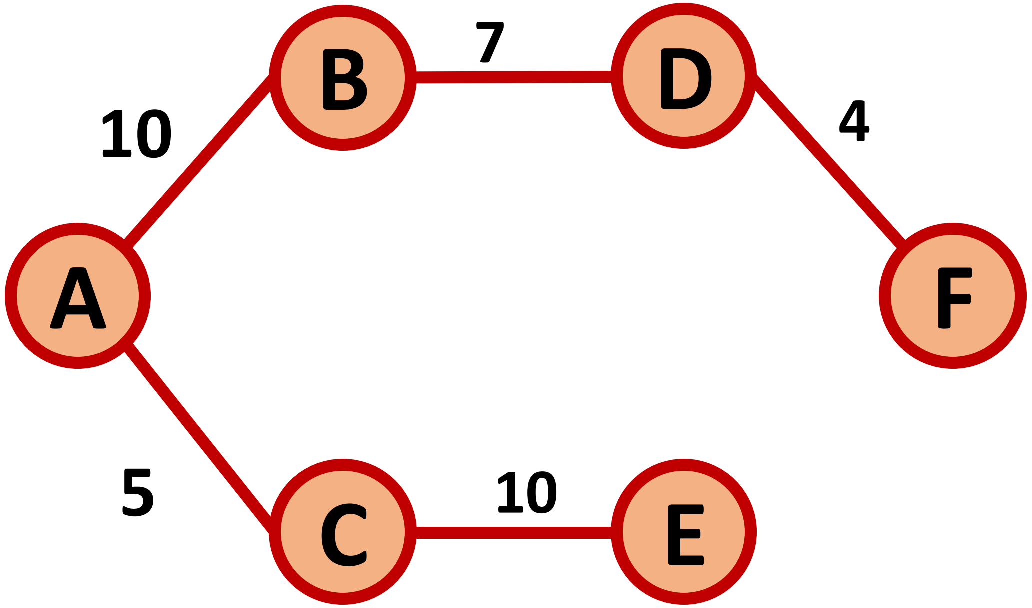
A
B
C
D
E
F

Shorted Distance

A	0
B	10
C	5
D	17
E	15
F	21

Mark the selected neighbor (F) as visited

Dijkstra’s link-state routing algorithm: Result



Link	Next Hop	Overall Cost
A->B	B	10
A->C	C	5
A->D	B	17
A->E	C	15
A->F	B	21

Network layer: "control plane" roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Distance vector algorithm

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

Bellman-Ford equation

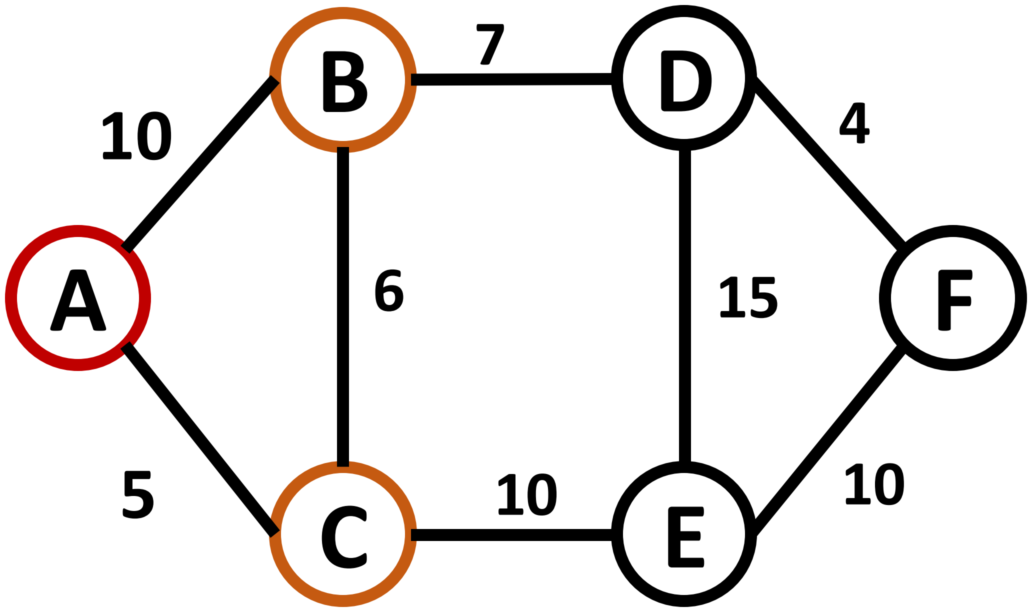
Let $D_x(y)$: cost of least-cost path from x to y.

Then:

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

- *min* taken over all neighbors v of x
- direct cost of link from x to v
- v's estimated least-cost-path cost to y

Bellman-Ford Example



Q: Find the shortest distance A -> F

A has two neighbors: B and C

Shortest distance:

B->F : 11 | **C->F : 20**

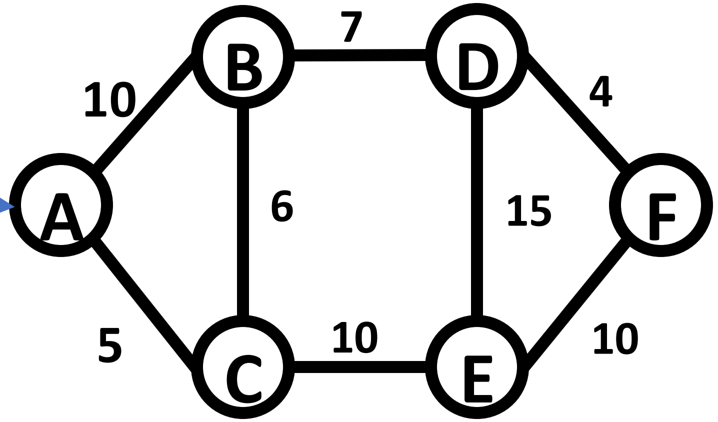
A->B : 10 | **A->C : 5**

A->B->F : 21 < A->C->F : 25

Shortest Path: A->B->F | **Next Hop: B** | **Total Cost: 21**

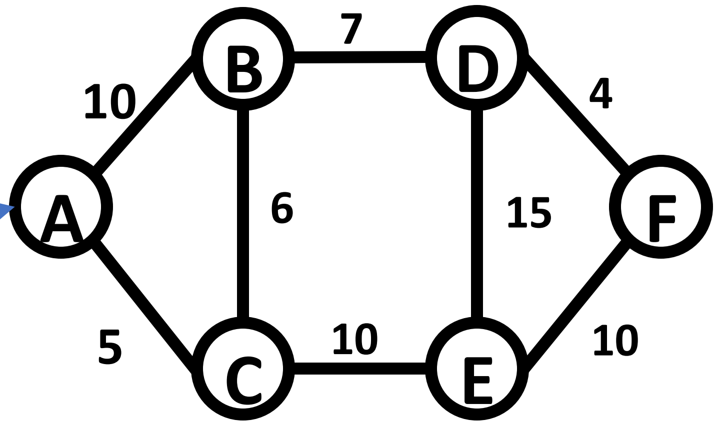
Distance vector algorithm:

Link	Cost
A->B	10
A->C	5



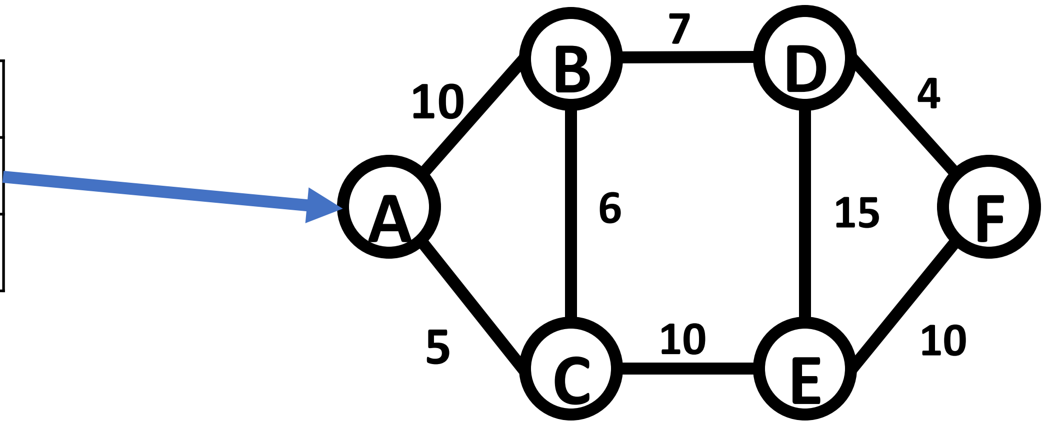
Distance vector algorithm:

Link	Cost
A->B	10
A->C	5
A->D	∞
A->E	∞
A->F	∞



Distance vector algorithm:

Link	Cost
A->B	10
A->C	5

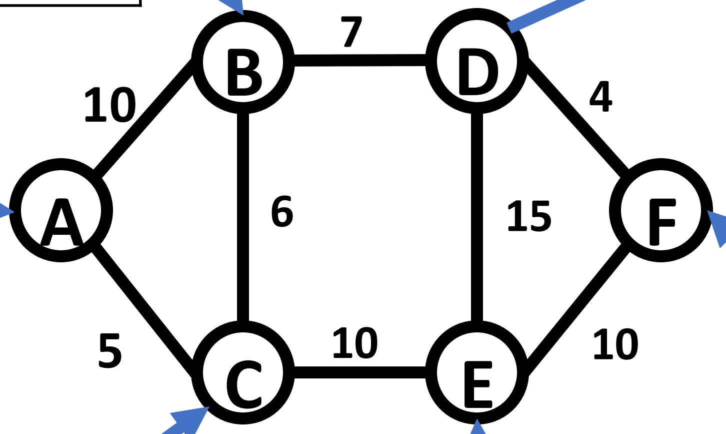


Distance vector algorithm:

Link	Cost
B->A	10
B->C	6
B->D	7

Link	Cost
D->B	7
D->E	15
D->F	4

Link	Cost
A->B	10
A->C	5



Link	Cost
F->D	4
F->E	10

Link	Cost
C->A	5
C->B	6
C->E	10

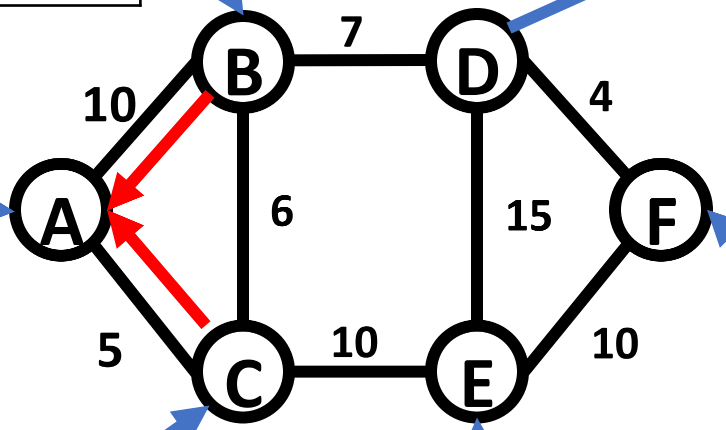
Link	Cost
E->C	10
E->D	15
E->F	10

Distance vector algorithm: iteration 1

Link	Cost
B->A	10
B->C	6
B->D	7

Link	Cost
D->B	7
D->E	15
D->F	4

Link	Cost
A->B	10
A->C	5



Link	Cost
F->D	4
F->E	10

Link	Cost
C->A	5
C->B	6
C->E	10

Link	Cost
E->C	10
E->D	15
E->F	10

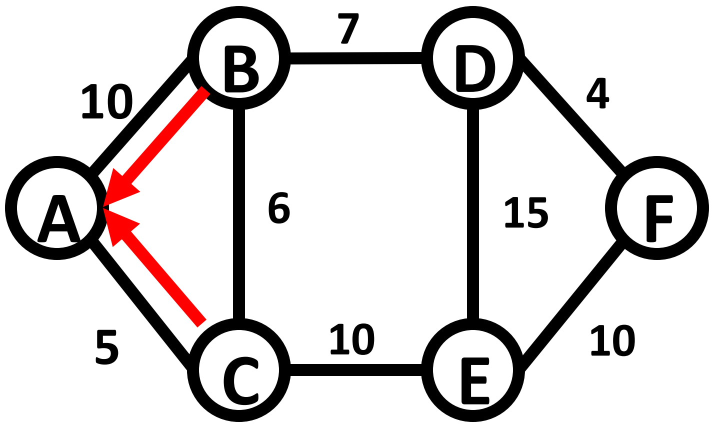
Distance vector algorithm: iteration 1

Router A has 3 distance vector tables

Link	Cost
A->B	10
A->C	5

Link	Cost
B->A	10
B->C	6
B->D	7

Link	Cost
C->A	5
C->B	6
C->E	10



New paths

A->B->C 16	>	A->C 5
A->B->D 17		New destination
A->C->B 11	>	A->B 10
A->C->E 15		New destination

New table

Link	Cost
A->B	10
A->C	5
A->D	17
A->E	15

Distance vector algorithm: iteration 1

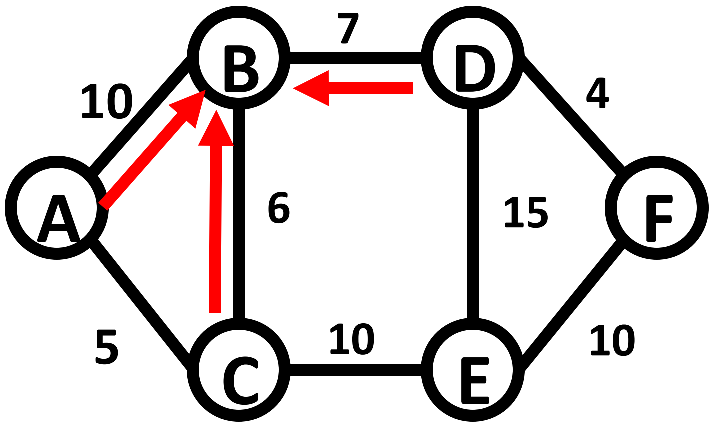
Router B has 4 distance vector tables

Link	Cost
A->B	10
A->C	5

Link	Cost
B->A	10
B->C	6
B->D	7

Link	Cost
C->A	5
C->B	6
C->E	10

Link	Cost
D->B	7
D->E	15
D->F	4



New paths

B->A->C 15 > B->C 6

B->C->E 16 New destination

B->C->A 11 > B->A 10

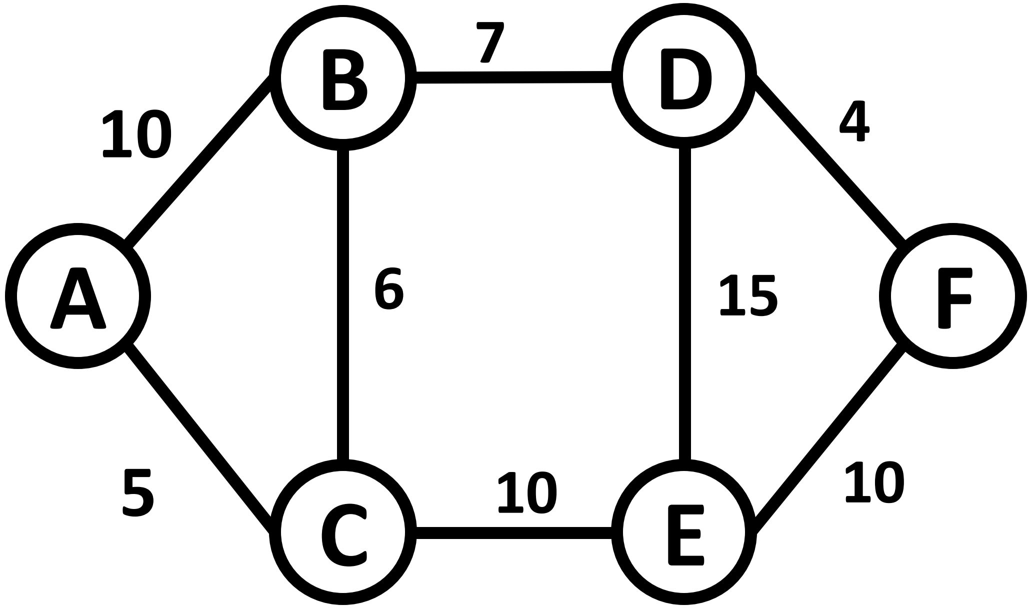
B->D->E 21 > B->C->E 16

B->D->F 11 New destination

New table

Link	Cost
B->A	10
B->C	6
B->D	7
B->E	16
B->F	11

Distance vector algorithm:



each node:

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

recompute DV estimates using DV received from neighbor

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

Distance vector algorithm:

each node:

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

↓

recompute DV estimates using DV received from neighbor

↓

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

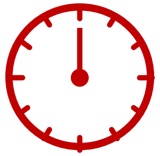
iterative, asynchronous: each local iteration caused by:

- 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!

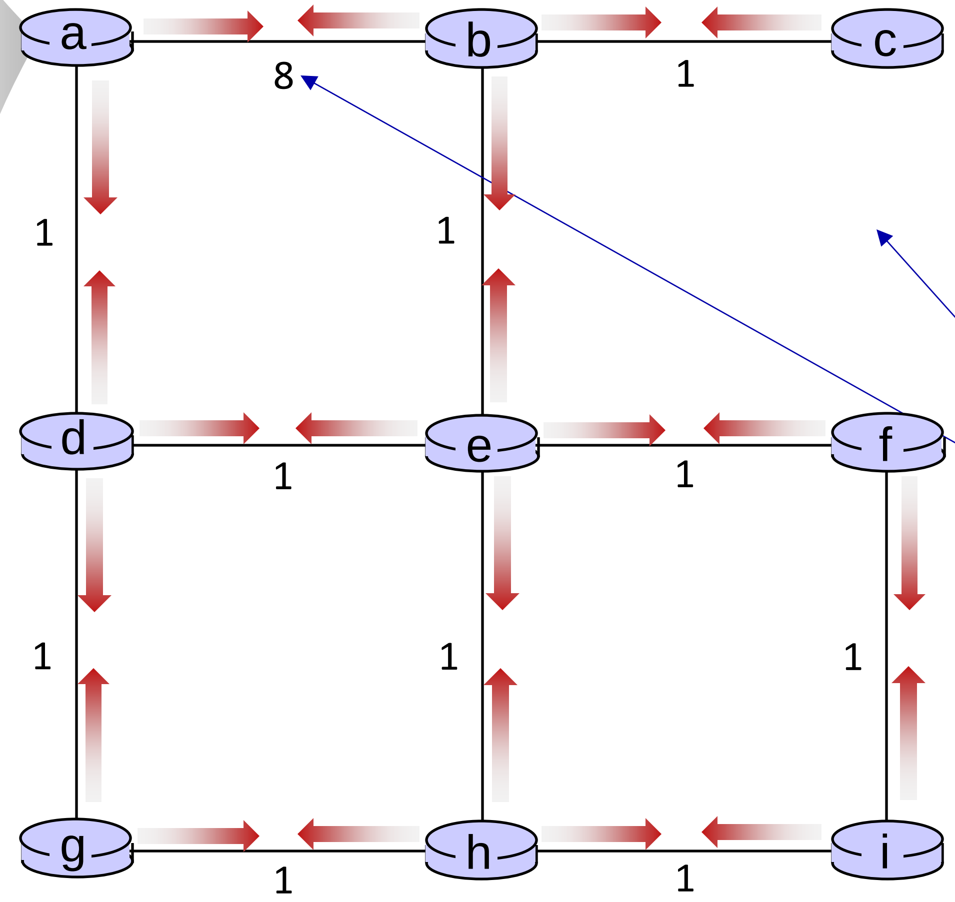
Distance vector: example



t=0

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

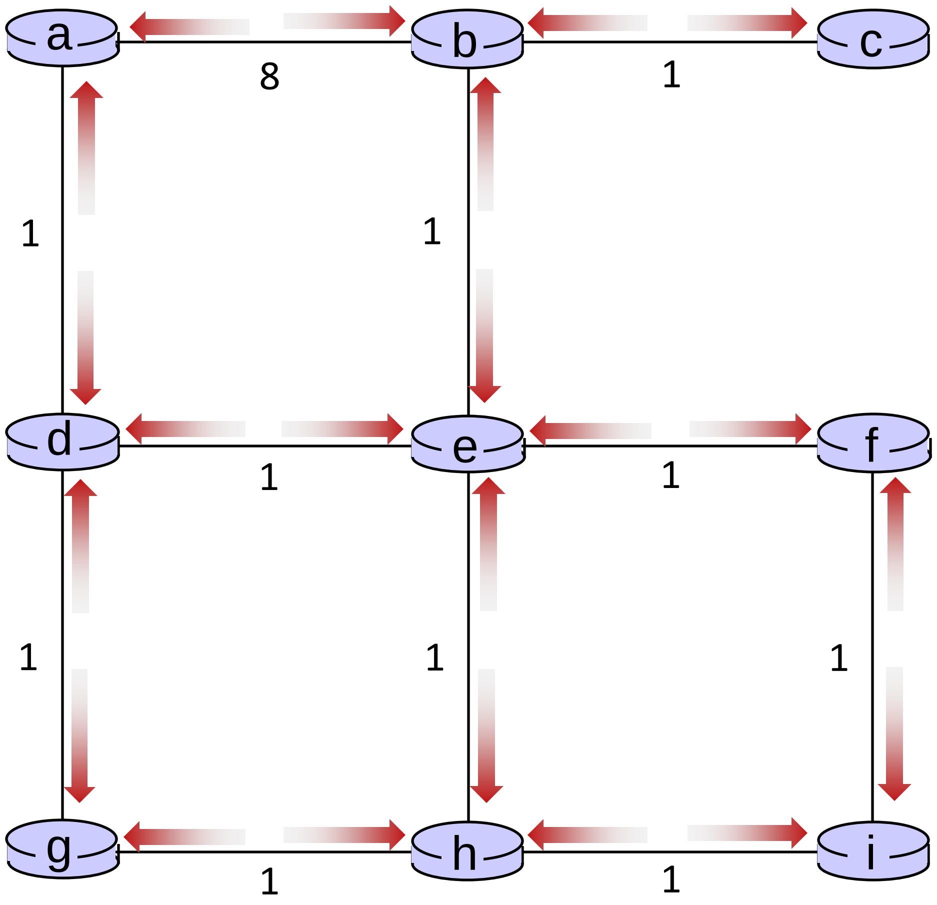
DV in a:
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



A few asymmetries:
- missing link
- larger cost

Distance vector example: iteration

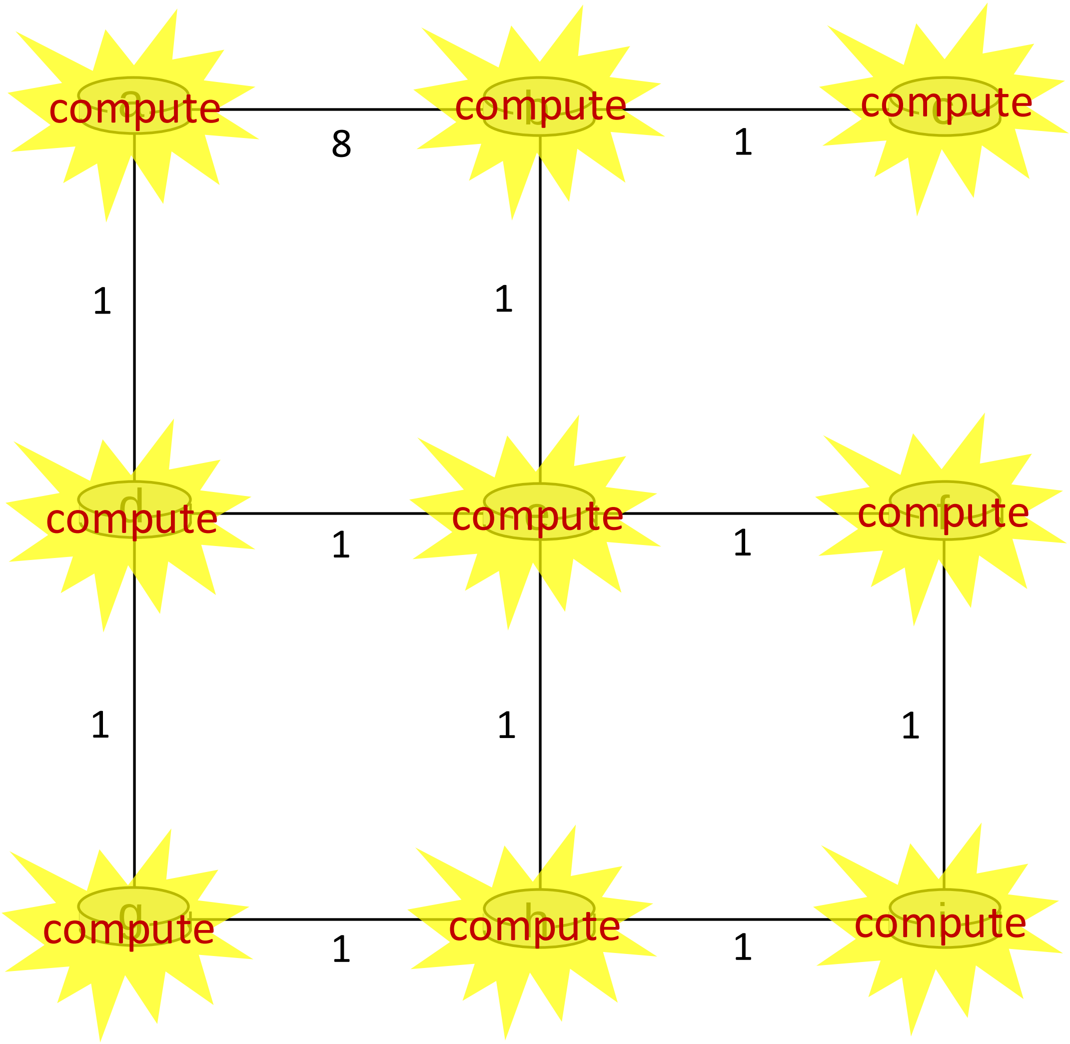
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

t=1

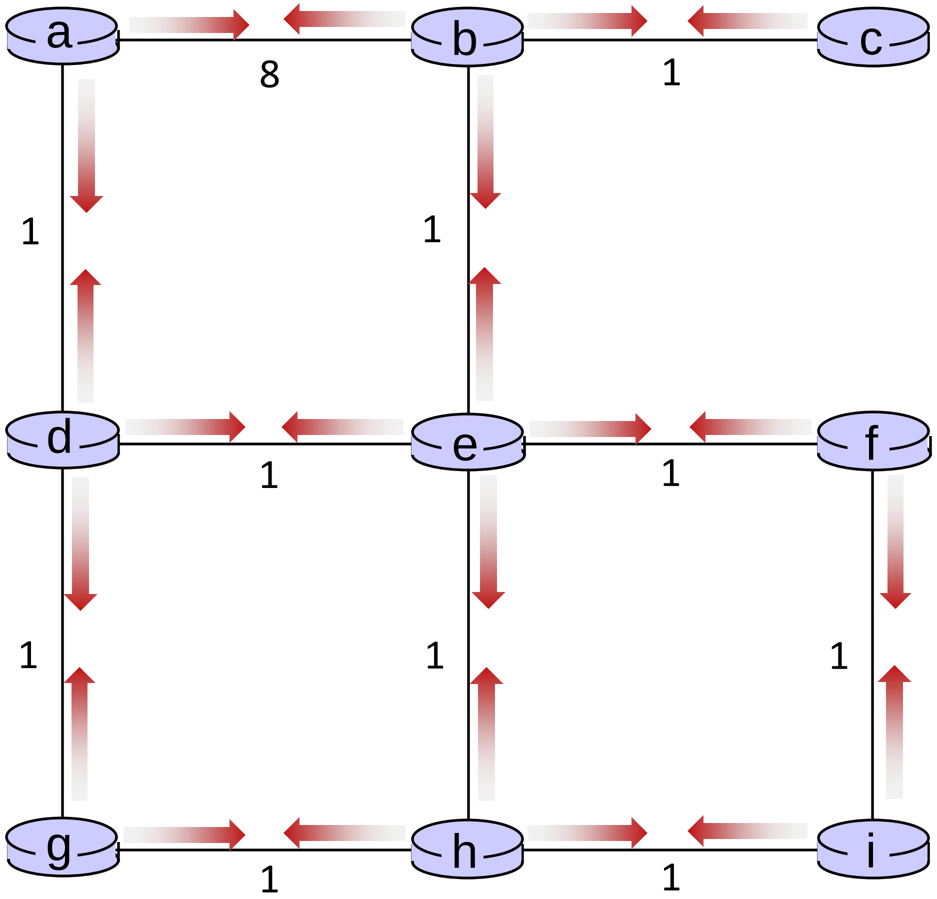
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



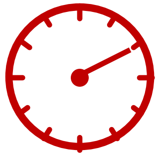
Distance vector example: iteration

t=1

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



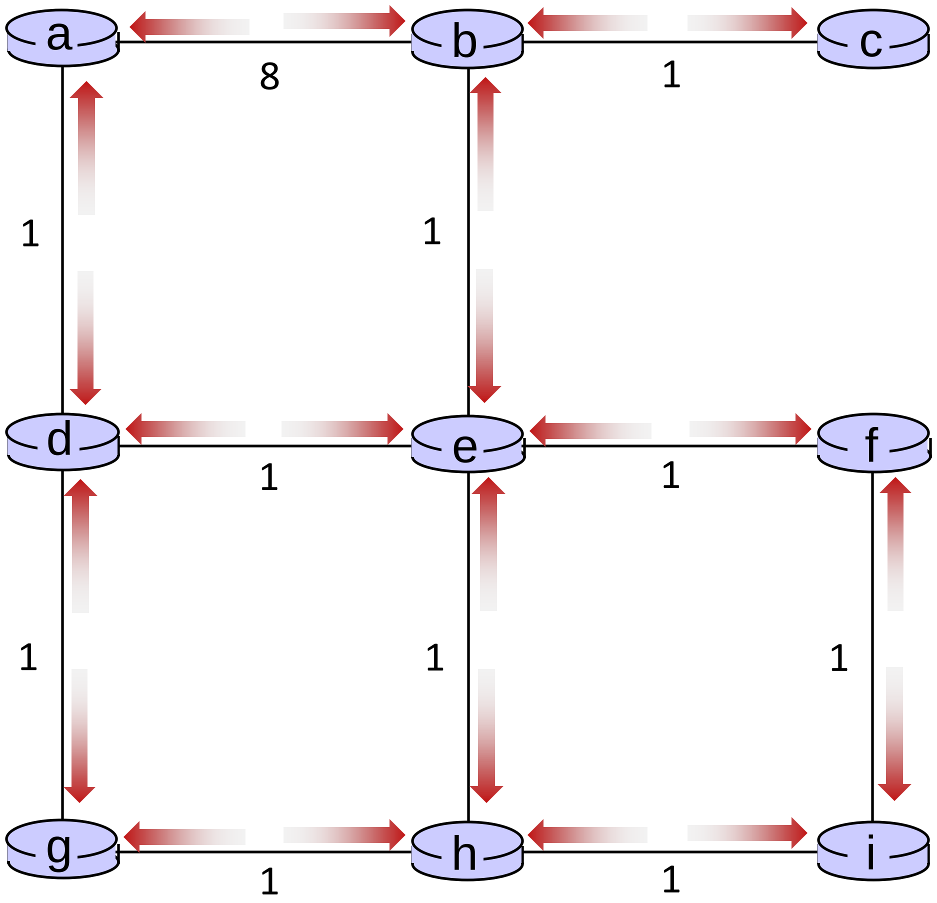
Distance vector example: iteration



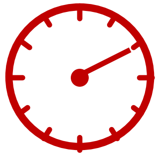
t=2

All nodes:

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



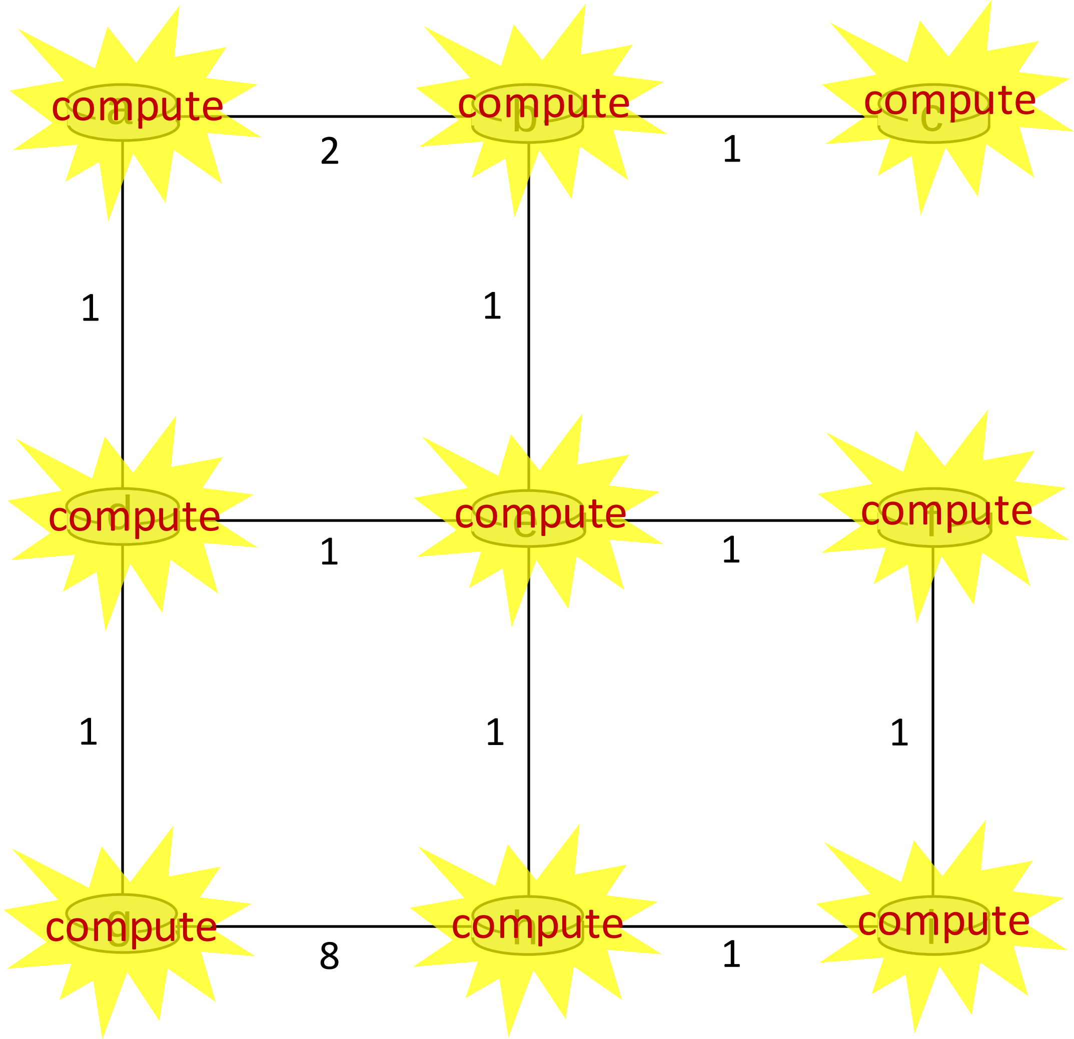
Distance vector example: iteration



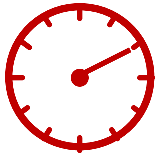
t=2

All nodes:

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

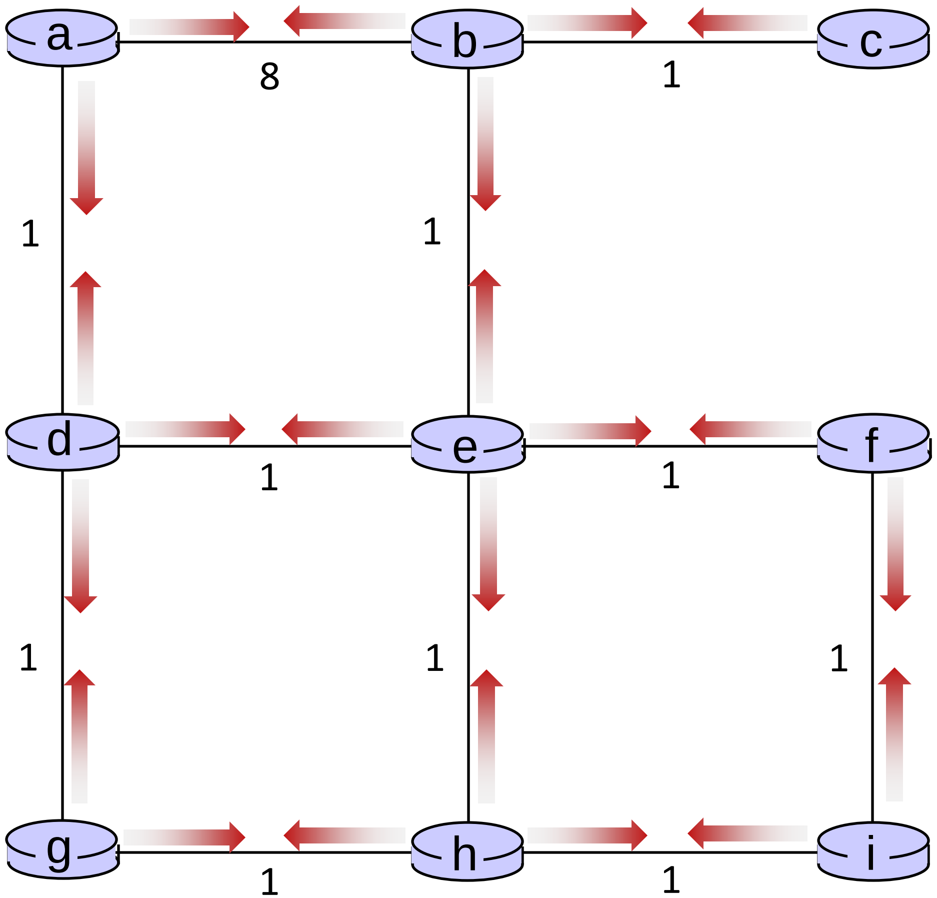


Distance vector example: iteration



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: state information diffusion

Iterative communication, computation steps diffuses information through network:

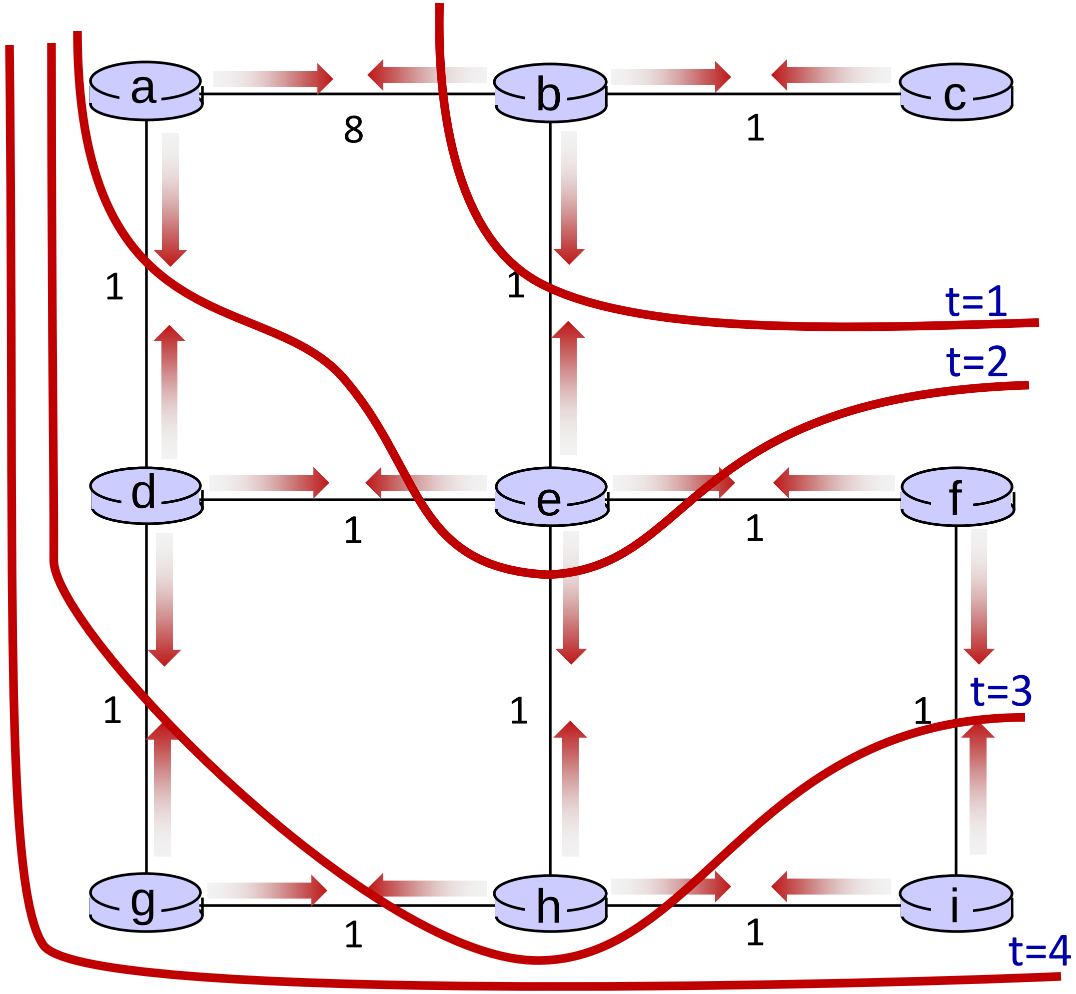
t=0 c's state at t=0 is at c only

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



Key difference between LS and DV

global: all routers have *complete* topology, link cost info

- "link state" algorithms

decentralized: iterative process of computation, exchange of info with neighbors

- routers initially only know link costs to attached neighbors
- "distance vector" algorithms