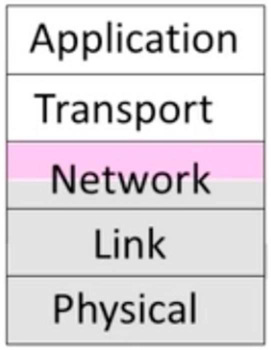
Computer Network Design

Network Layer III

Yalda Edalat – Spring 23

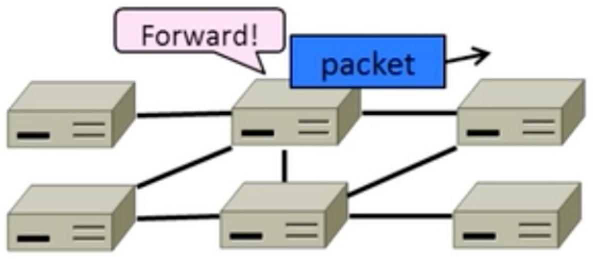
Where We Are?

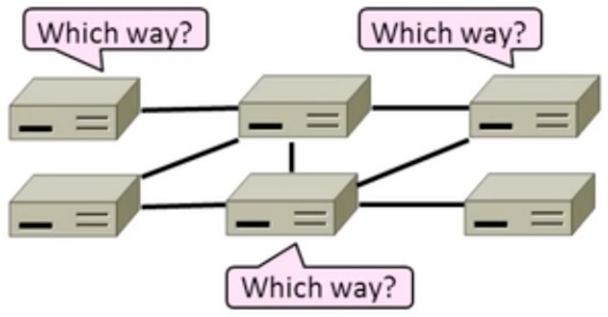
- We have covered packet forwarding
- Now We will learn about routing



Routing vs Forwarding

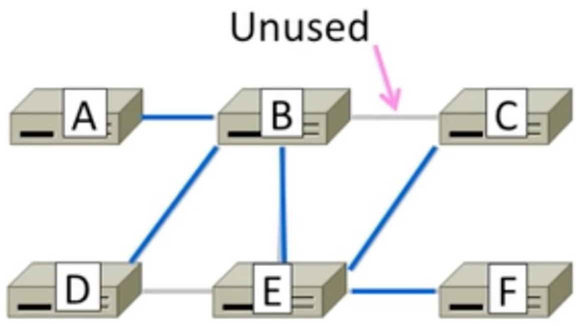
- Forwarding is the process of sending a packet on its way
- Routing is the process of deciding in which direction to send traffic

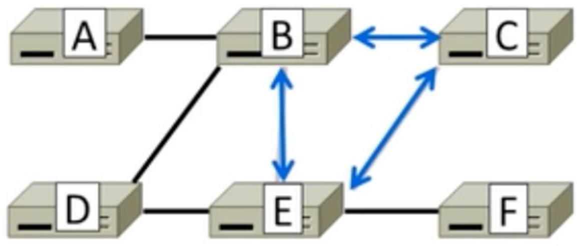




Improving on the Spanning Tree

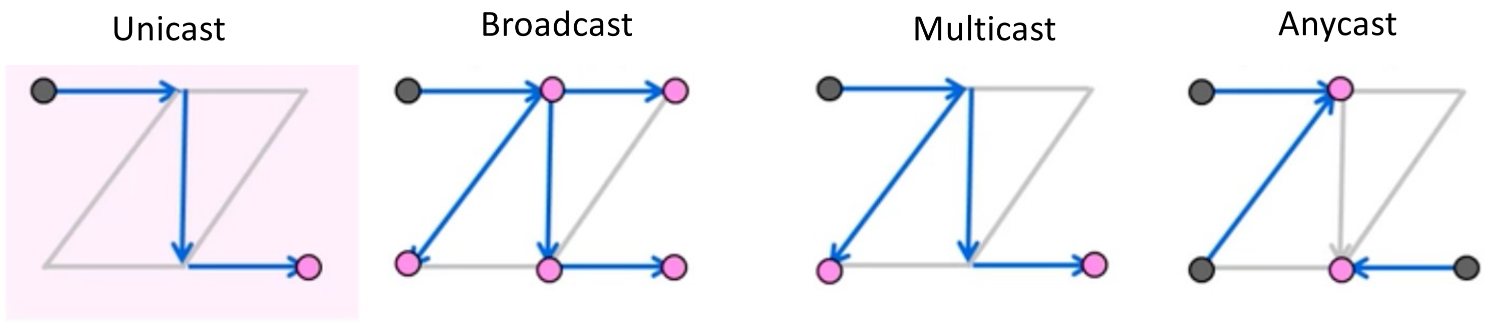
- Spanning tree provides basic connectivity
  - E.g., some path B->C
- Routing uses all links to find "best" paths
  - E.g., use BC, BE and CE





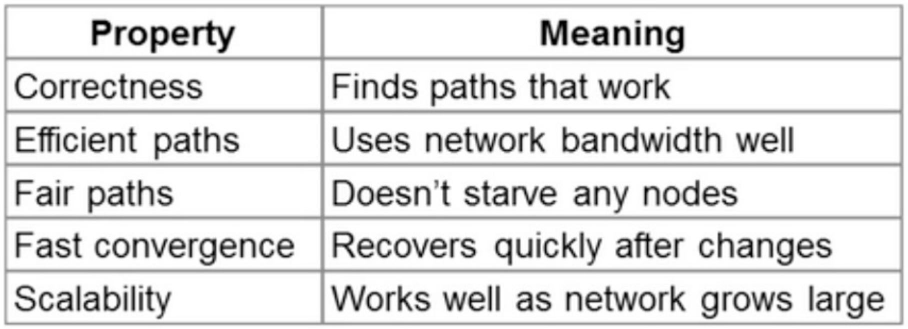
Delivery Methods

- Different routing used for different delivery methods



Goals of Routing Algorithms

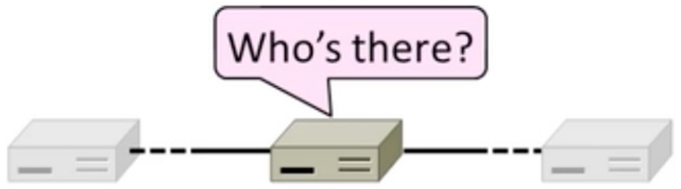
- We want several properties of any routing scheme:



Property	Meaning
Correctness	Finds paths that work
Efficient paths	Uses network bandwidth well
Fair paths	Doesn’t starve any nodes
Fast convergence	Recovers quickly after changes
Scalability	Works well as network grows large

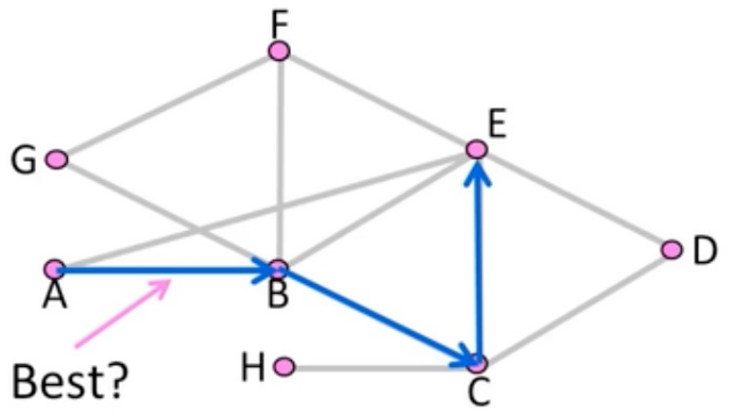
Rules of Routing Algorithms

- Decentralized, distributed setting
  - All nodes are alike; no controller
  - Nodes only know what they learn by exchanging messages with neighbors
  - Nodes operate concurrently
  - May be node/link/message failure



What are "Best" Paths Anyhow?

- Many possibilities:
  - Latency, avoid circuitous paths
  - Bandwidth, avoid slow links
  - Money, avoid expensive links
  - Hops, to reduce switching
- But only consider topology
  - Ignore workload, e.g., hotspots



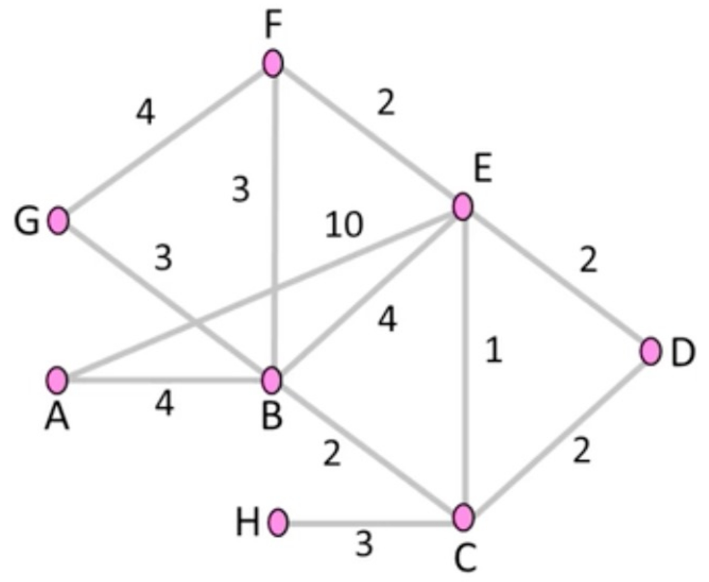
Shortest Paths

- We will approximate “best” by a cost function that captures the factors
  - Often call lowest “shortest”

1. Assign each link a cost (distance)
2. Define best path between each pair of nodes as the path that has the lowest total cost (or is shortest)
3. Pick randomly to any break tie

Shortest Paths (2)

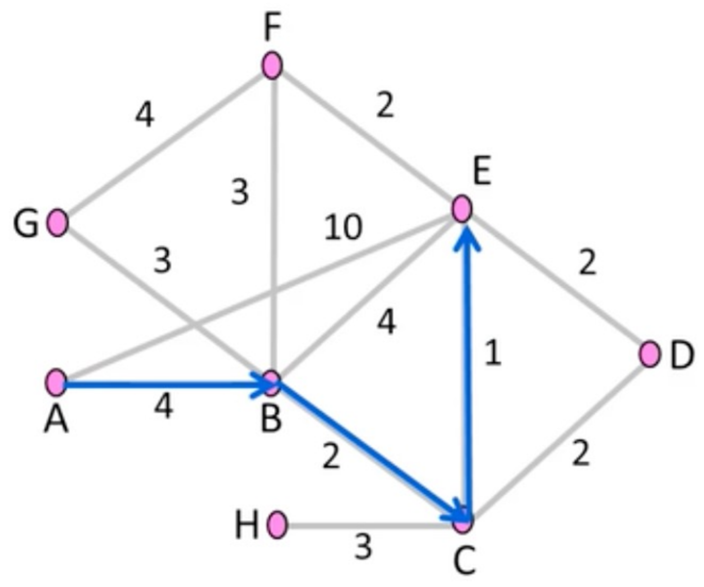
- Find the shortest path A->E
- All links are bidirectional, with equal costs direction
  - Can extend model to unequal costs if needed



Shortest Paths (3)

- ABCE is a shortest path
- Dist(ABCE) 4 + 2 +1 = 7

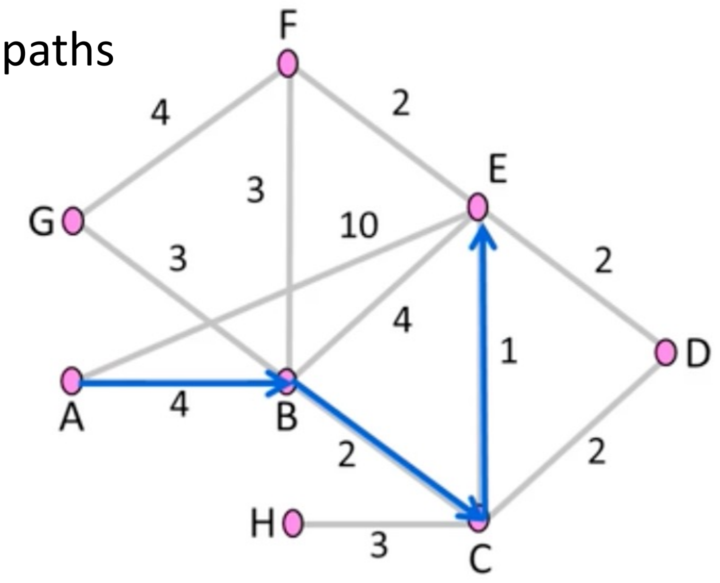
- This is less than:
  - Dist(ABE) = 8
  - Dist(ABFE) = 9
  - Dist(AE) = 10
  - Dist(ABCDE) = 10



Shortest Paths (4)

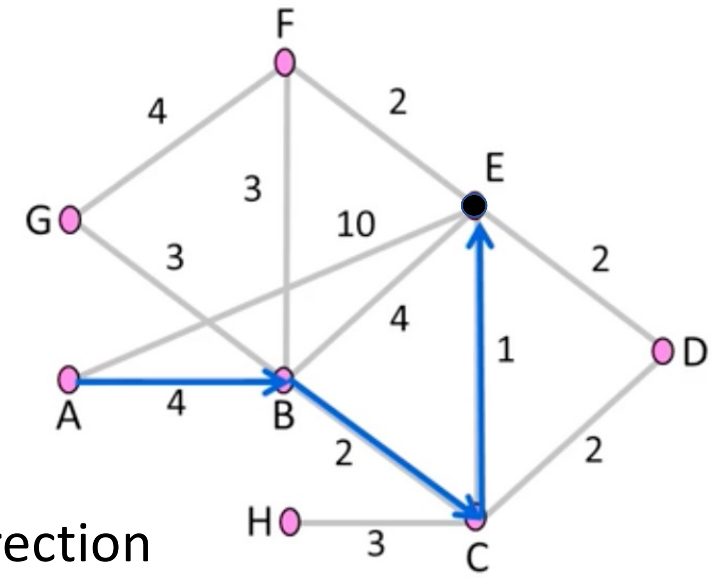
- Optimality property:
  - Subpaths of shortest paths are also shortest paths

  - ABCE is a shortest path
  - ->so are ABC, AB, BCE, BC, CE



Sink Trees

- Sink tree for a destination is the union of all shortest paths towards the destination
    - Find the sink tree for E

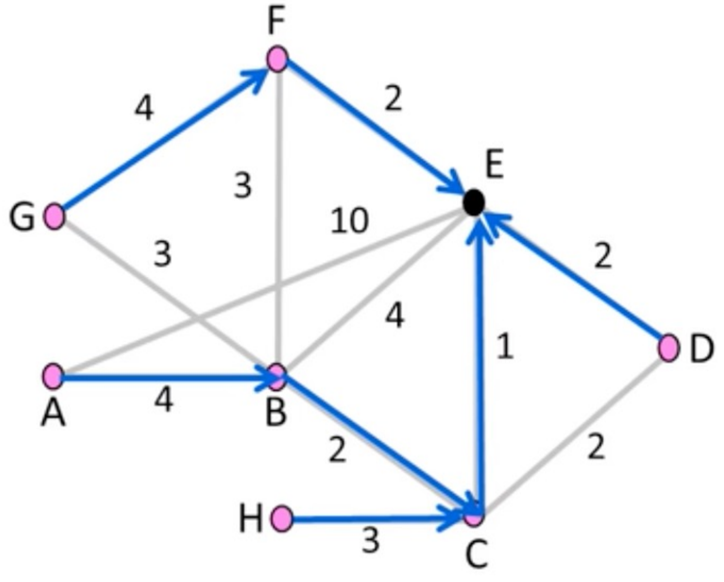


- Source tree is same tree with reverse direction

Sink Trees (2)

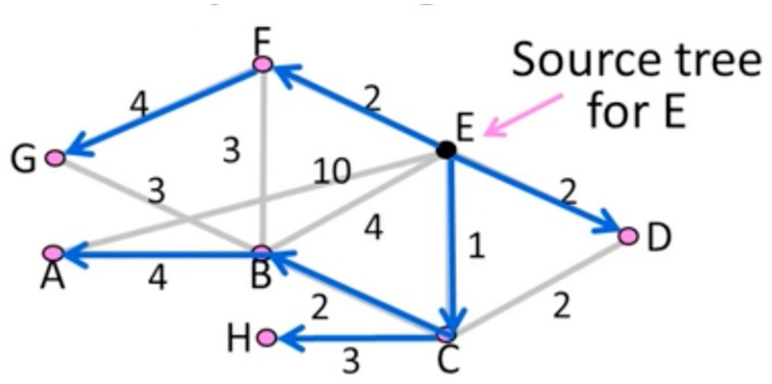
- Implications:
  - Only need to use destination to follow shortest paths
  - Each node only need to send to the next hop

- Forwarding table at a node
  - Lists next hop for each destination
  - Routing table may know more



Find Shortest Path

- How to compute shortest paths given the network topology
  - With Dijkstra’s algorithm

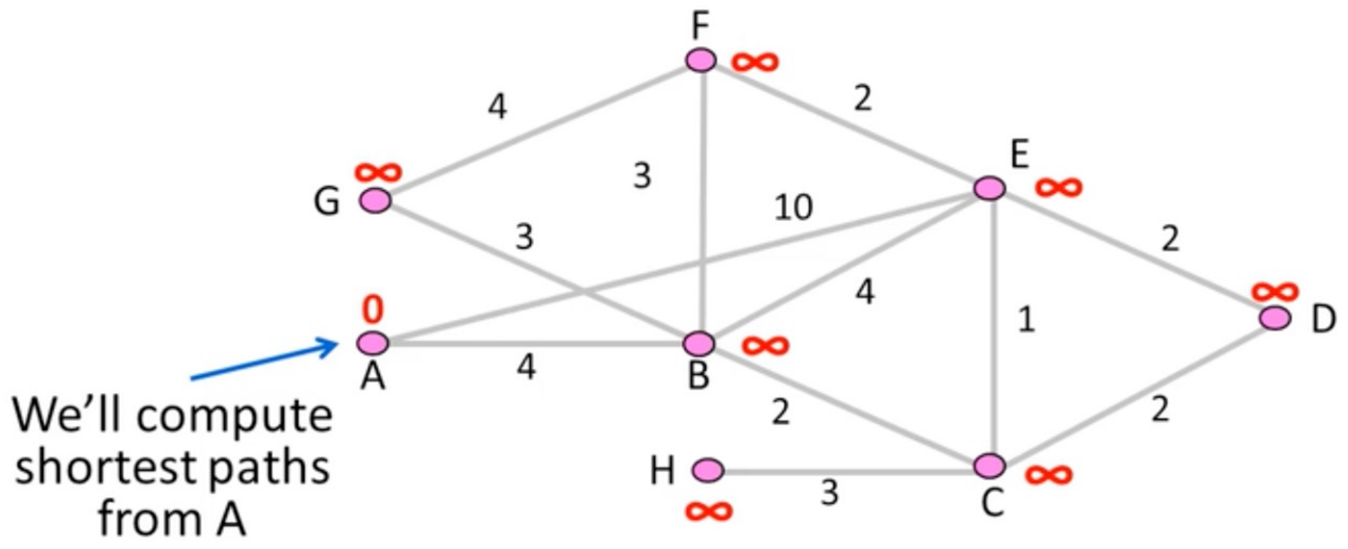


Dijkstra’s Algorithm

- Algorithm:
  - Mark all nodes tentative, set distances from source to 0 for source and ∞ for all other nodes
  - While tentative nodes remain:
    - Extract N, a node with lowest distance
    - Add link to N to the shortest path tree
    - Relax the distances of neighbors of N by lowering any better distance estimates

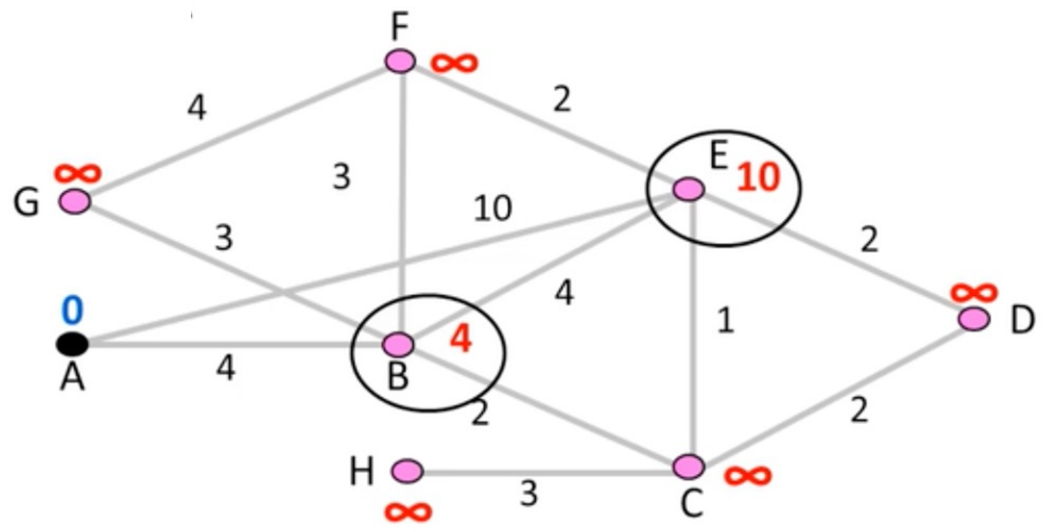
Dijkstra's Algorithm (2)

- Initialization



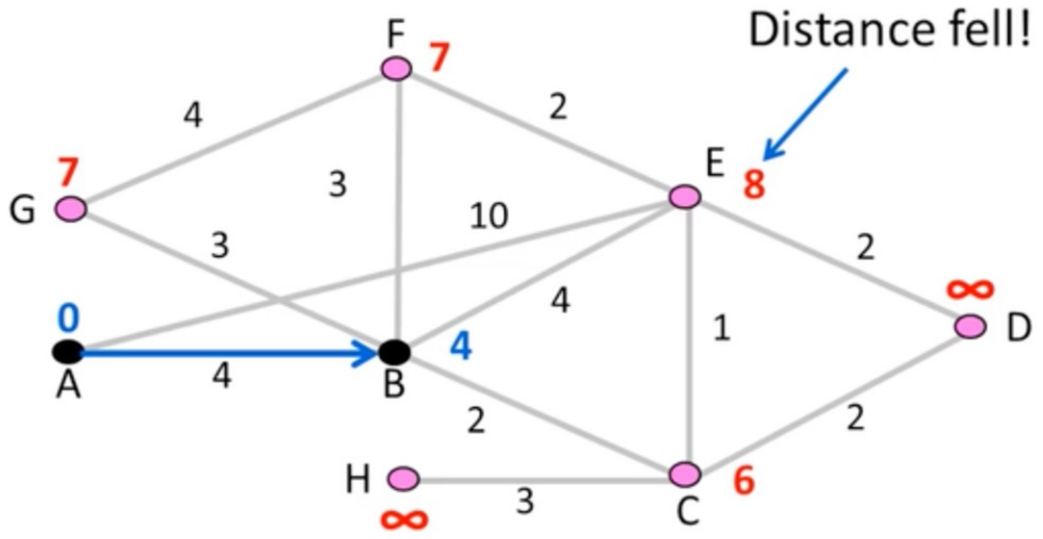
Dijkstra’s Algorithm (3)

- Relax around A



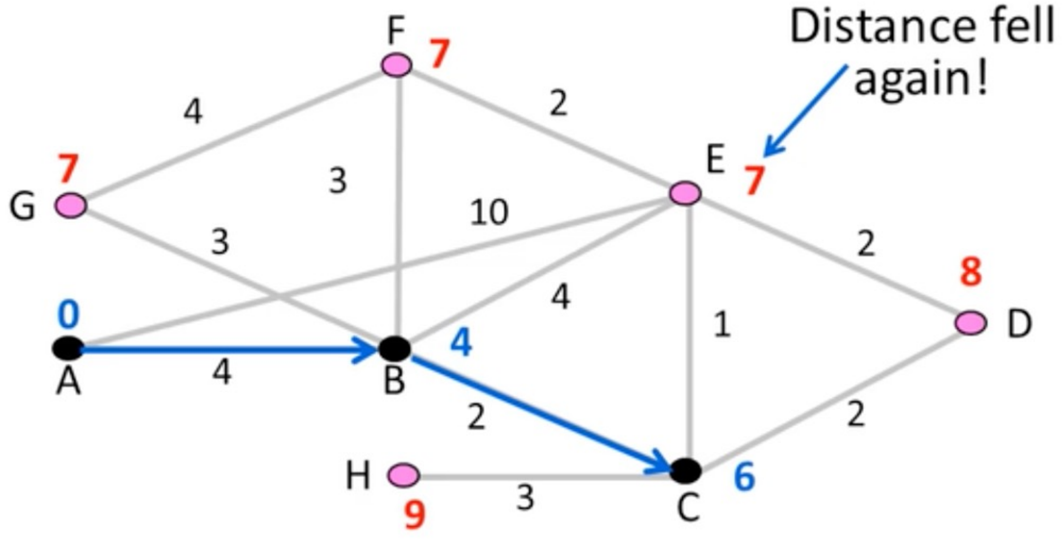
Dijkstra’s Algorithm (4)

- Relax around B



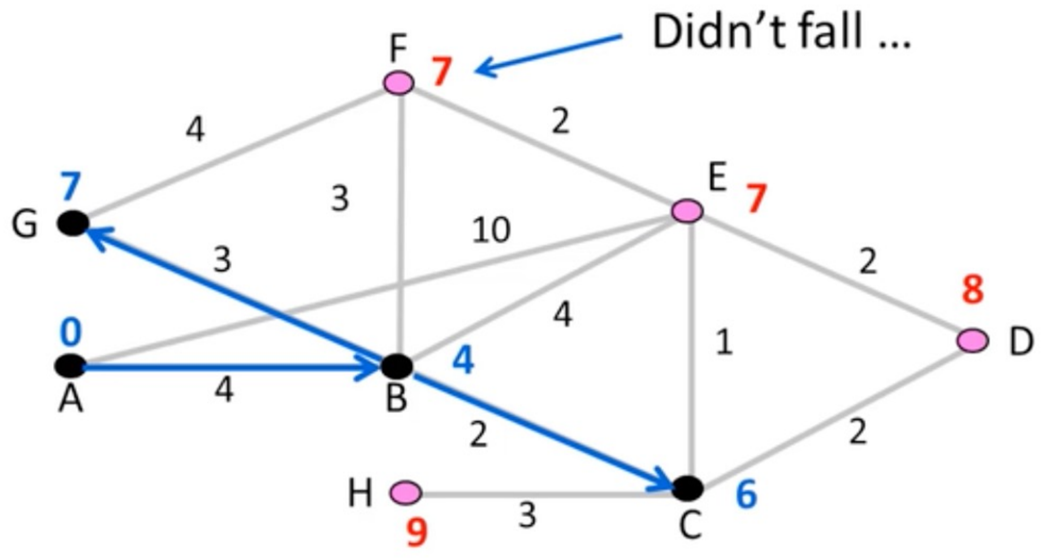
Dijkstra’s Algorithm (5)

- Relax around C



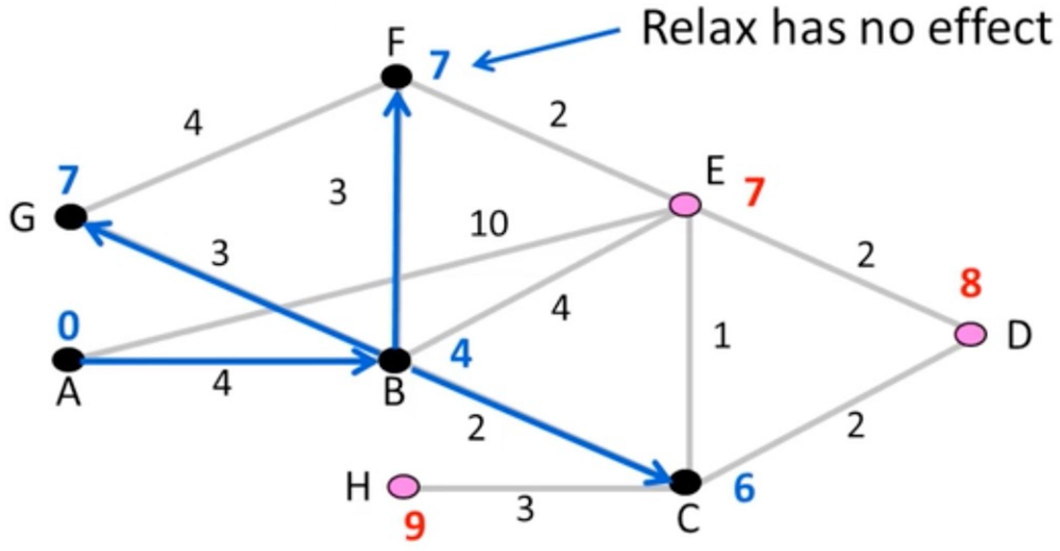
Dijkstra’s Algorithm (6)

- Relax around G



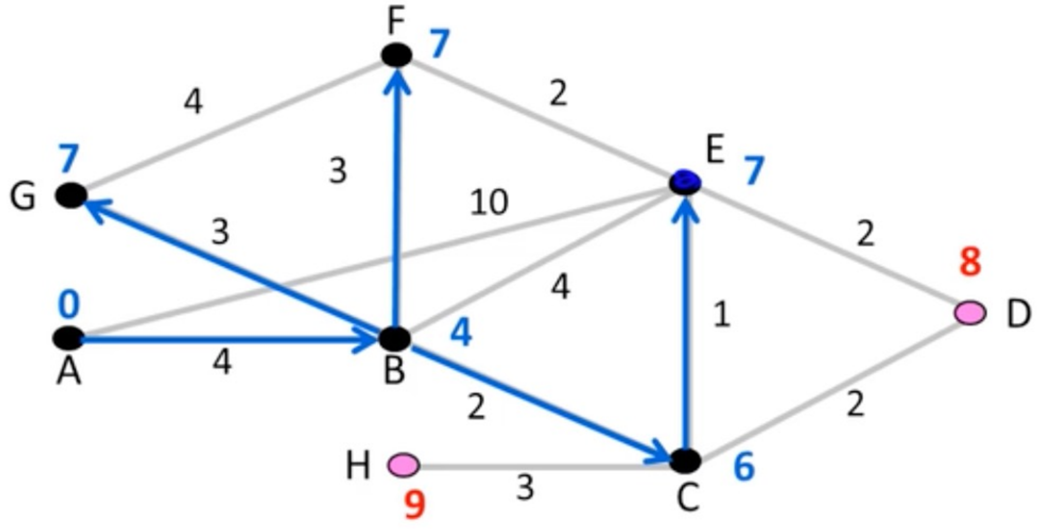
Dijkstra’s Algorithm (7)

- Relax around F



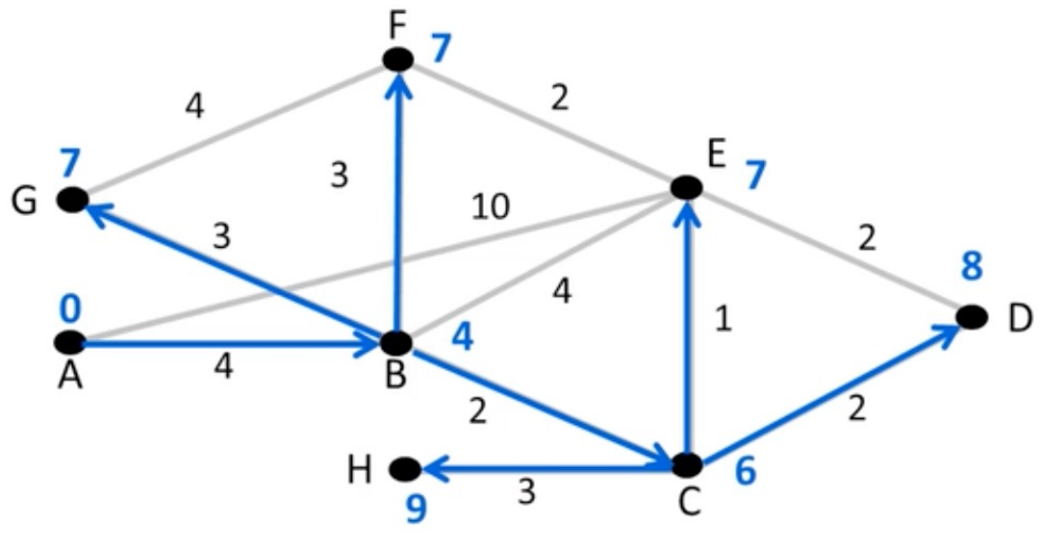
Dijkstra’s Algorithm (8)

- Relax around E



Dijkstra's Algorithm (10)

- Finally H, done!



Dijkstra Comments

- Finds shortest paths in order of increasing distance from source
  - Leverages optimality property
- Runtime depends on efficiency of extracting min-cost node
  - Superlinear in network size (grows fast)
- Gives complete source/sink tree
  - More than needed for forwarding!
  - But requires complete topology

Building a Routing Table

- Static routing
  - Manually fill the routing table

- Dynamic routing
  - Routing protocols are used. There are two classes:
    - **Distance Vector** routing protocols
      - RIP
      - IGRP
    - **Link State** routing protocols
      - OSPF
      - IS-IS

Distance Vector Routing

- Simple, early routing approach
- Works! But very slow convergence after failure
- Distributed version of Bellman-Ford algorithm

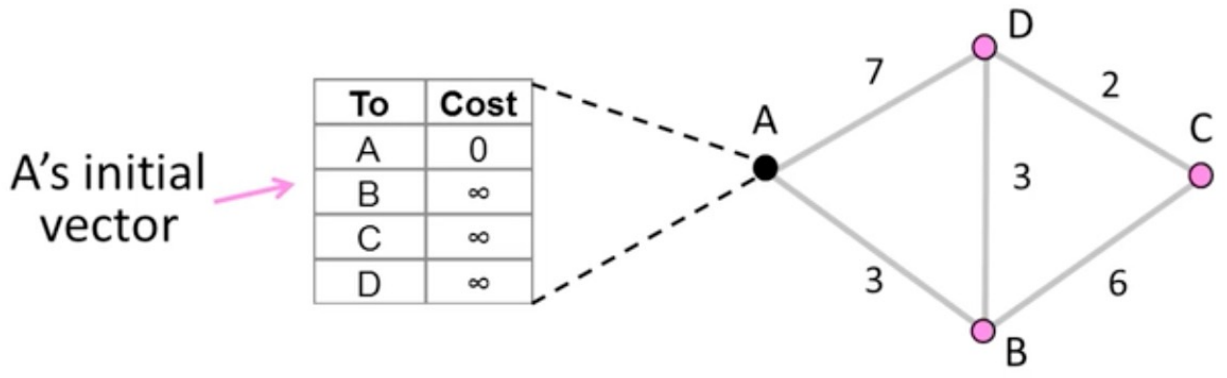
- Each node computes its forwarding table in a distributed setting:
    1. Nodes know only the cost to their neighbors; not the topology
    2. Nodes can talk only to their neighbors using messages
    3. All nodes run the same algorithm concurrently
    4. Nodes and links may fail, messages may be lost

Distance Vector Algorithm

- Each node maintains a vector of distances (and next hops) to all destinations
  1. Initialized vector with 0 cost to self and ∞ to other destinations
  2. Periodically send vector to neighbors
  3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
     - Use the best neighbor for forwarding

Distance Vector Example

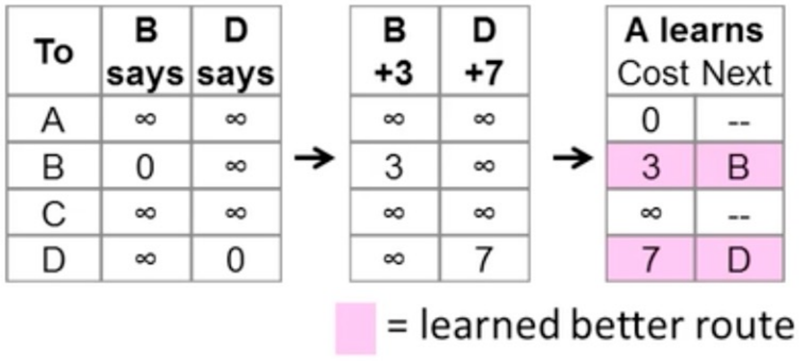
- Consider a simple network. Each node runs on its own
  - E.g., node A can only talk to nodes B and D



To	Cost
A	0
B	∞
C	∞
D	∞

Distance Vector Example (2)

- First exchange, A hears from B, D and finds 1-hop routes
  - A always learns min (B+3, D+7)



To	B says	D says
A	∞	∞
B	0	∞
C	∞	∞
D	∞	0

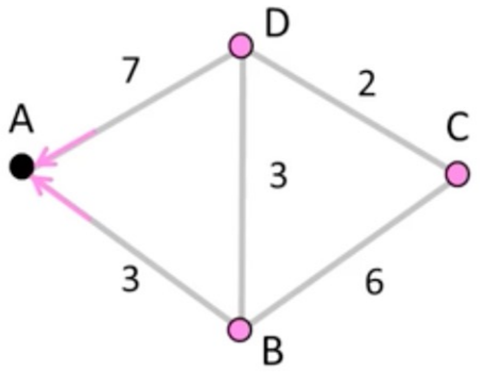
→

B +3	D +7
∞	∞
3	∞
∞	∞
∞	7

→

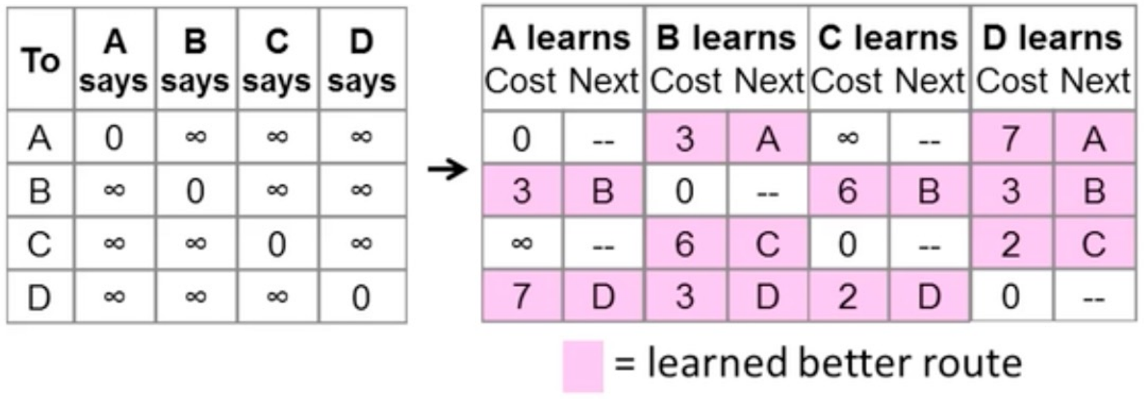
A learns	
Cost	Next
0	--
3	B
∞	--
7	D

▨ = learned better route



Distance Vector Example (3)

- First exchange for all nodes to find best 1-hop routes
  - E.g., B learns min (A+3, C+6, D+3)

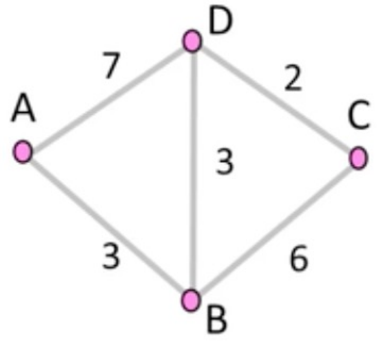


To	A says	B says	C says	D says
A	0	∞	∞	∞
B	∞	0	∞	∞
C	∞	∞	0	∞
D	∞	∞	∞	0

→

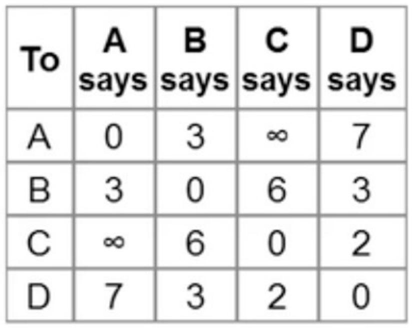
A learns		B learns		C learns		D learns	
Cost	Next	Cost	Next	Cost	Next	Cost	Next
0	--	3	A	∞	--	7	A
3	B	0	--	6	B	3	B
∞	--	6	C	0	--	2	C
7	D	3	D	2	D	0	--

■ = learned better route



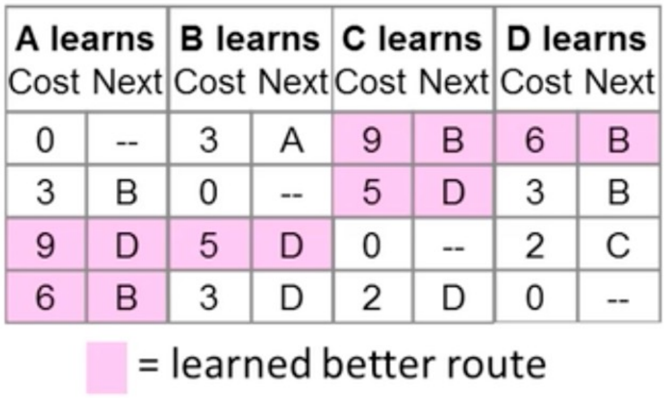
Distance Vector Example (4)

- Second exchange for all nodes to find best 2-hop routes



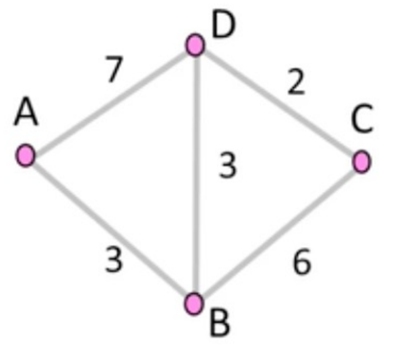
To	A says	B says	C says	D says
A	0	3	∞	7
B	3	0	6	3
C	∞	6	0	2
D	7	3	2	0

→



A learns		B learns		C learns		D learns	
Cost	Next	Cost	Next	Cost	Next	Cost	Next
0	--	3	A	9	B	6	B
3	B	0	--	5	D	3	B
9	D	5	D	0	--	2	C
6	B	3	D	2	D	0	--

🟪 = learned better route



Distance Vector Example (5)

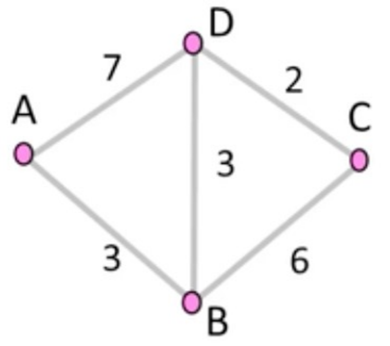
- Third exchange for all nodes to find best 3-hop routes

To	A says	B says	C says	D says
A	0	3	9	6
B	3	0	5	3
C	9	5	0	2
D	6	3	2	0

→

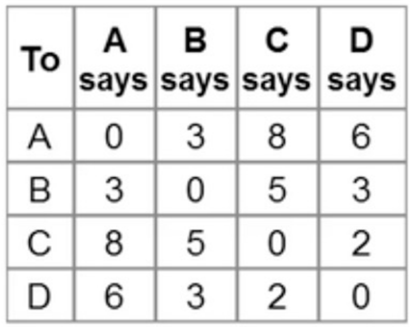
A learns		B learns		C learns		D learns	
Cost	Next	Cost	Next	Cost	Next	Cost	Next
0	--	3	A	8	D	6	B
3	B	0	--	5	D	3	B
8	B	5	D	0	--	2	C
6	B	3	D	2	D	0	--

▨ = learned better route



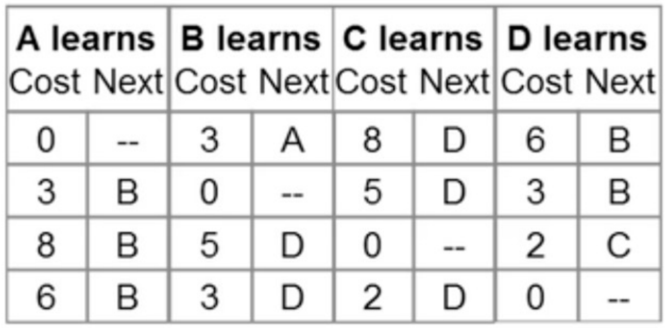
Distance Vector Example (6)

- Fourth and subsequent exchanges; converged



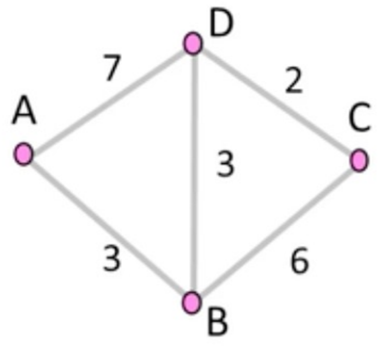
To	A says	B says	C says	D says
A	0	3	8	6
B	3	0	5	3
C	8	5	0	2
D	6	3	2	0

→



A learns		B learns		C learns		D learns	
Cost	Next	Cost	Next	Cost	Next	Cost	Next
0	--	3	A	8	D	6	B
3	B	0	--	5	D	3	B
8	B	5	D	0	--	2	C
6	B	3	D	2	D	0	--

🟪 = learned better route



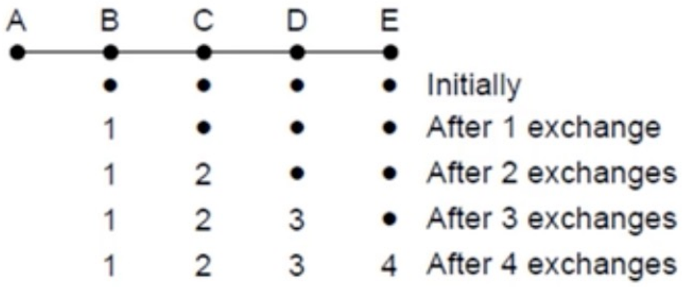
Distance Vector Dynamics

- Adding routes:
  - News travels one hop per exchange
- Removing routes:
  - When node fails, no more exchange, other nodes forget

- But <u>partitions</u> (unreachable nodes in divided network) are a problem
  - “Count to infinity” scenario

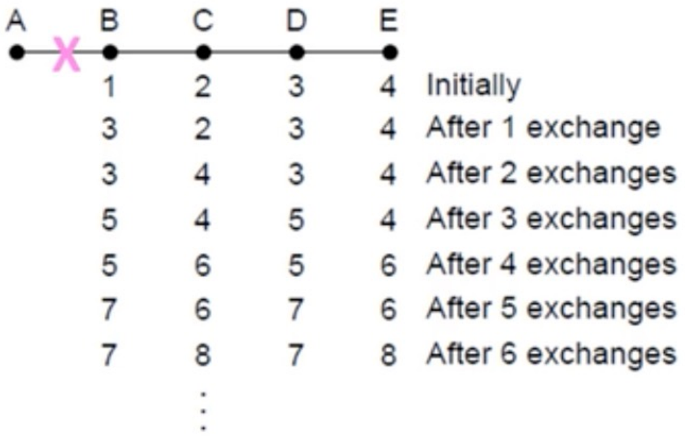
Distance Vector Dynamics (2)

- Good news travels quickly, bad news slowly (inferred)



A	B	C	D	E	
	•	•	•	•	Initially
	1	•	•	•	After 1 exchange
	1	2	•	•	After 2 exchanges
	1	2	3	•	After 3 exchanges
	1	2	3	4	After 4 exchanges

Desired convergence



A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			

“Count to infinity” scenario

Distance Vector Dynamics (3)

- Various heuristics to address
  - E.g., “Split horizon, poison reverse” (Don’t send route back to where you learned it from)
- But none are very effective
  - Link state now favored in practice
  - Except when very resource-limited

RIP (Routing Information Protocol)

- DV protocol with hop count as metric
  - Infinity is 16 hops; limits network size
  - Includes split horizon, poison reverse
- Routers send vectors every 30 seconds
  - Runs on top of UDP
  - Timeout in 180 secs to detect failures

Link-State Routing

- One of the two approaches to routing
  - Trades more computation than distance vector for better dynamics
  - Widely used in practice (used in Internet/ARPANET from 1979)

Link-State Setting

- Nodes compute their forwarding table in the same distributed setting as for distance vector:
    1. Nodes know only the cost to their neighbors; not the topology
    2. Nodes can talk only to their neighbors using messages
    3. All nodes run the same algorithm concurrently
    4. Nodes/links may fail, messages may be lost

Link-State Algorithm

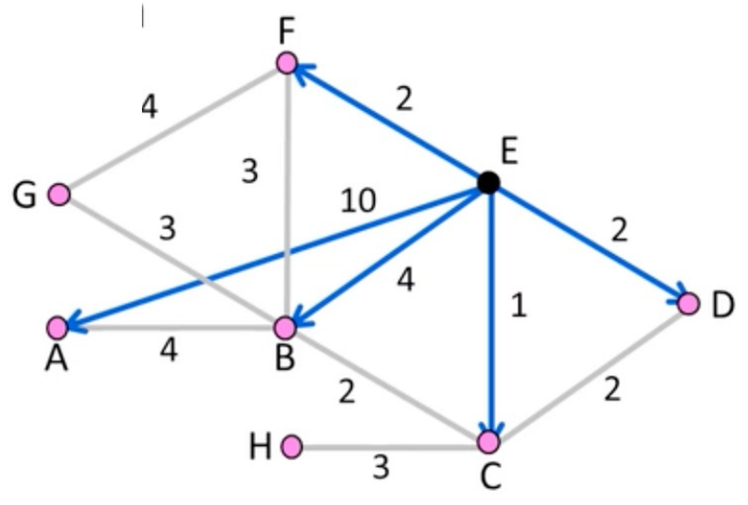
- Proceeds in two phases:
  1. Nodes flood topology in the form of link state packets
     - Each node learns full topology
  2. Each node computes its own forwarding table
     - By running Dijkstra

Phase 1: Topology Dissemination

- Each node floods link state packet (LSP) that describes their portion of the topology

Node E's LSP flooded to A, B, C, D, and F

Seq. #	
A	10
B	4
C	1
D	2
F	2

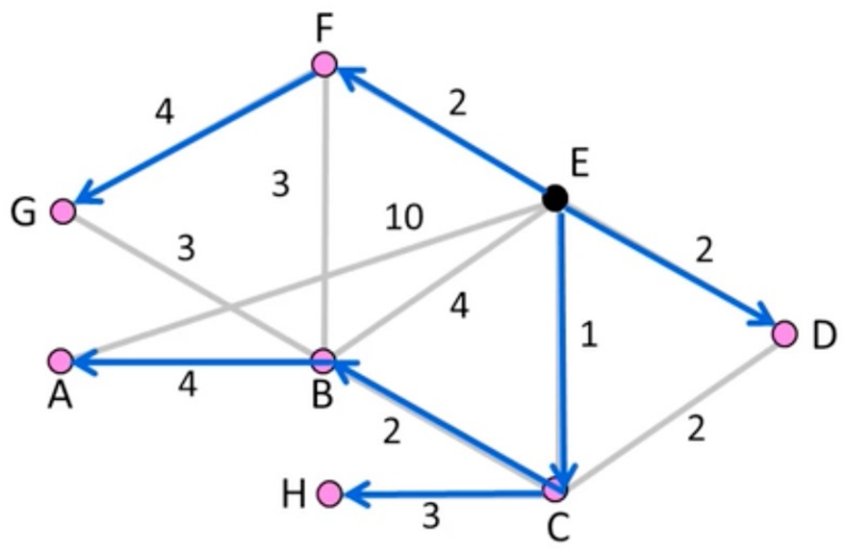


Phase 2: Route Computation

- Each node has full topology
  - By combining all LSPs
- Each node simply runs Dijkstra
  - Some replicated computation, but finds required routes directly
  - Compile forwarding table from sink/source tree
  - That is it!

Forwarding Table

Source Tree for E (from Dijkstra)



E's Forwarding Table

To	Next
A	C
B	C
C	C
D	D
E	--
F	F
G	F
H	C

Handling Changes

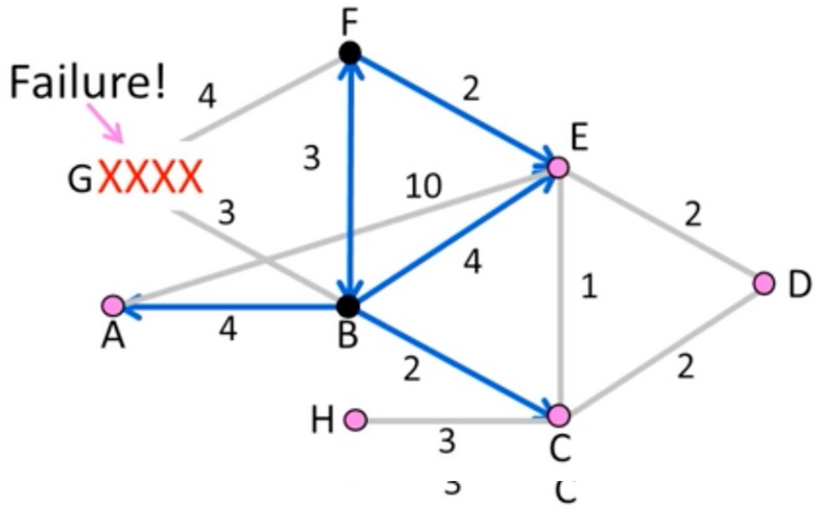
- On change, flood updated LSPs, and re-compute routes
  - E.g., nodes adjacent to failed link or node initiate

B’s LSP

	Seq. #
A	4
C	2
E	4
F	3
G	∞

F’s LSP

	Seq. #
B	3
E	2
G	∞



Handling Changes (2)

- Link failure
    - Both nodes notice, send updated LSPs
    - Link is removed from topology
- Node failure
    - All neighbors notice a link has failed
    - Failed node cant update its own LSP
    - But it is OK: all links to node removed

Handling Changes (3)

- Addition of a link or node
  - Add LSP of new node to topology
  - Old LSPs are updated with new link
- Additions are the easy case

Link-State Complications

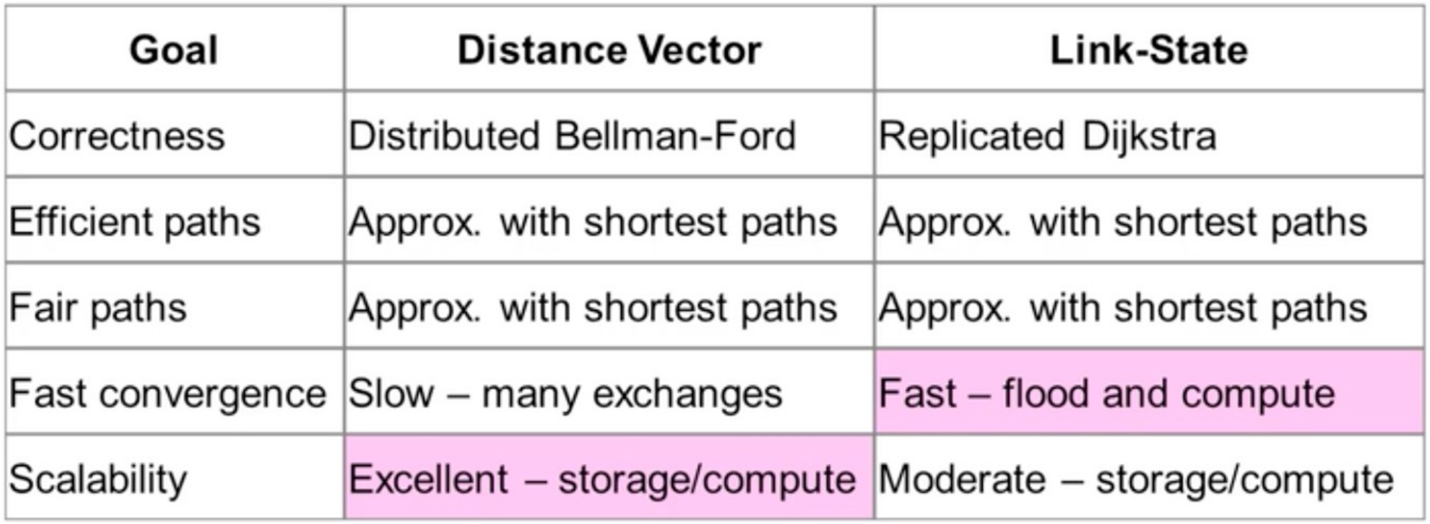
- Things that can go wrong:
  - Seq. number reaches max, or is corrupted
  - Node crashes and loses seq. number
  - Network partitions then heals

- Strategy:
  - Include age on LSPs and forget old information that is not refreshed

IS-IS and OSPF

- Widely used in large enterprise and ISP networks
  - IS-IS = Intermediate System to Intermediate System
  - OSPF = Open Shortest Path First

DV/LS Comparison



Goal	Distance Vector	Link-State
Correctness	Distributed Bellman-Ford	Replicated Dijkstra
Efficient paths	Approx. with shortest paths	Approx. with shortest paths
Fair paths	Approx. with shortest paths	Approx. with shortest paths
Fast convergence	Slow – many exchanges	Fast – flood and compute
Scalability	Excellent – storage/compute	Moderate – storage/compute

To-do

- Quiz next week
- Lab2 due date is on 15th