Recap of routing thus far

Distributed routing, nodes exchange knowledge of destinations

But dealing with individual destinations does not scale

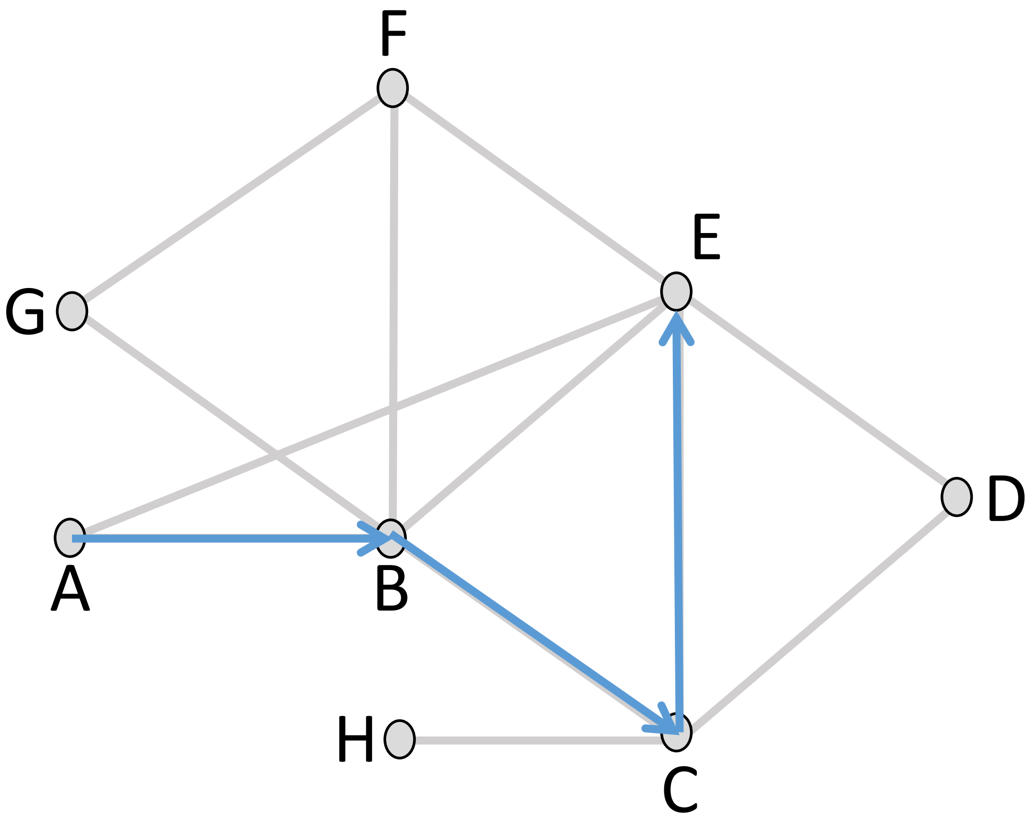
Scaling techniques

- Hierarchical routing – at coarser granularity
- Aided by prefix structure
  - Subnetting – break coarser prefixes into granular ones while allocating them
  - Aggregation – combine granular prefixes into coarser ones

Finding “Best” Paths

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'll approximate "best" by a cost function that captures the factors

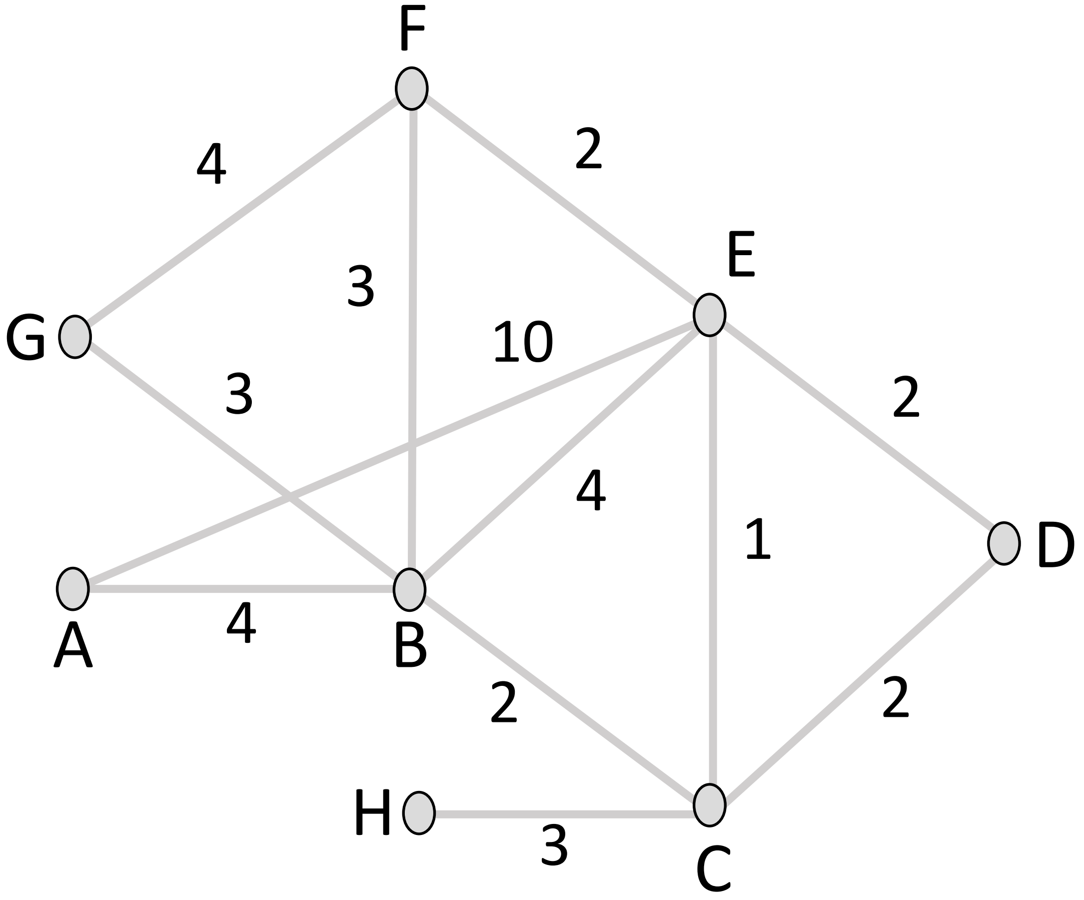
- Often called "least cost" or "shortest"

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

Shortest Paths (2)

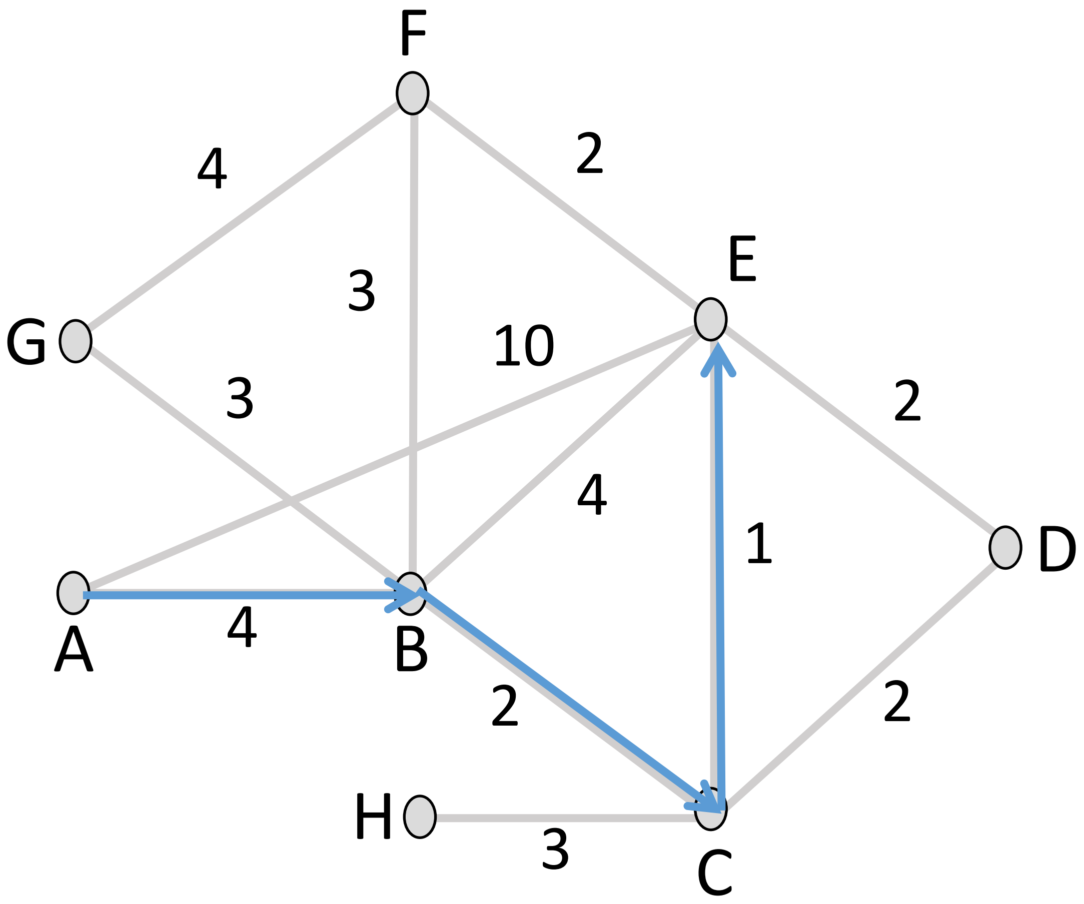
- Find the shortest path A → E

- All links are bidirectional, with equal costs in each direction
  - Can extend model to unequal costs if needed



Shortest Paths (3)

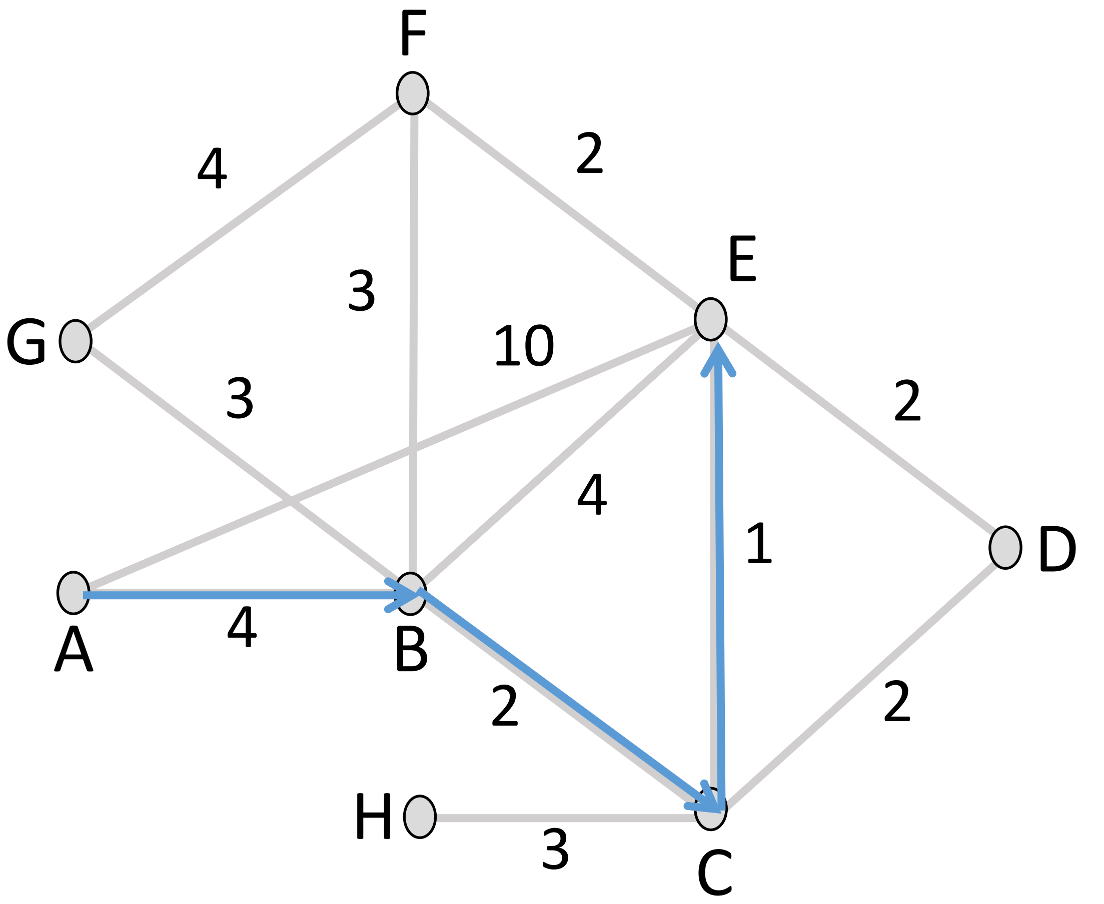
- ABCE is a shortest path
  - cost(ABCE) = 4 + 2 + 1 = 7
- It is shorter than:
  - cost(ABE) = 8
  - cost(ABFE) = 9
  - cost(AE) = 10
  - cost(ABCDE) = 10



Shortest Paths (4)

- Optimality property:
  - Subpaths of shortest paths are also shortest paths
- ABCE is a shortest path

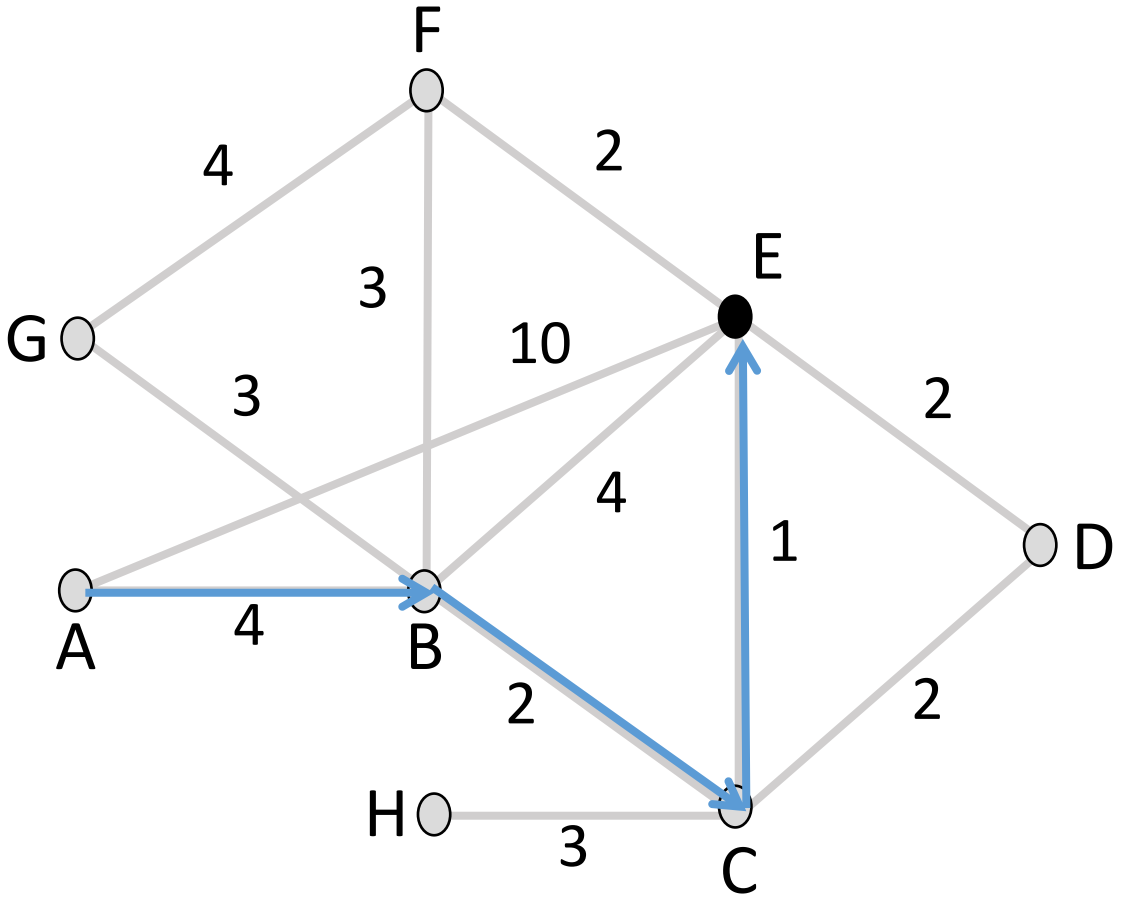
  →So are ABC, AB, BCE, BC, CE



Sink Trees

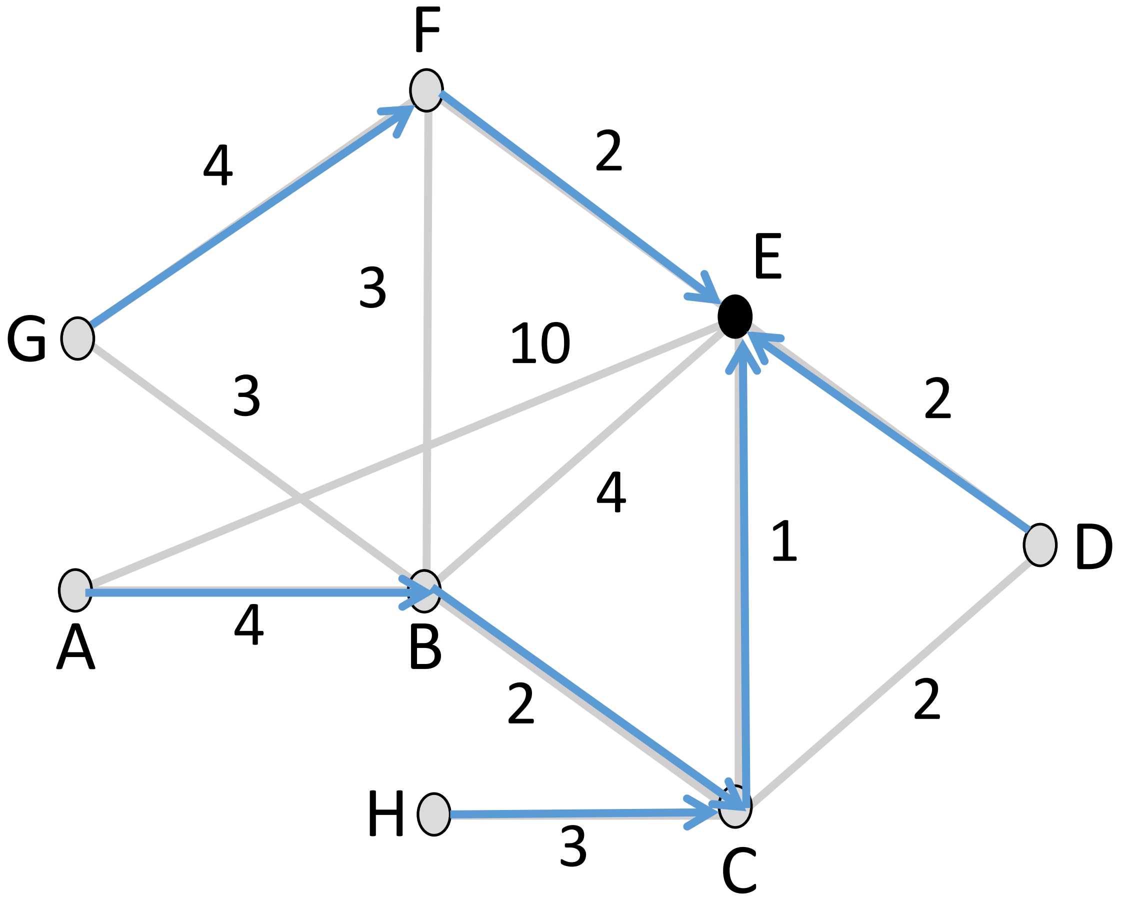
- Sink tree for a destination is the union of all shortest paths towards the destination
  - Similarly source tree

- Find the sink tree for E



Sink Trees (2)

- Implications:
  - Only need to use destination to follow shortest paths
  - Each node only need to send to the next hop
- <u>Forwarding table</u> at a node
  - Lists next hop for each destination
  - Routing table may know more



Distance Vector Routing

Distance Vector Routing

- Simple, early routing approach
  - Used in ARPANET, and RIP
- One of two main approaches to routing
  - Distributed version of Bellman-Ford
  - Works, but very slow convergence after some failures
- Link-state algorithms are now typically used in practice
  - More involved, better behavior

Distance Vector Setting

Each node computes its forwarding table in a distributed setting:

1. Nodes know only the cost to their neighbors; not 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 (distance, next hop) to all destinations

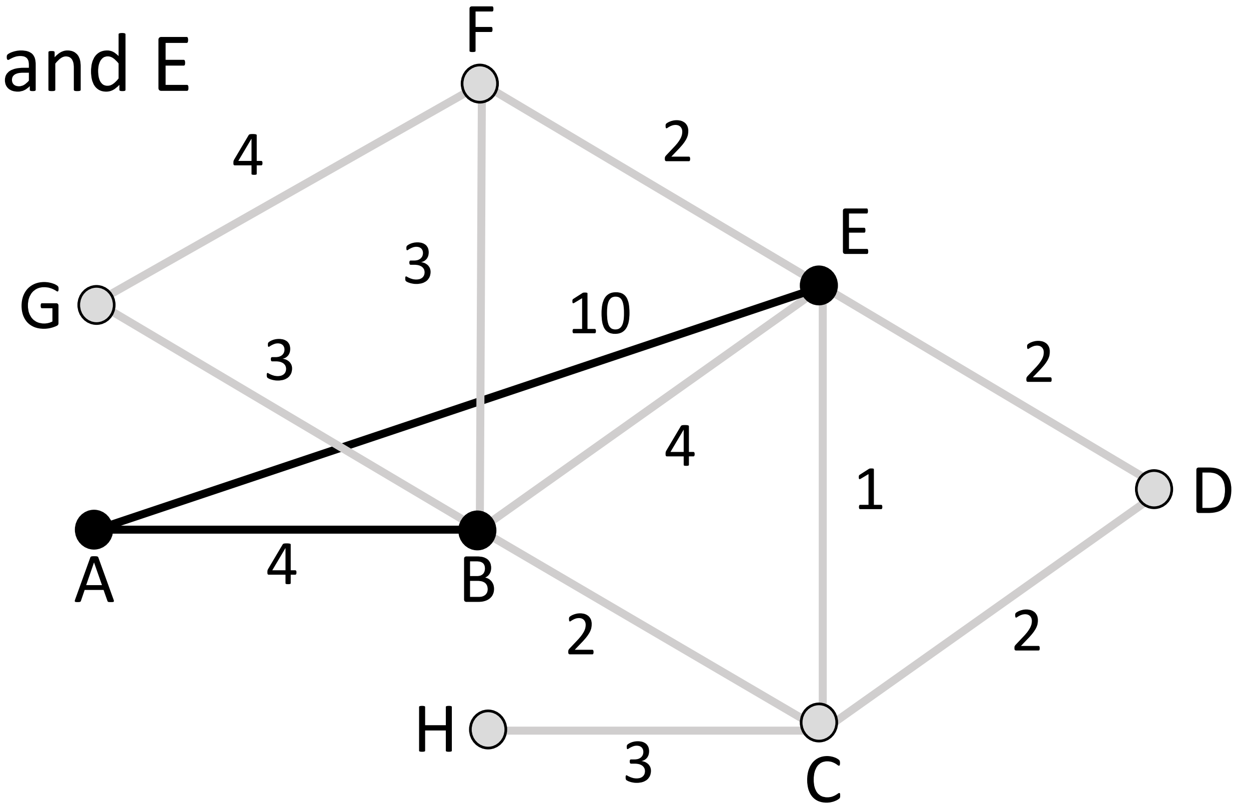
1. Initialize vector with 0 (zero) cost to self, ∞ (infinity) to other destinations
2. Periodically send vector to neighbors
3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
4. Use the best neighbor for forwarding

Distance Vector (2)

- Consider from the point of view of node A
  - Can only talk to nodes B and E

Initial vector

To	Cost
A	0
B	∞
C	∞
D	∞
E	∞
F	∞
G	∞
H	∞

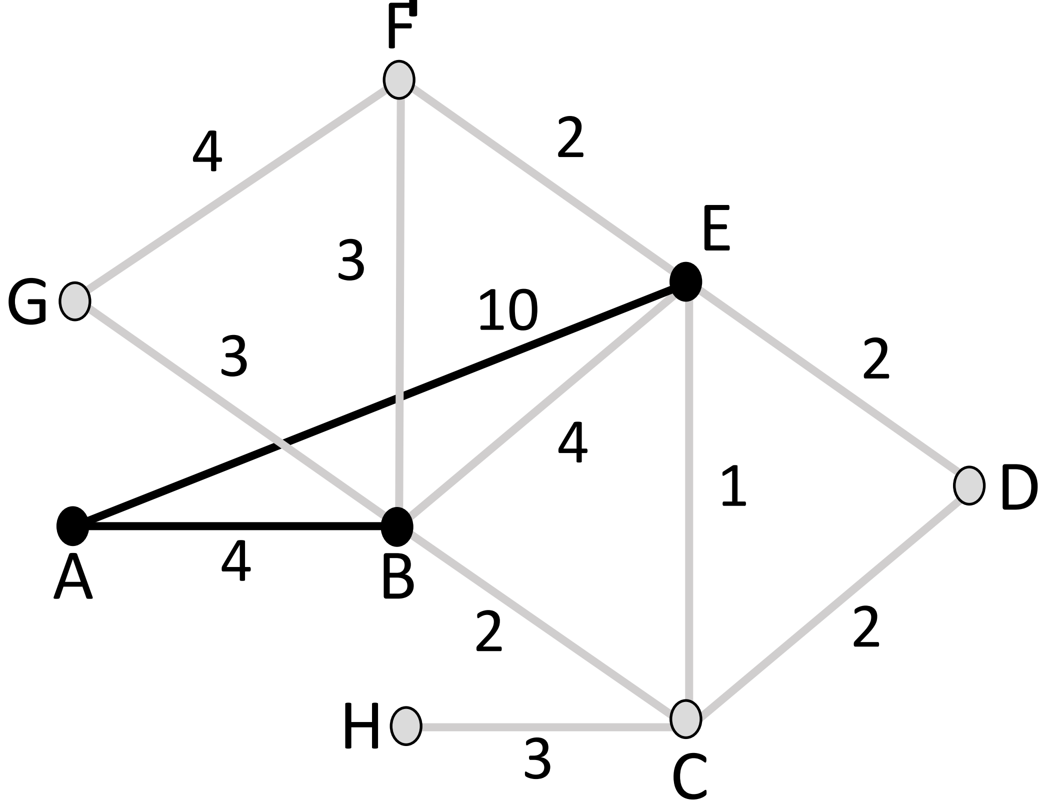


Distance Vector (3)

- First exchange with B, E; learn best 1-hop routes

To	B says	E says		B +4	E +10		A's Cost	A's Next
A	∞	∞		∞	∞		0	--
B	0	∞		4	∞		4	B
C	∞	∞	→	∞	∞	→	∞	--
D	∞	∞		∞	∞		∞	--
E	∞	0		∞	10		10	E
F	∞	∞		∞	∞		∞	--
G	∞	∞		∞	∞		∞	--
H	∞	∞		∞	∞		∞	--

Learned better route



Distance Vector (4)

- Second exchange; learn best 2-hop routes

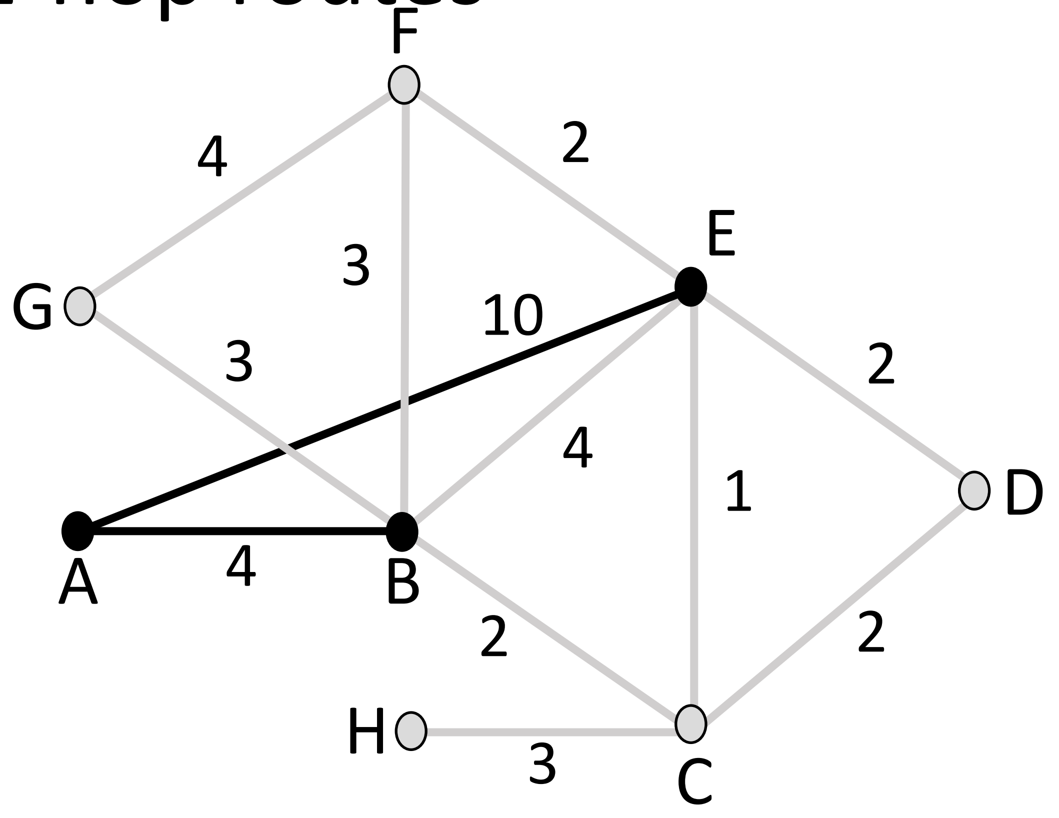
To	B says	E says
A	4	10
B	0	4
C	2	1
D	∞	2
E	4	0
F	3	2
G	3	∞
H	∞	∞

→

B +4	E +10
8	20
4	14
6	11
∞	12
8	10
7	12
7	∞
∞	∞

→

A's Cost	A's Next
0	--
4	B
6	B
12	E
8	B
7	B
7	B
∞	--



Distance Vector (4)

- Third exchange; learn best 3-hop routes

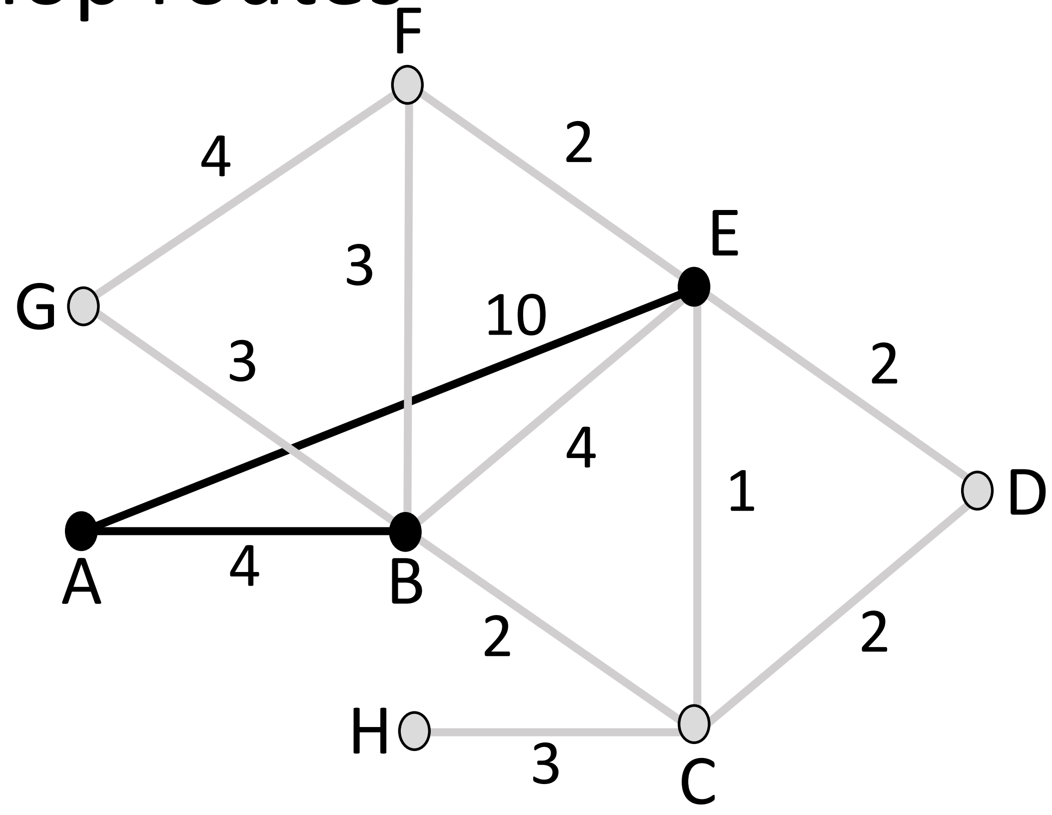
To	B says	E says
A	4	8
B	0	3
C	2	1
D	4	2
E	3	0
F	3	2
G	3	6
H	5	4

→

B +4	E +10
8	18
4	13
6	11
8	12
7	10
7	12
7	16
9	14

→

A's Cost	A's Next
0	--
4	B
6	B
8	B
7	B
7	B
7	B
9	B



Distance Vector (5)

- Subsequent exchanges; converged

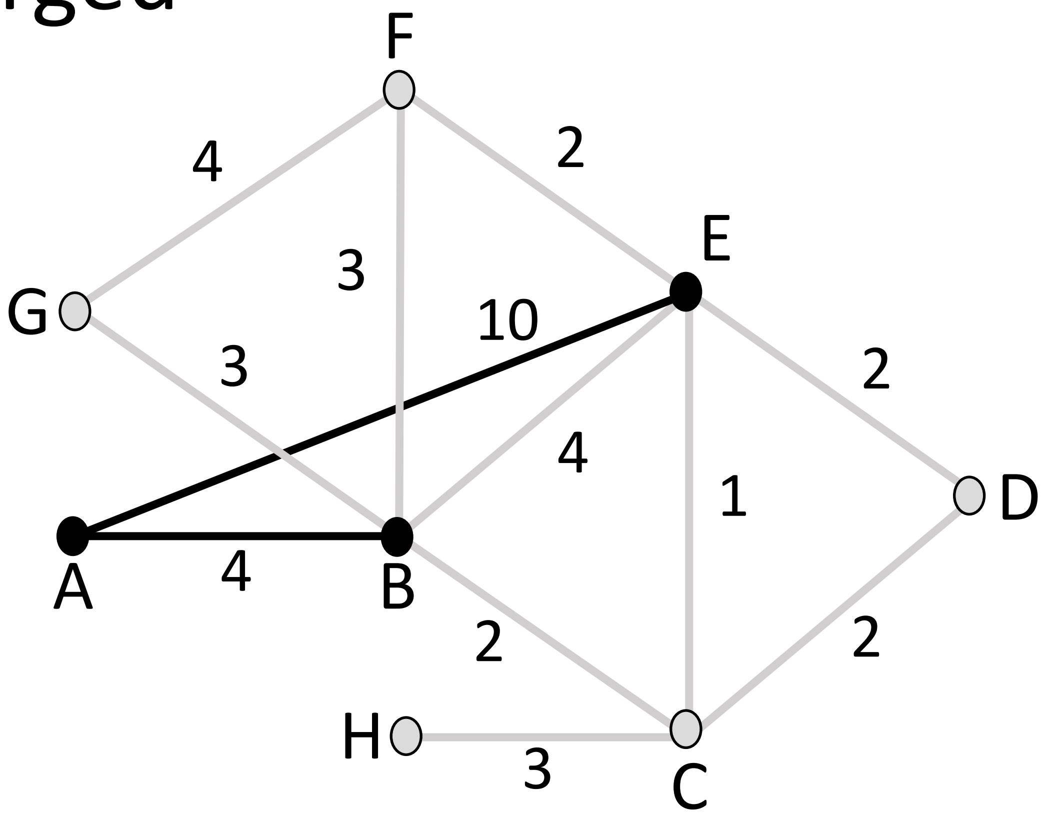
To	B says	E says
A	4	7
B	0	3
C	2	1
D	4	2
E	3	0
F	3	2
G	3	6
H	5	4

→

B +4	E +10
8	17
4	13
6	11
8	12
7	10
7	12
7	16
9	14

→

A's Cost	A's Next
0	--
4	B
6	B
8	B
8	B
7	B
7	B
9	B



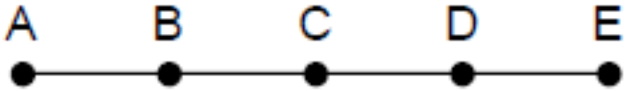
Distance Vector Dynamics

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

Problem?

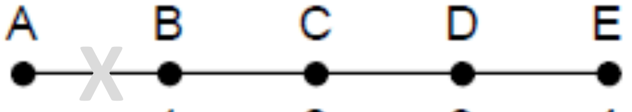
Count to Infinity: Problem

- Good news travels quickly, bad news slowly



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

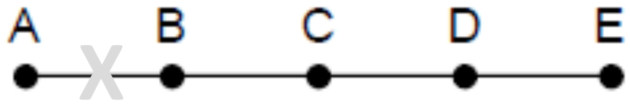


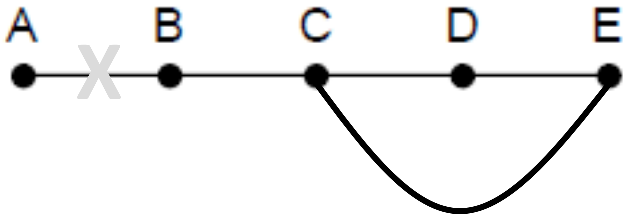
B	C	D	E	
1	2	3	4	Initially
3	2	3	4	After 1 exchange
3	4	3	4	After 2 exchanges
5	4	5	4	After 3 exchanges
5	6	5	6	After 4 exchanges
7	6	7	6	After 5 exchanges
7	8	7	8	After 6 exchanges
	⋮			

“Count to infinity” scenario

Count to Infinity: Heuristics

- "Split horizon"
  - Don't send route back to where you learned it from.
- Poison reverse
  - Send "infinity" when you notice a disconnect





Count to Infinity: Heuristics (2)

- Neither split horizon and poison reverse are very effective in practice
  - Link state is now favored except when 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
  - Time-out in 180 secs to detect failures
- RIPv1 specified in RFC1058 (1988)