CSE 461: Computer networks

Spring 2021

Ratul Mahajan

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

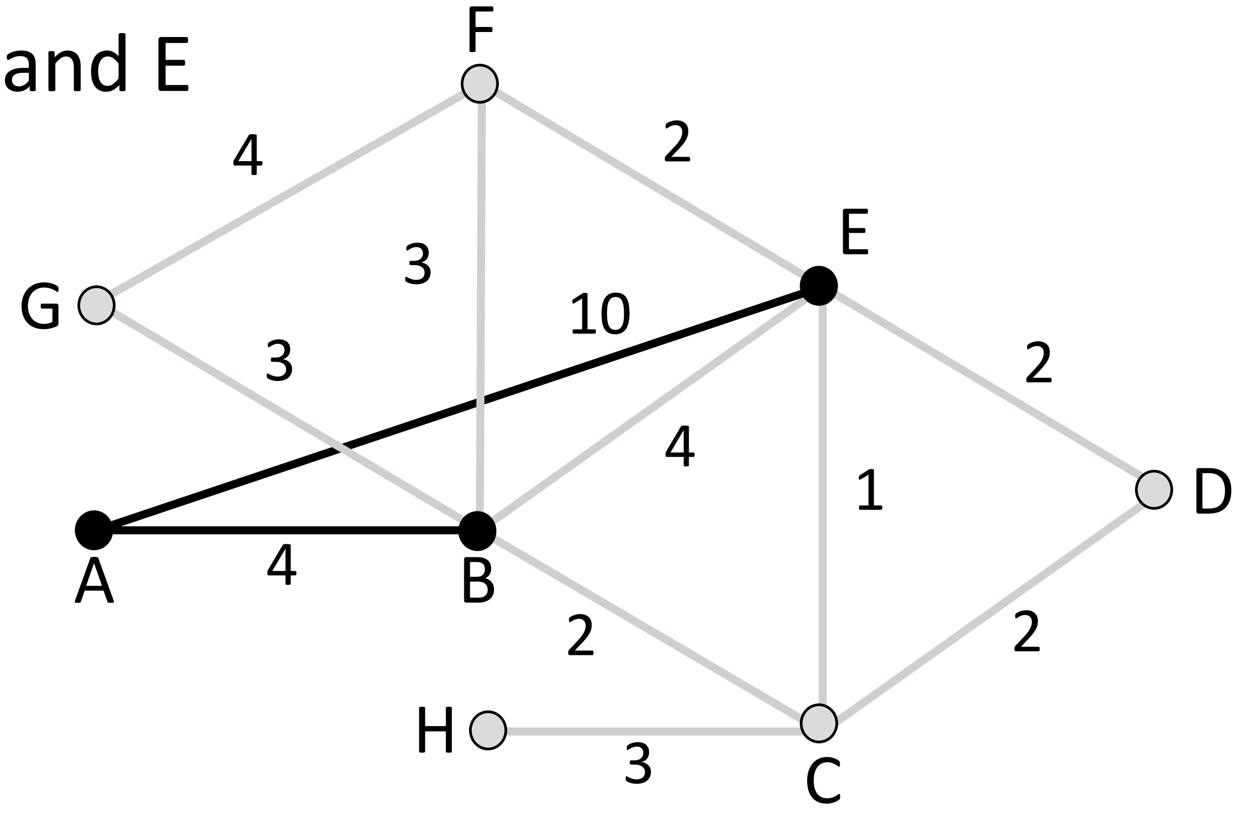
1. Initialize vector with 0 (zero) cost to self, ∞ (infinity) to other destinations
2. Periodically send vector to neighbors
3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
4. Use the best neighbor for forwarding

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

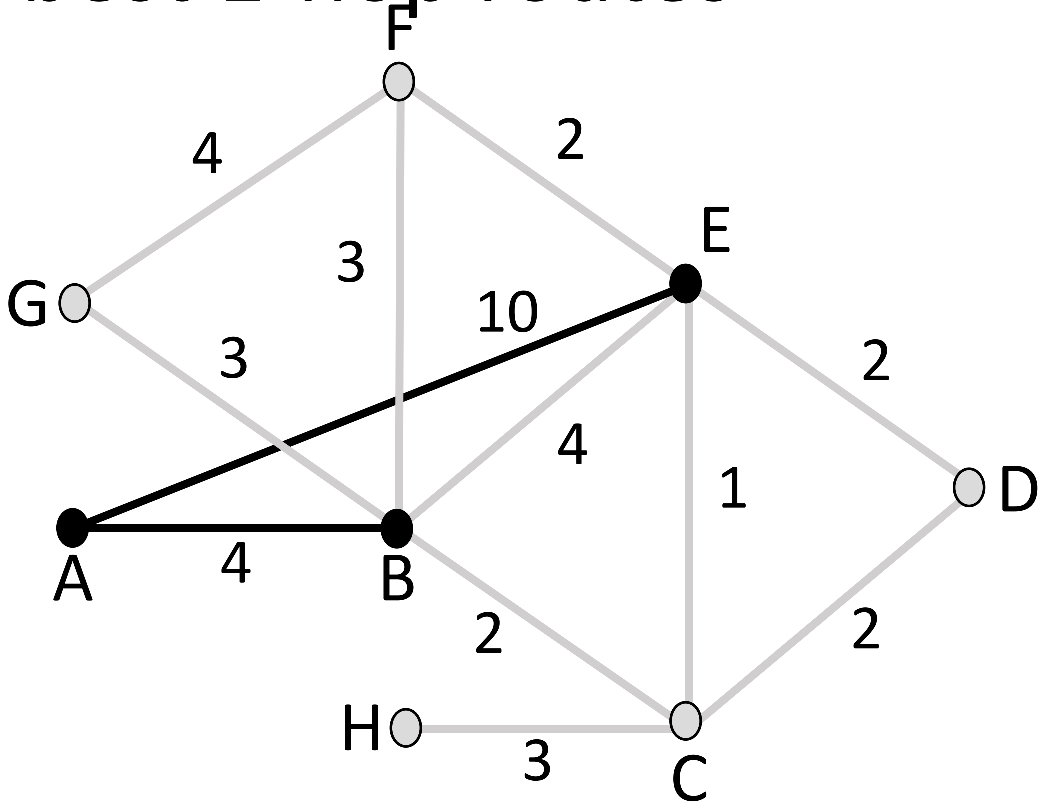
→

B +4	E +10
∞	∞
4	∞
∞	∞
∞	∞
∞	10
∞	∞
∞	∞
∞	∞

→

A's Cost	A's Next
0	--
4	B
∞	--
∞	--
10	E
∞	--
∞	--
∞	--

Learned better route



Distance Vector (4)

- Second exchange; learn best 2-hop routes

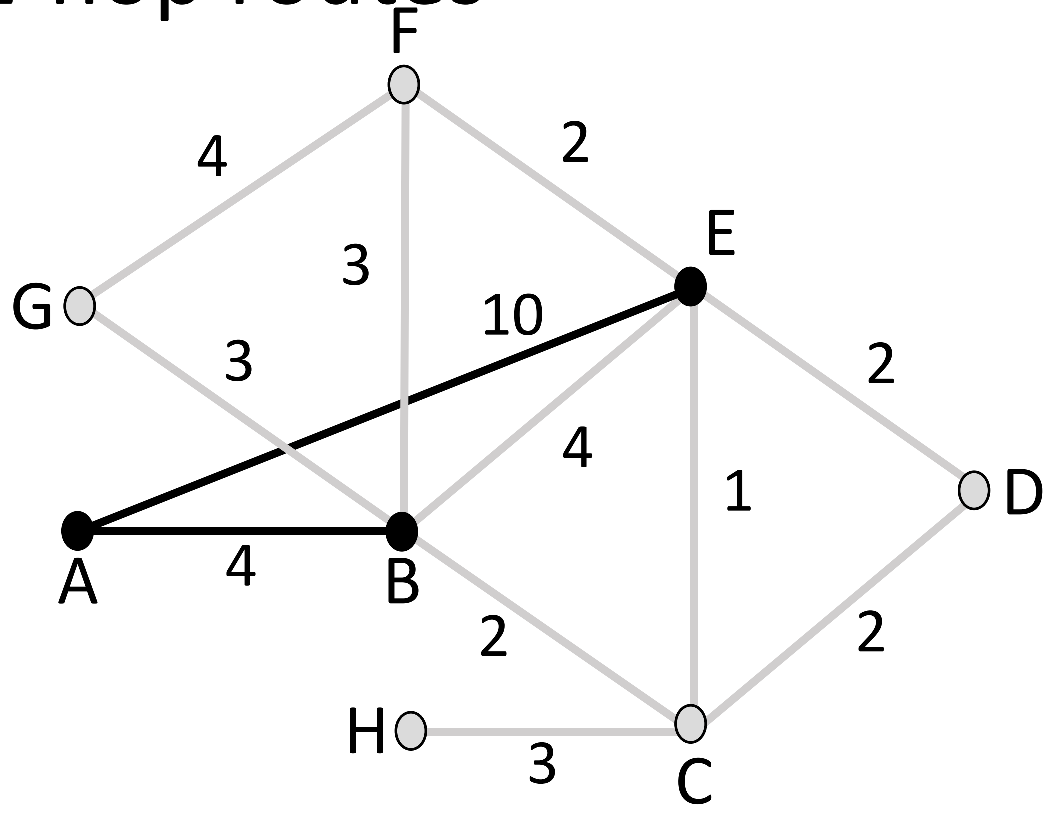
To	B says	E says
A	4	10
B	0	4
C	2	1
D	∞	2
E	4	0
F	3	2
G	3	∞
H	∞	∞

→

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

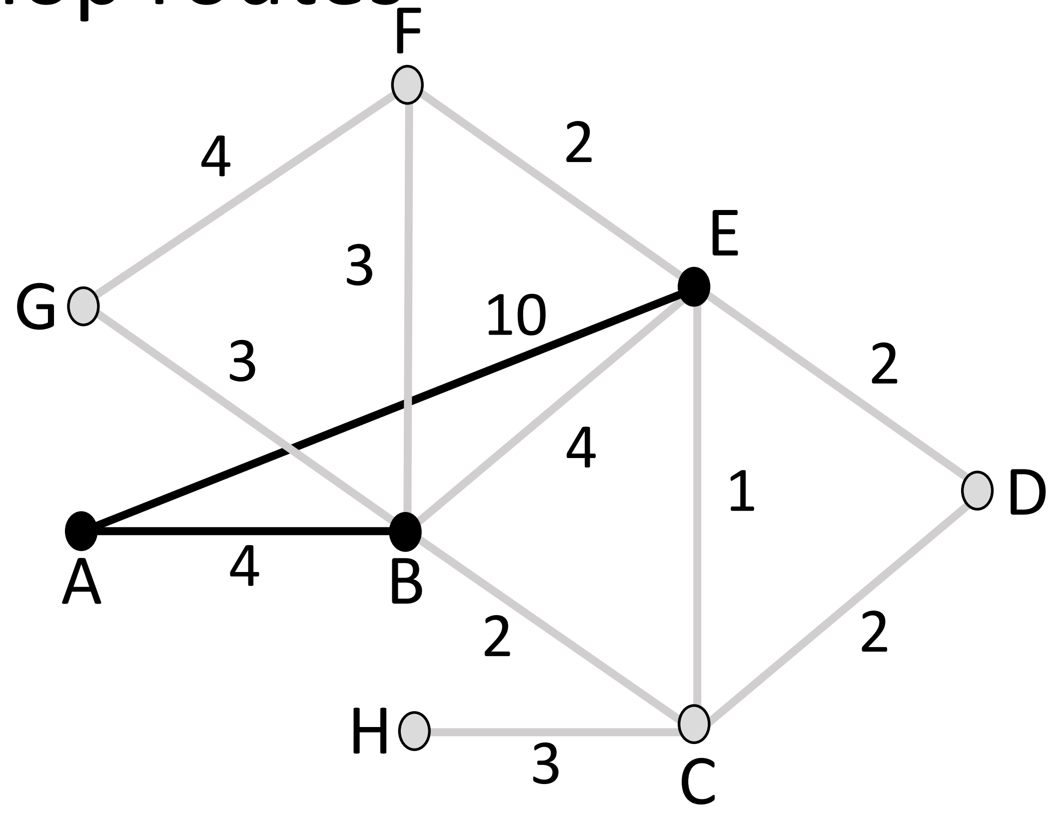
To	B says	E says
A	4	8
B	0	3
C	2	1
D	4	2
E	3	0
F	3	2
G	3	6
H	5	4

→

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

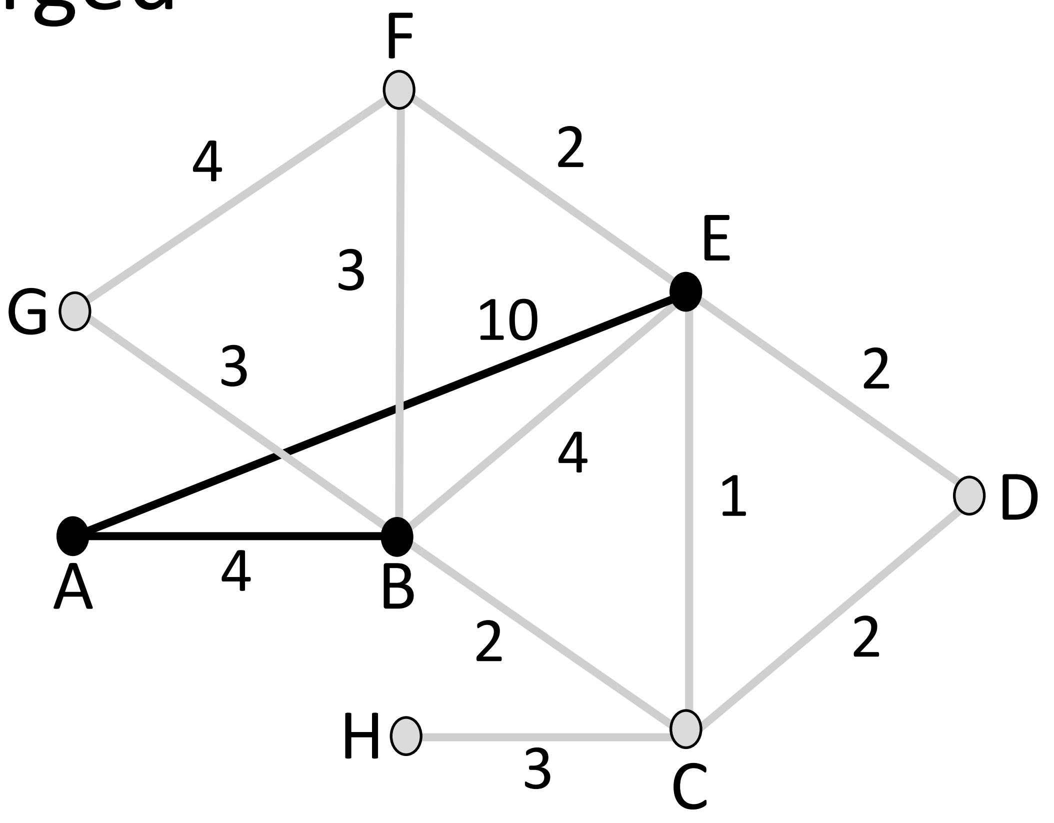
To	B says	E says
A	4	7
B	0	3
C	2	1
D	4	2
E	3	0
F	3	2
G	3	6
H	5	4

→

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

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

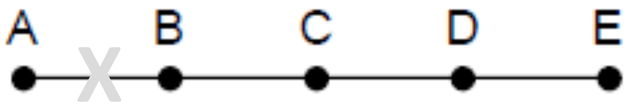
Desired convergence

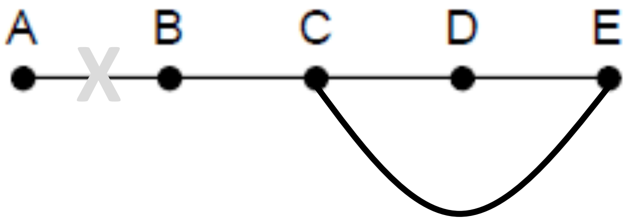
A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			

"Count to infinity" scenario

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)

Link-State Routing

Link-State Routing

- Second broad class of routing algorithms
  - More computation than DV but better dynamics
- Widely used in practice
  - Used in Internet/ARPANET from 1979
  - Modern networks use OSPF (L3) and IS-IS (L2)

Link-State Setting

Same distributed setting as for distance vector:

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/links may fail, messages may be lost

Link-State Algorithm

Proceeds in two phases:

1. Nodes <u>flood</u> topology with link state packets
   - Each node learns full topology
2. Each node computes its own forwarding table
   - By running Dijkstra (or equivalent)

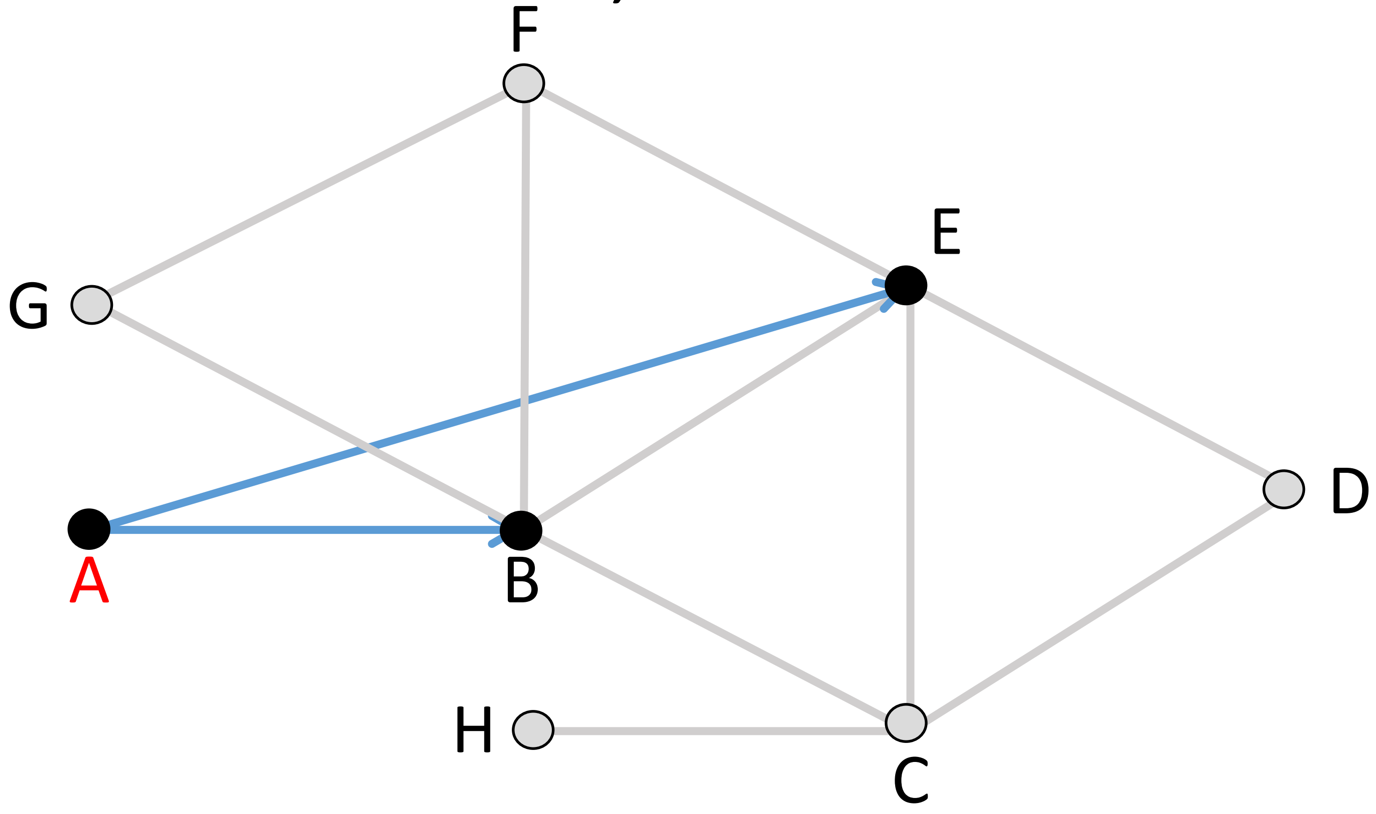
Part 1: Flooding

Flooding

- Rule used at each node:
  - Sends an incoming message on to all other neighbors
  - Remember the message so that it is only flood once

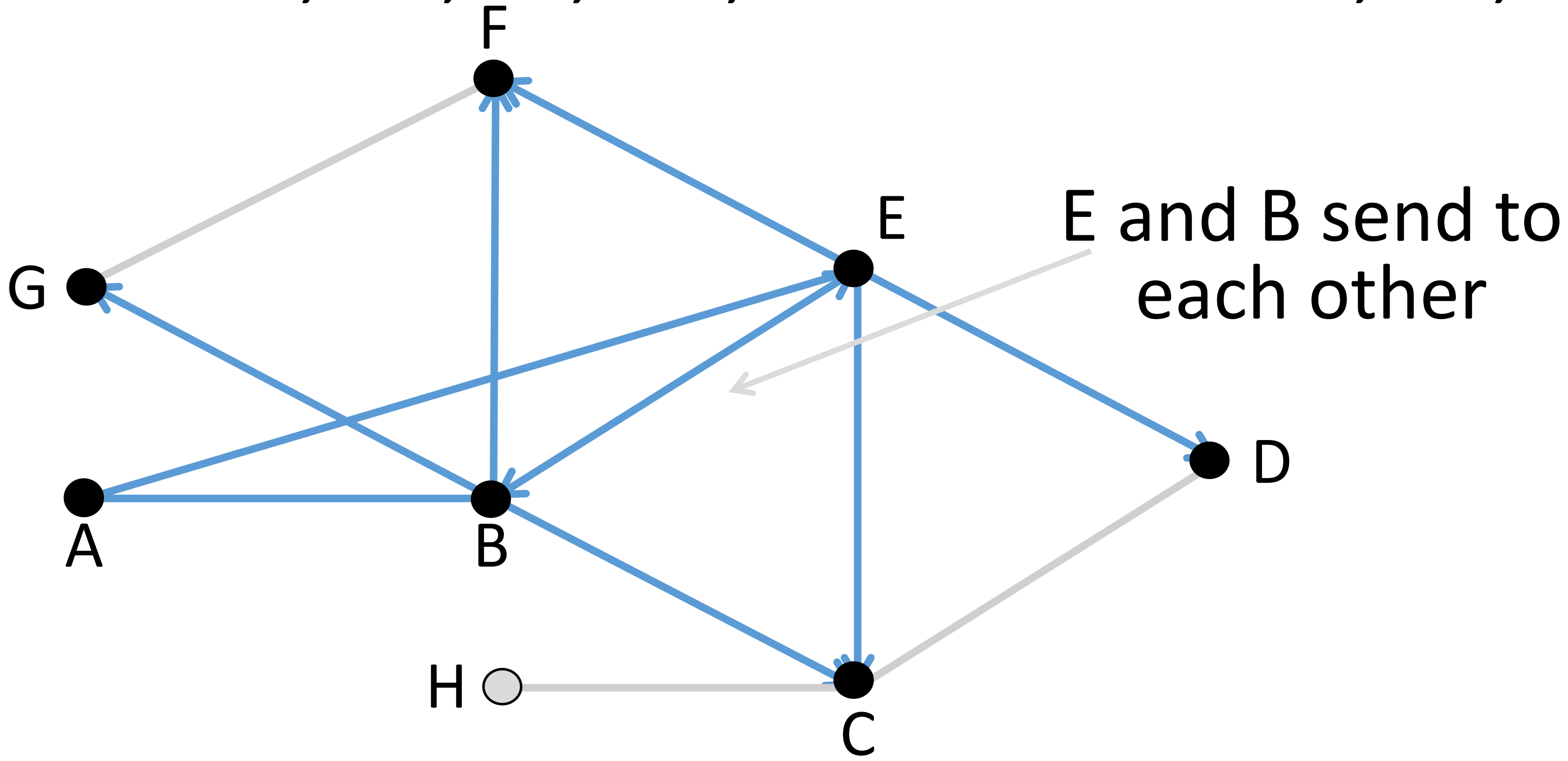
Flooding (2)

- Consider a flood from A; first reaches B via AB, E via AE



Flooding (3)

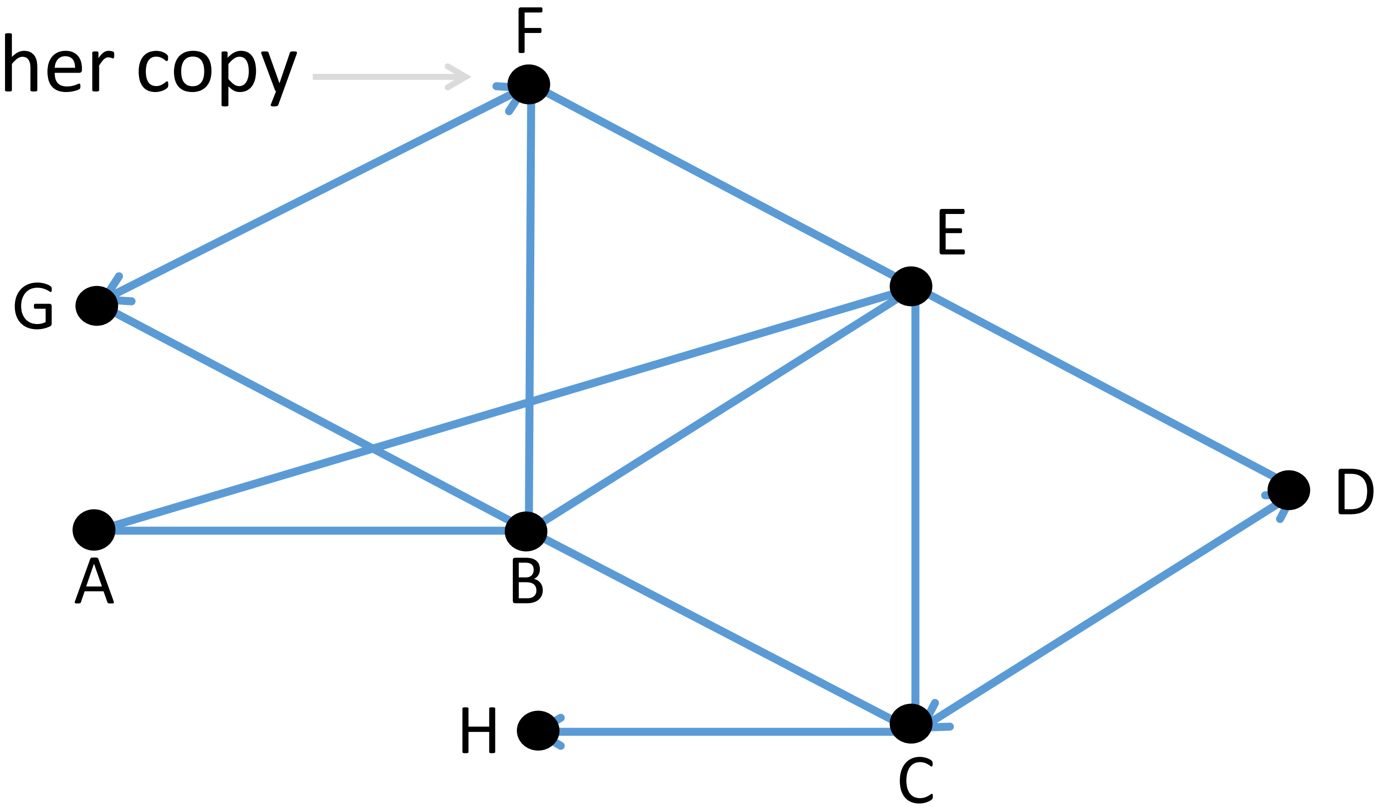
- Next B floods BC, BE, BF, BG, and E floods EB, EC, ED, EF



Flooding (4)

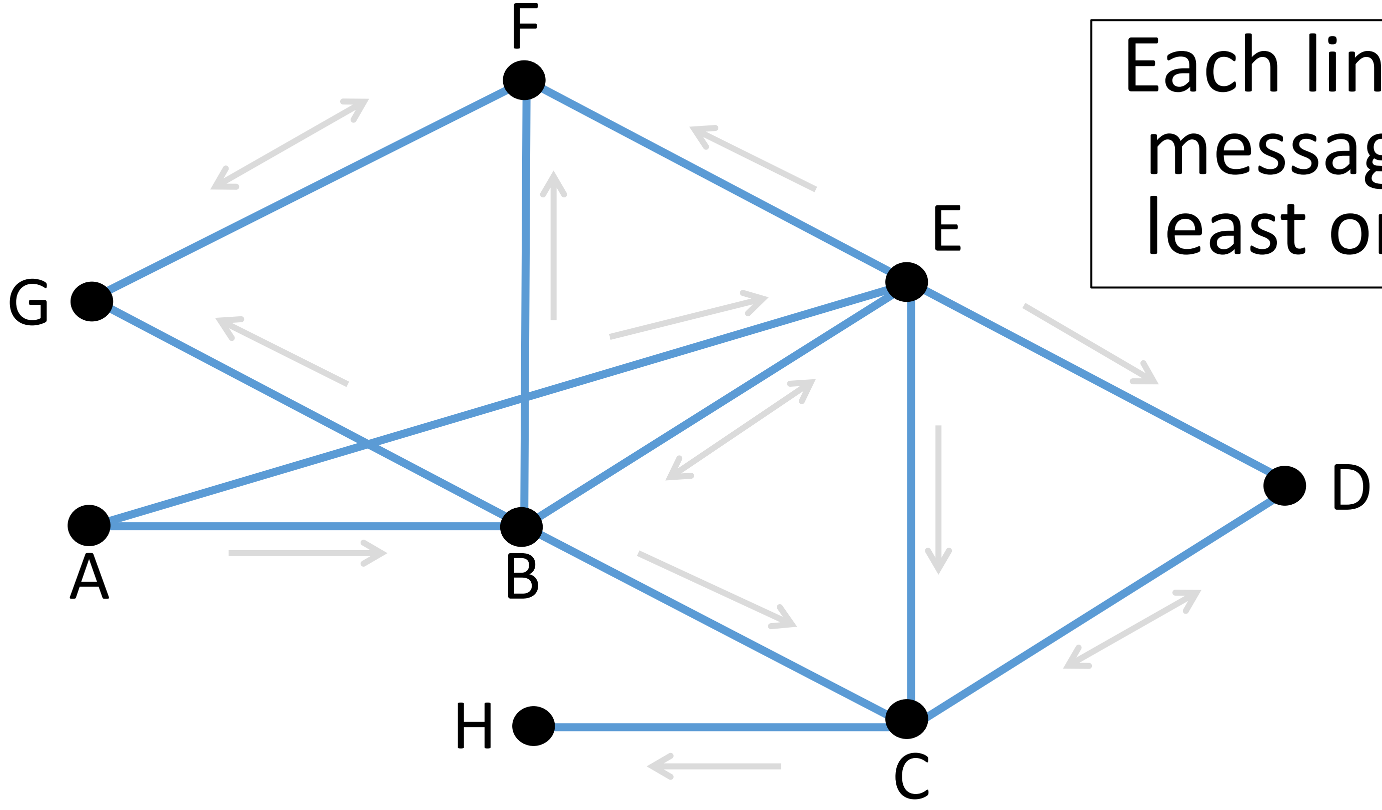
- C floods CD, CH; D floods DC; F floods FG; G floods GF

F gets another copy



Flooding (5)

- H has no-one to flood … and we're done



Each link carries the message, and in at least one direction

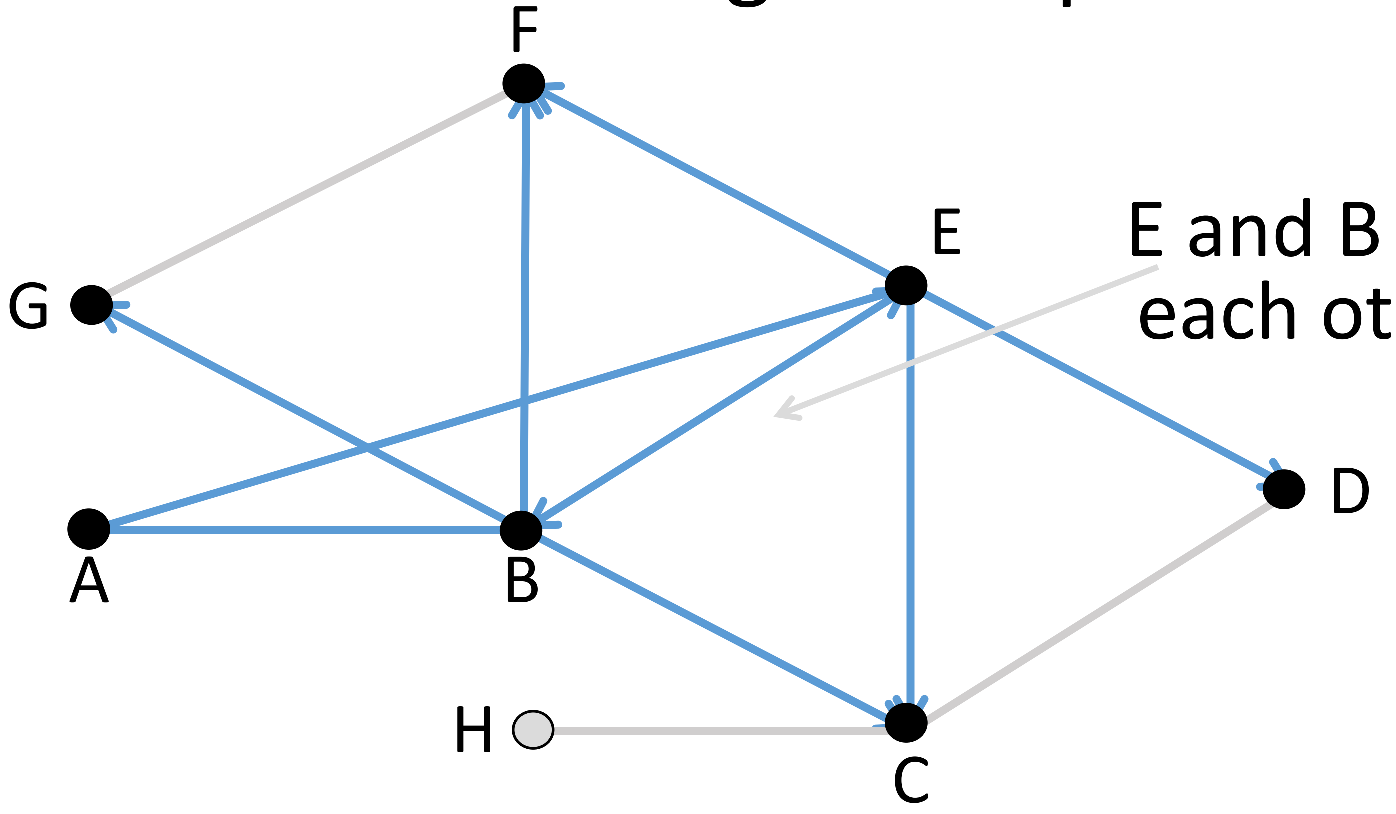
Flooding Details

- Remember message (to stop flood) using source and sequence number
  - So next message (with higher sequence) will go through
- To make flooding reliable, use ARQ
  - So receiver acknowledges, and sender resends if needed

Problem?

Flooding Problem

- F receives the same message multiple times



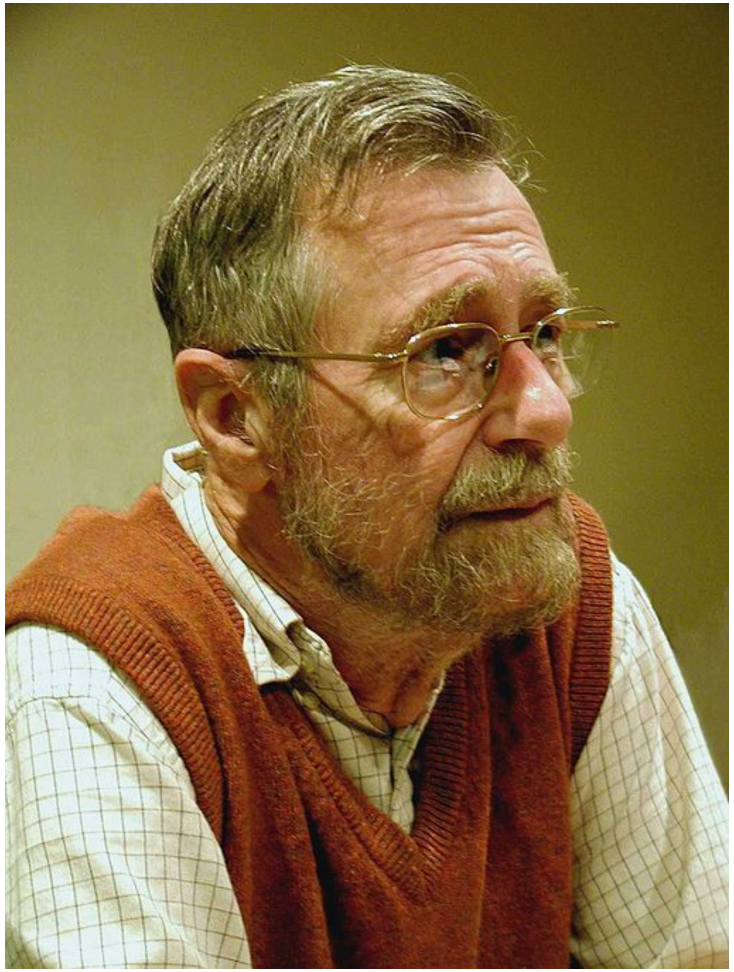
E and B send to each other too

Part 2: Dijkstra’s Algorithm

Edsger W. Dijkstra (1930-2002)

- Famous computer scientist
  - Programming languages
  - Distributed algorithms
  - Program verification

- Dijkstra’s algorithm, 1969
  - Single-source shortest paths, given network with non-negative link costs



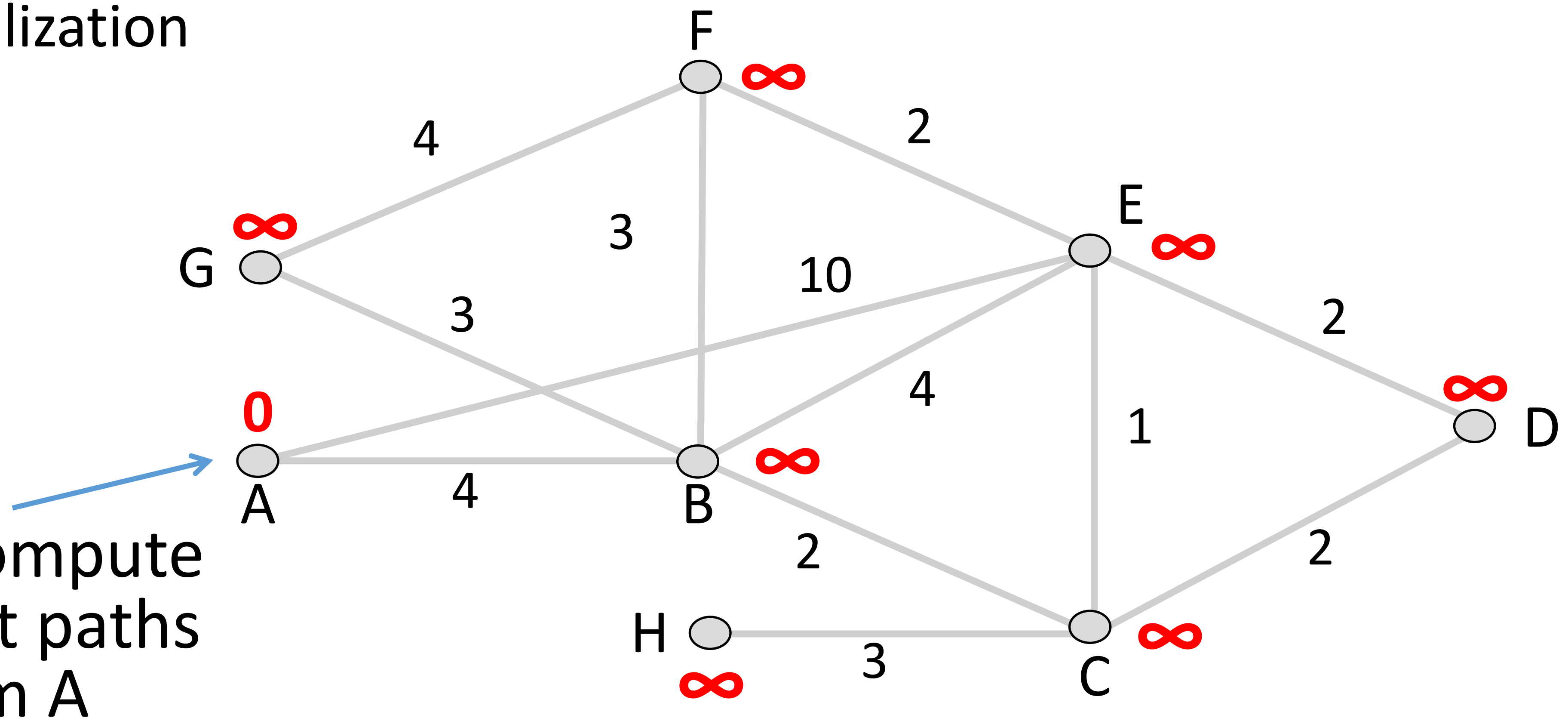
Dijkstra's Algorithm

Algorithm:

- Mark all nodes tentative, set distances from source to 0 (zero) for source, and ∞ (infinity) 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)

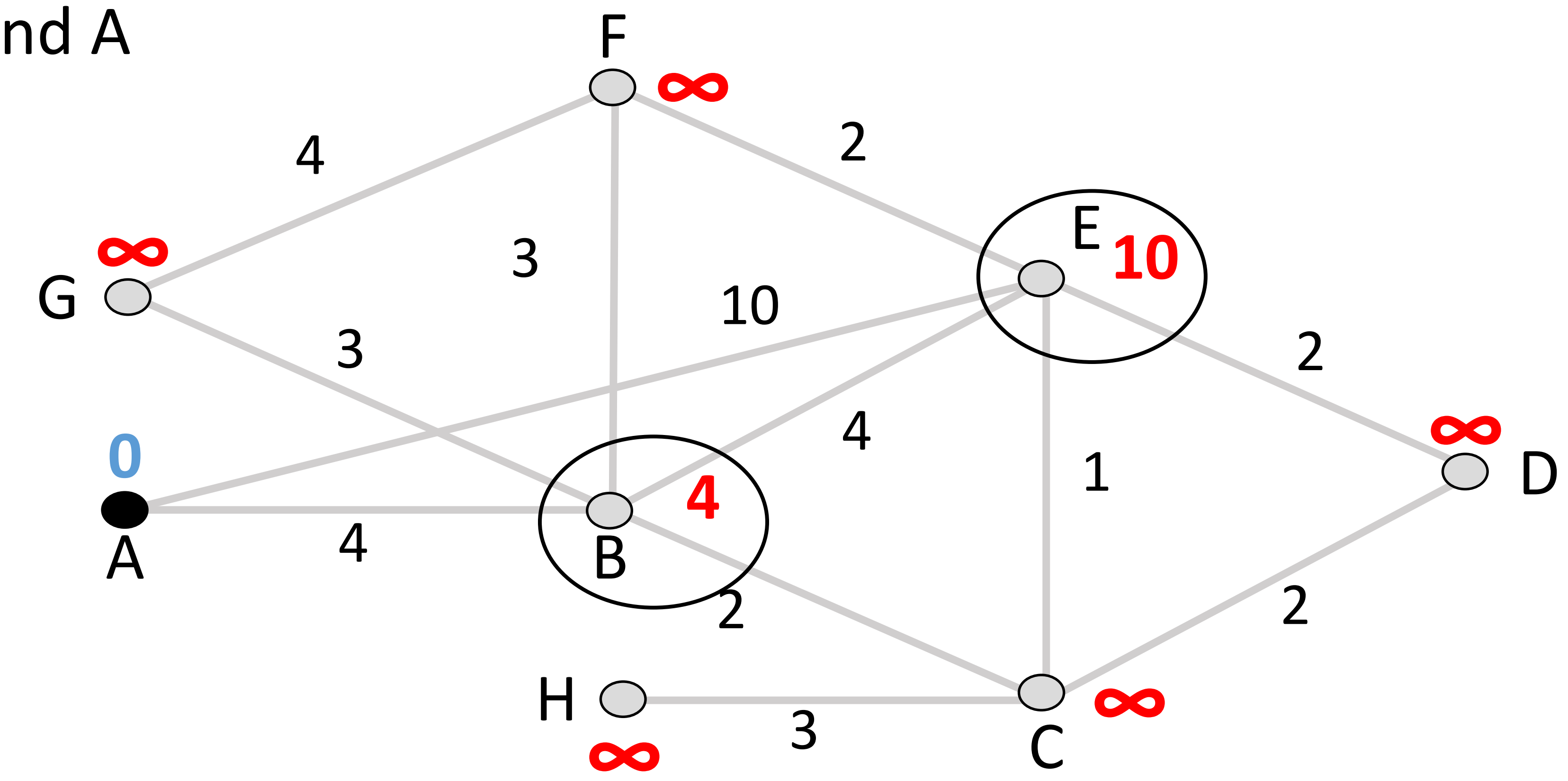
- Initialization



We'll compute shortest paths from A

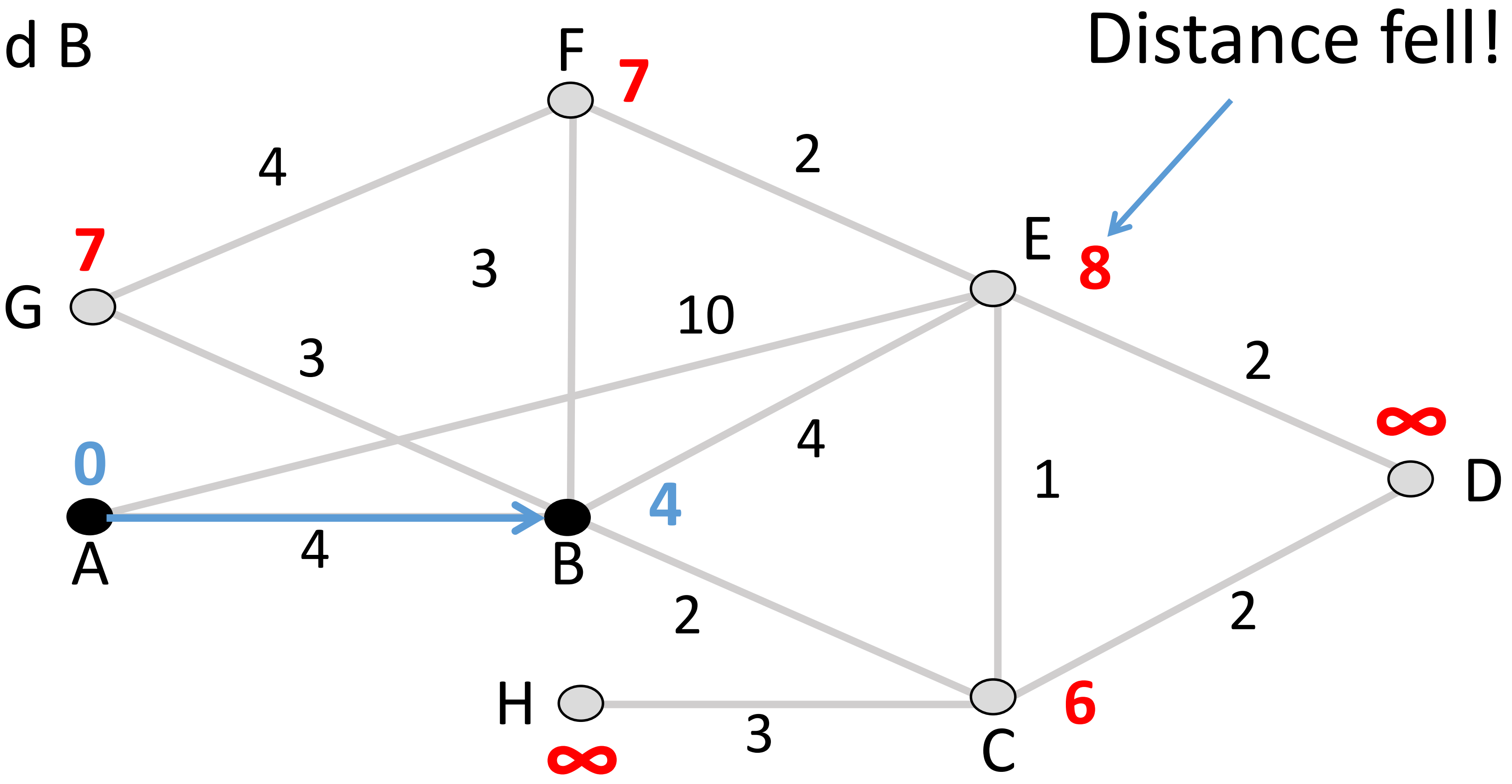
Dijkstra’s Algorithm (3)

- Relax around A



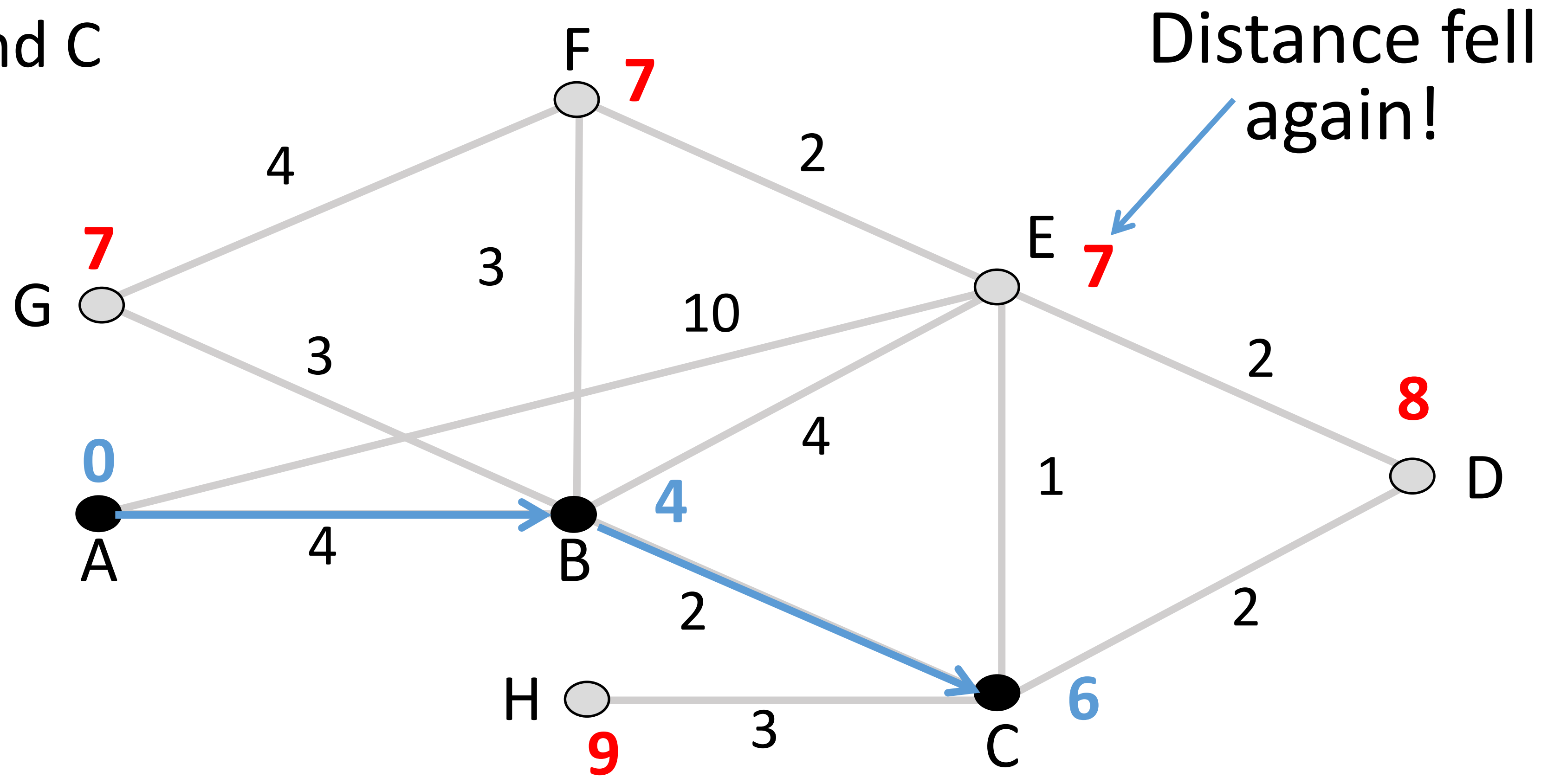
Dijkstra's Algorithm (4)

- Relax around B



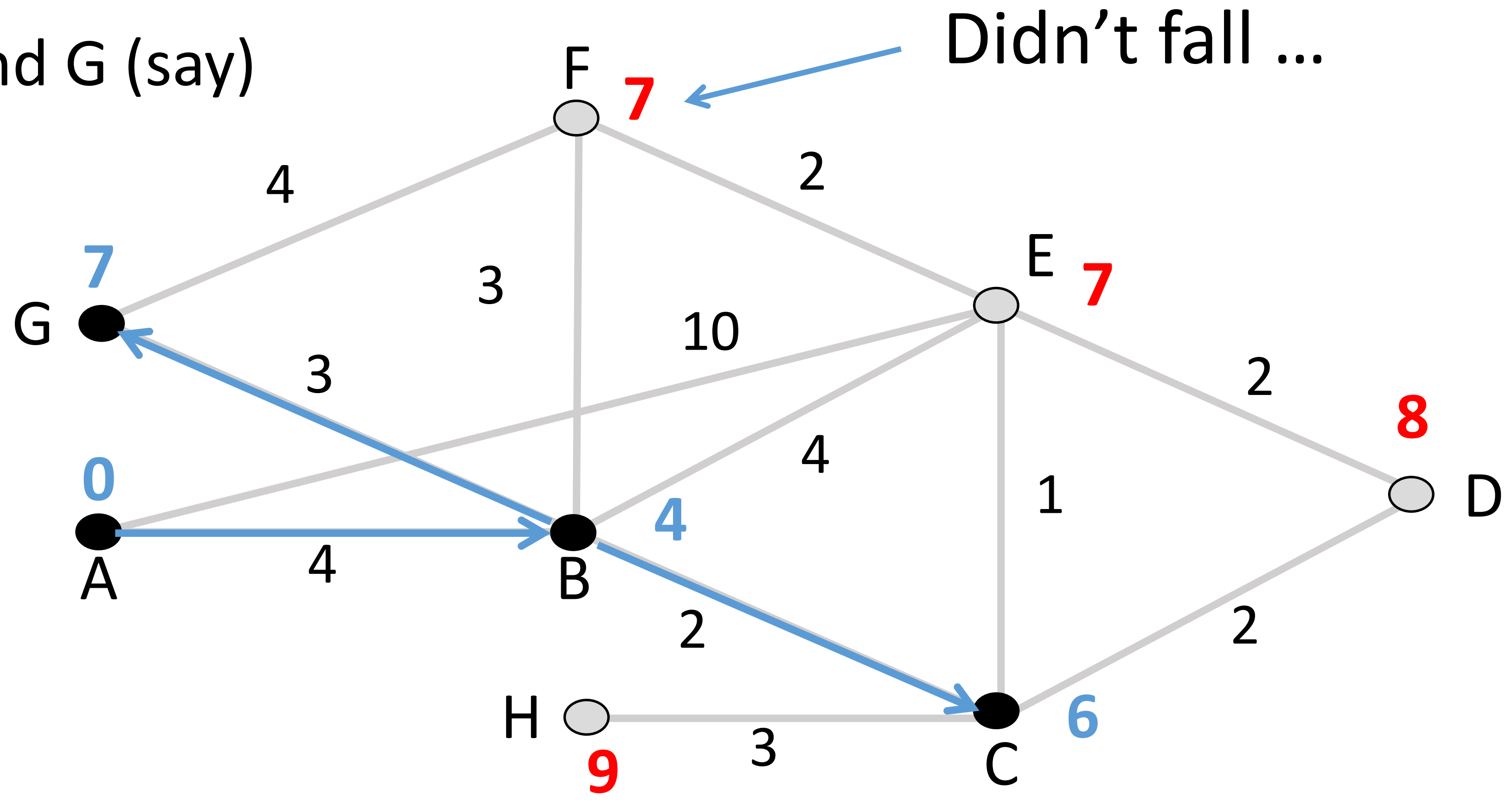
Dijkstra’s Algorithm (5)

- Relax around C



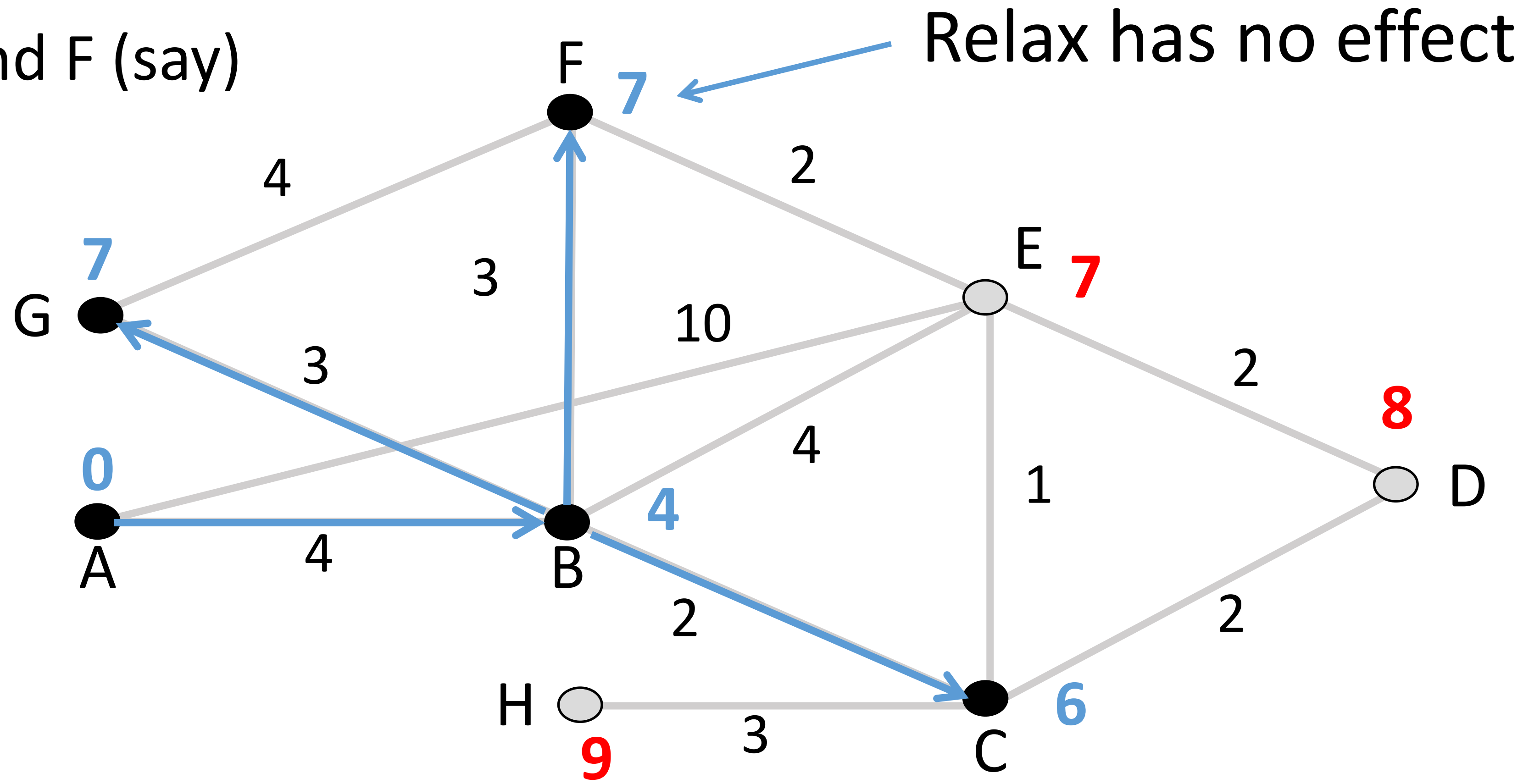
Dijkstra's Algorithm (6)

- Relax around G (say)



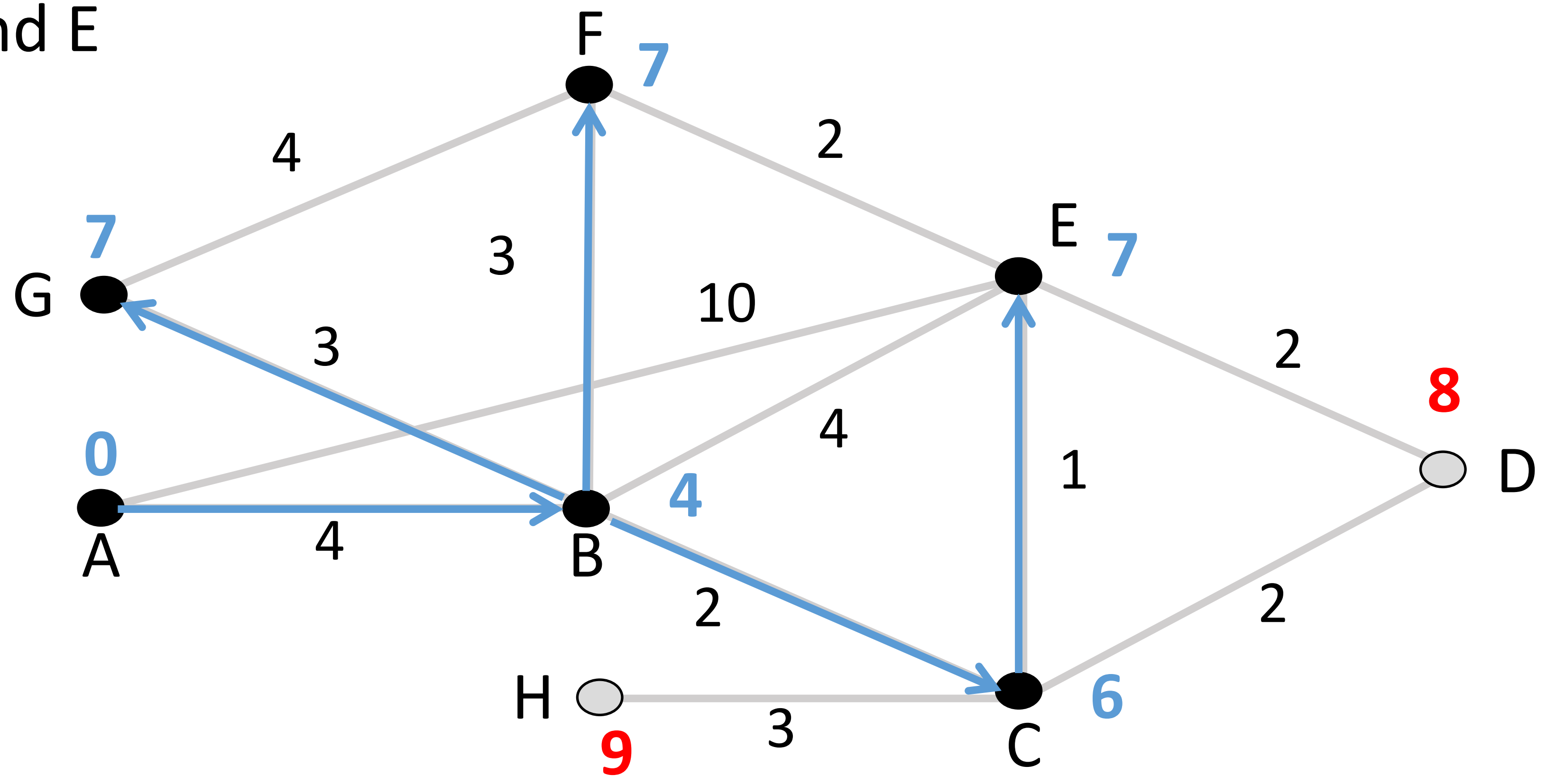
Dijkstra's Algorithm (7)

- Relax around F (say)



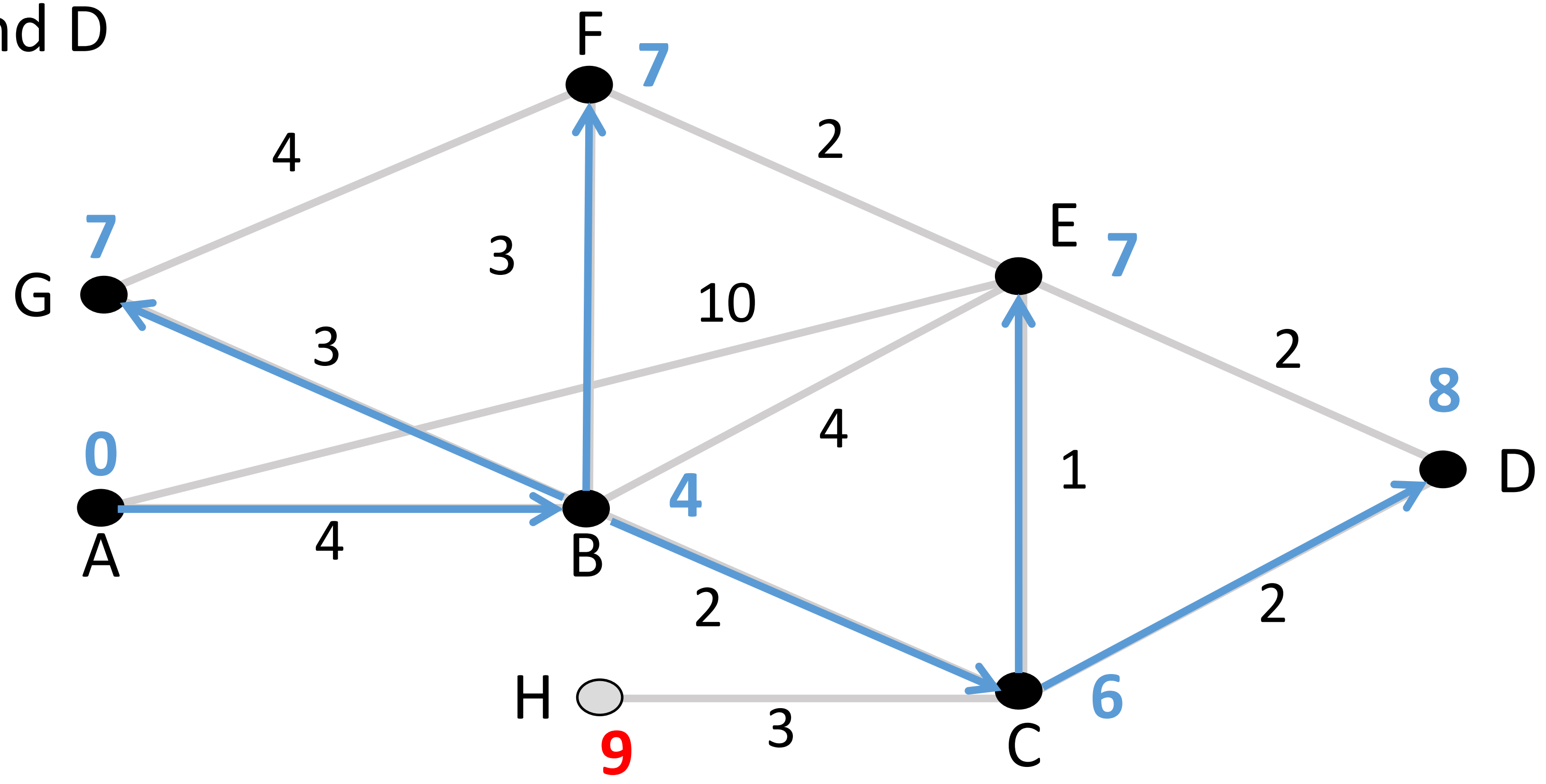
Dijkstra's Algorithm (8)

- Relax around E



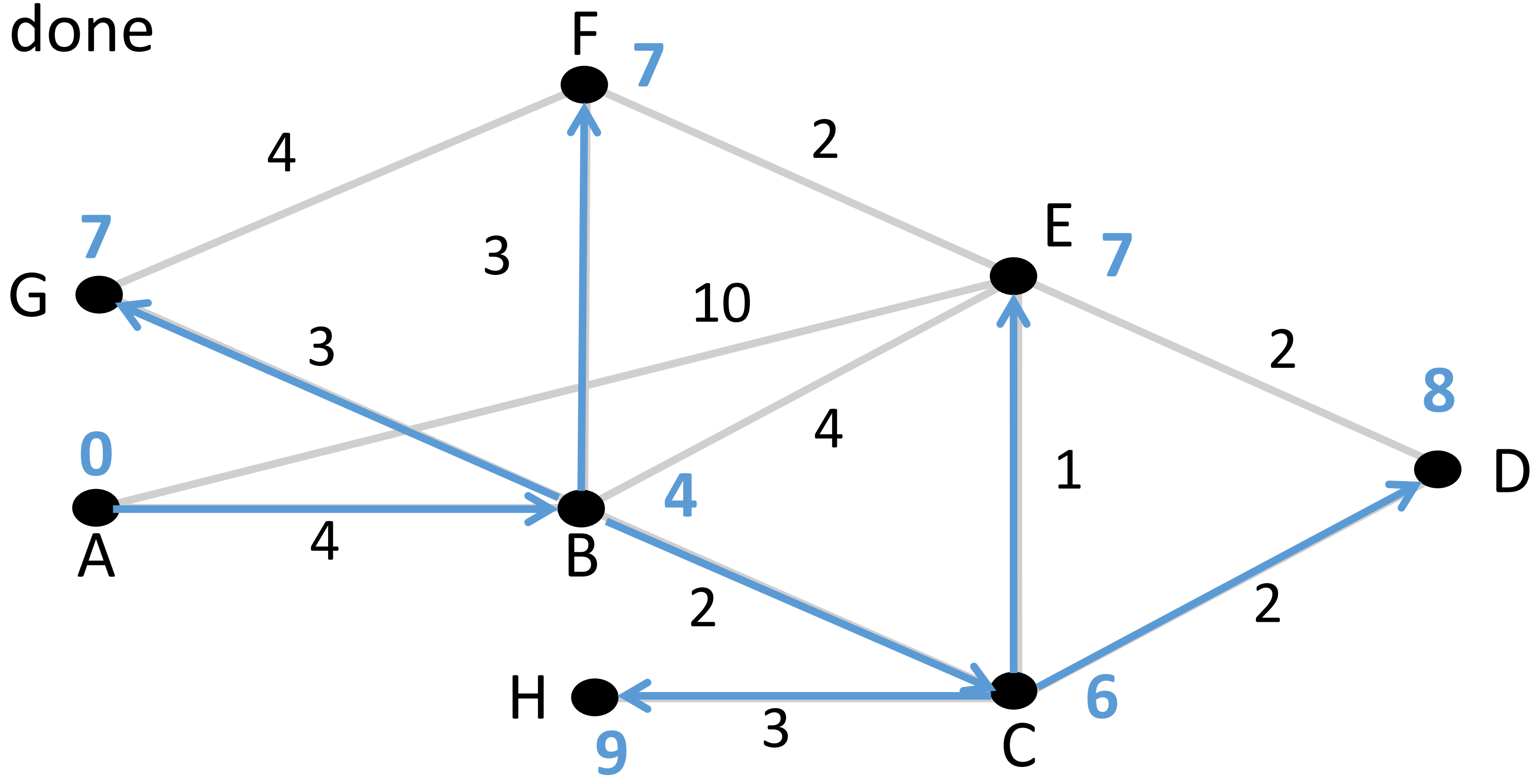
Dijkstra's Algorithm (9)

- Relax around D



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 cost of extracting min-cost node
  - Superlinear in network size (grows fast)
  - Using Fibonacci Heaps the complexity is $O(|E|+|V|\log|V|)$
- Gives complete source/sink tree
  - More than needed for forwarding!
  - But requires complete topology

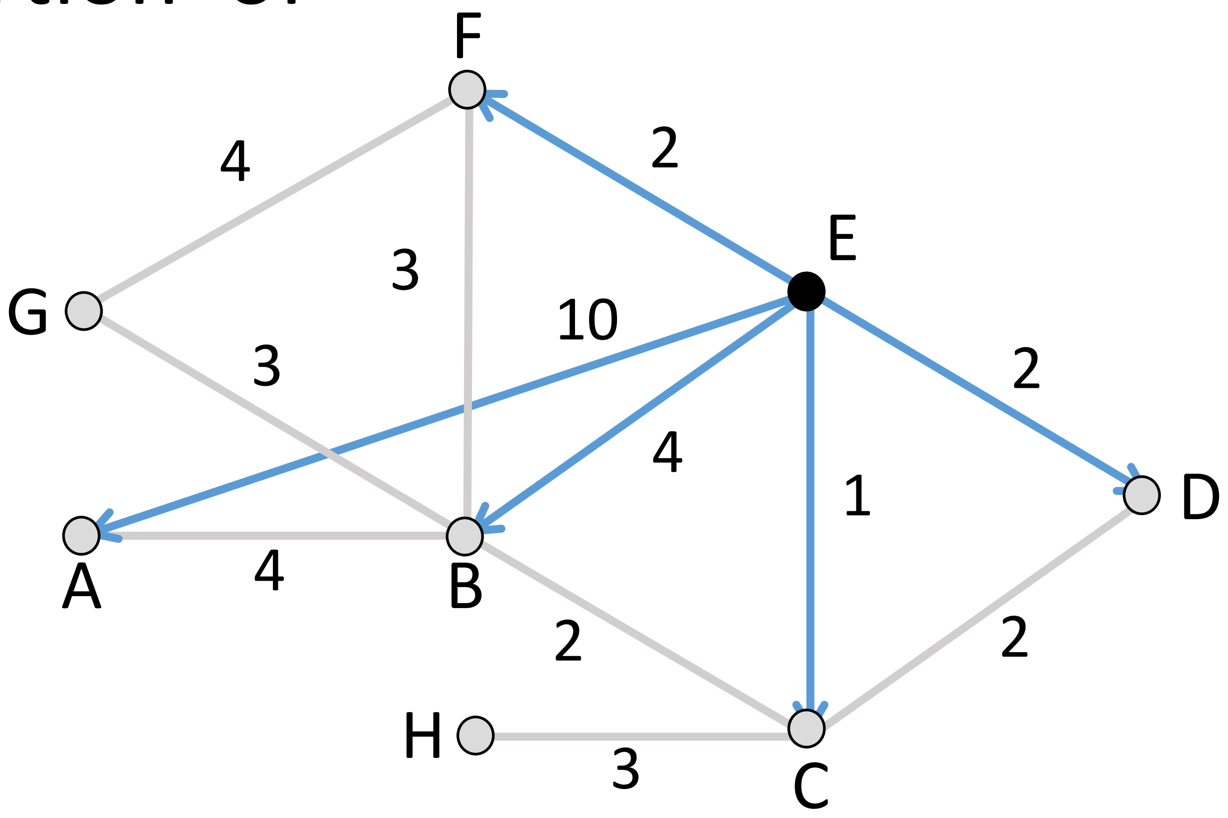
Bringing it all together…

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

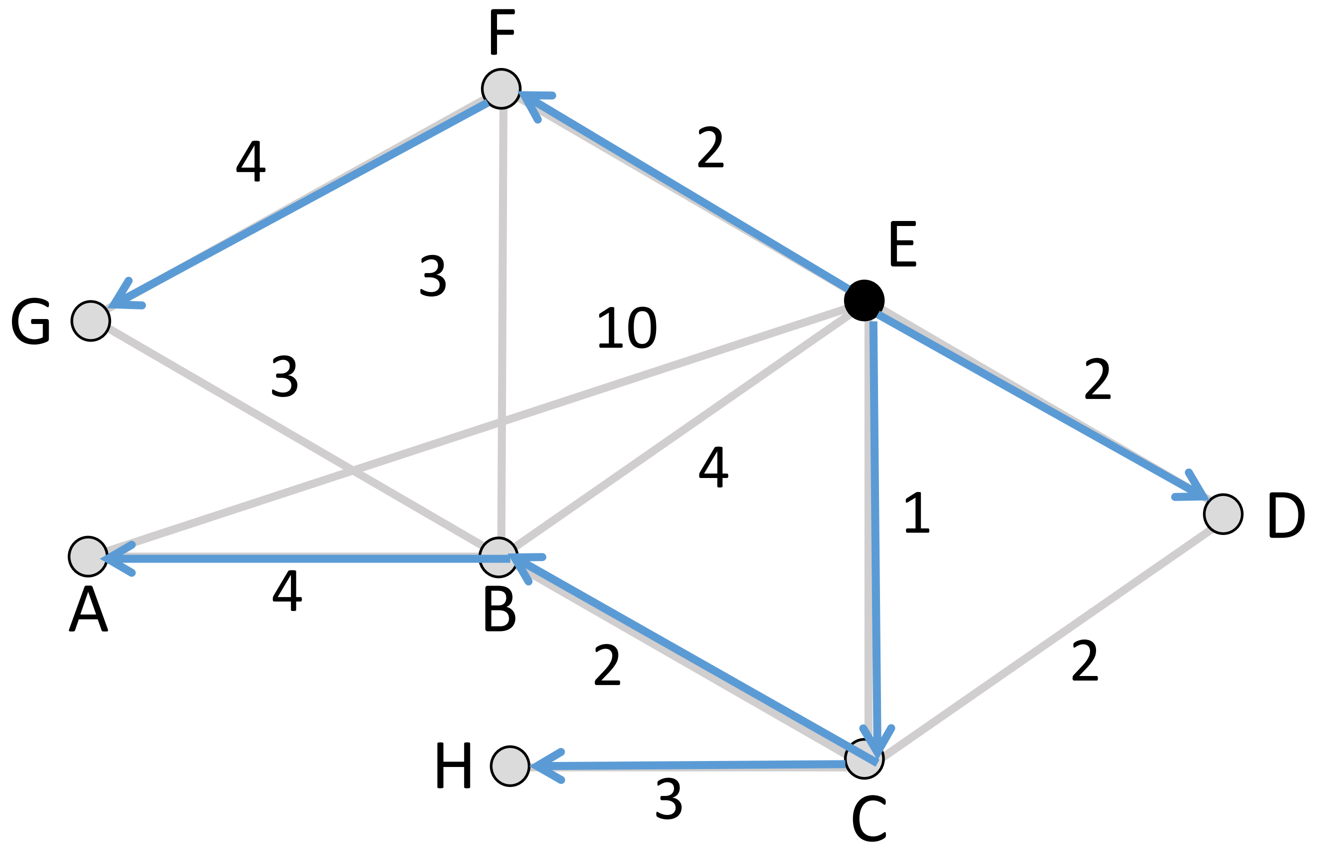


Phase 2: Route Computation

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

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

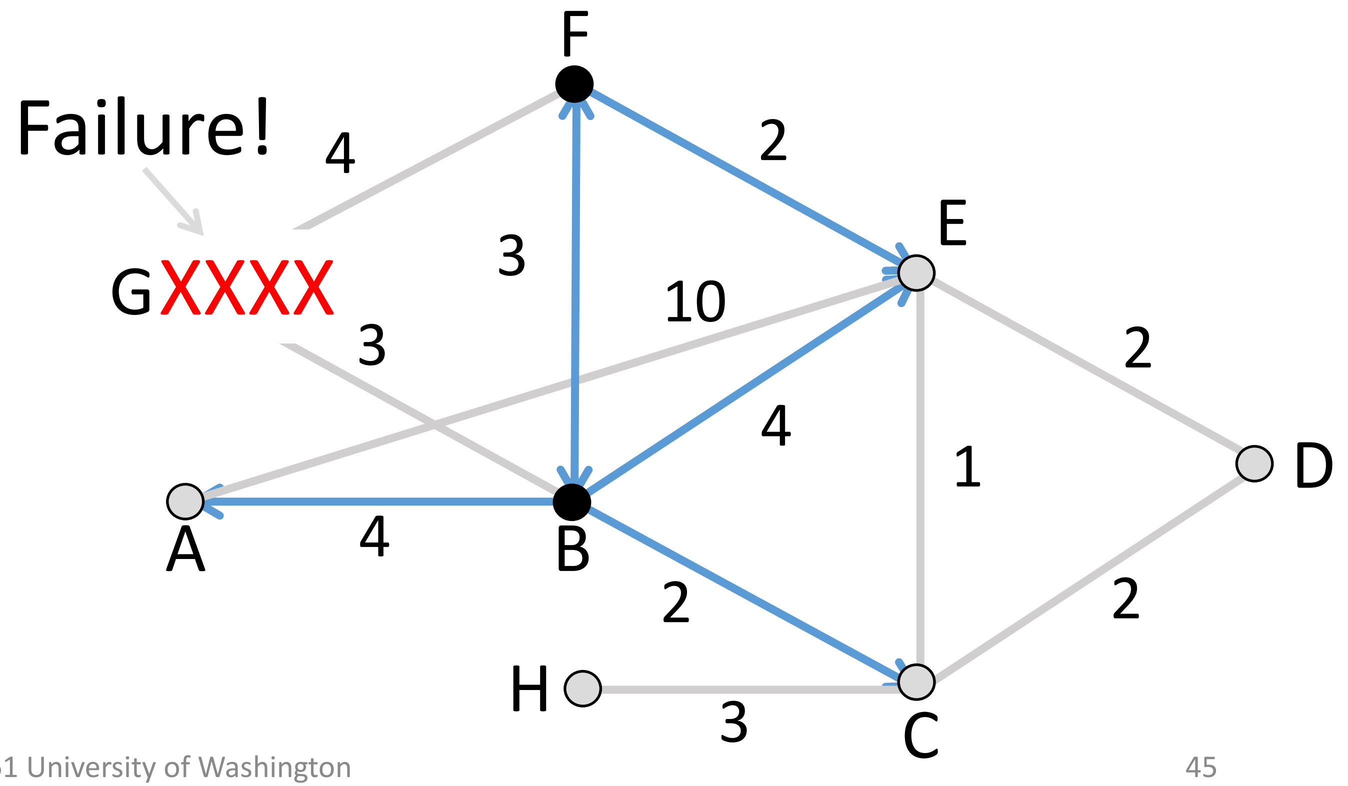
- On change, flood updated LSPs, re-compute routes
  - E.g., nodes adjacent to failed link or node initiate

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 can't 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
- Much of the complexity is due to handling corner cases

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

IS-IS and OSPF Protocols

- Widely used in large enterprise and ISP networks
  - IS-IS = Intermediate System to Intermediate System
  - OSPF = Open Shortest Path First
- Link-state protocol with many added features
  - E.g., “Areas” for scalability