

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

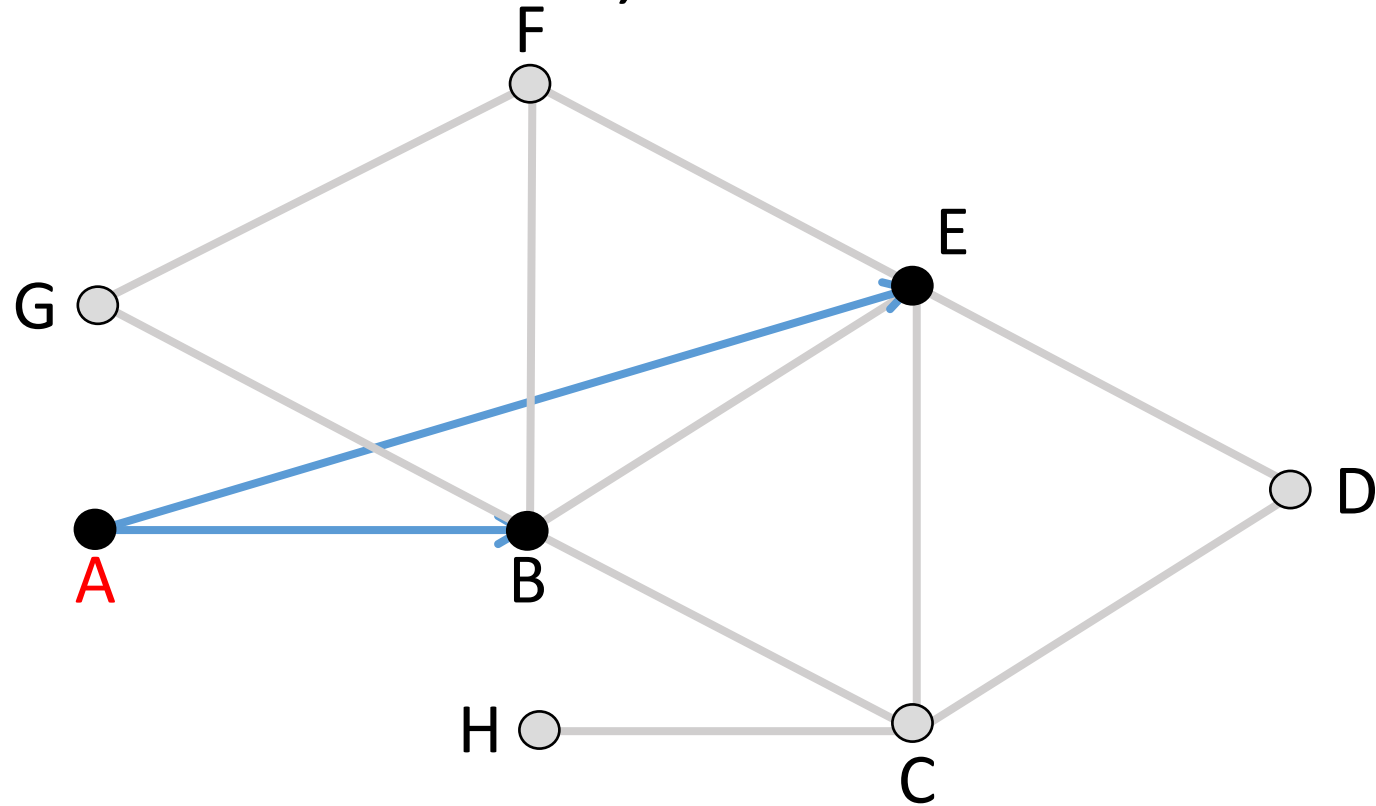
Part 1: Flooding

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

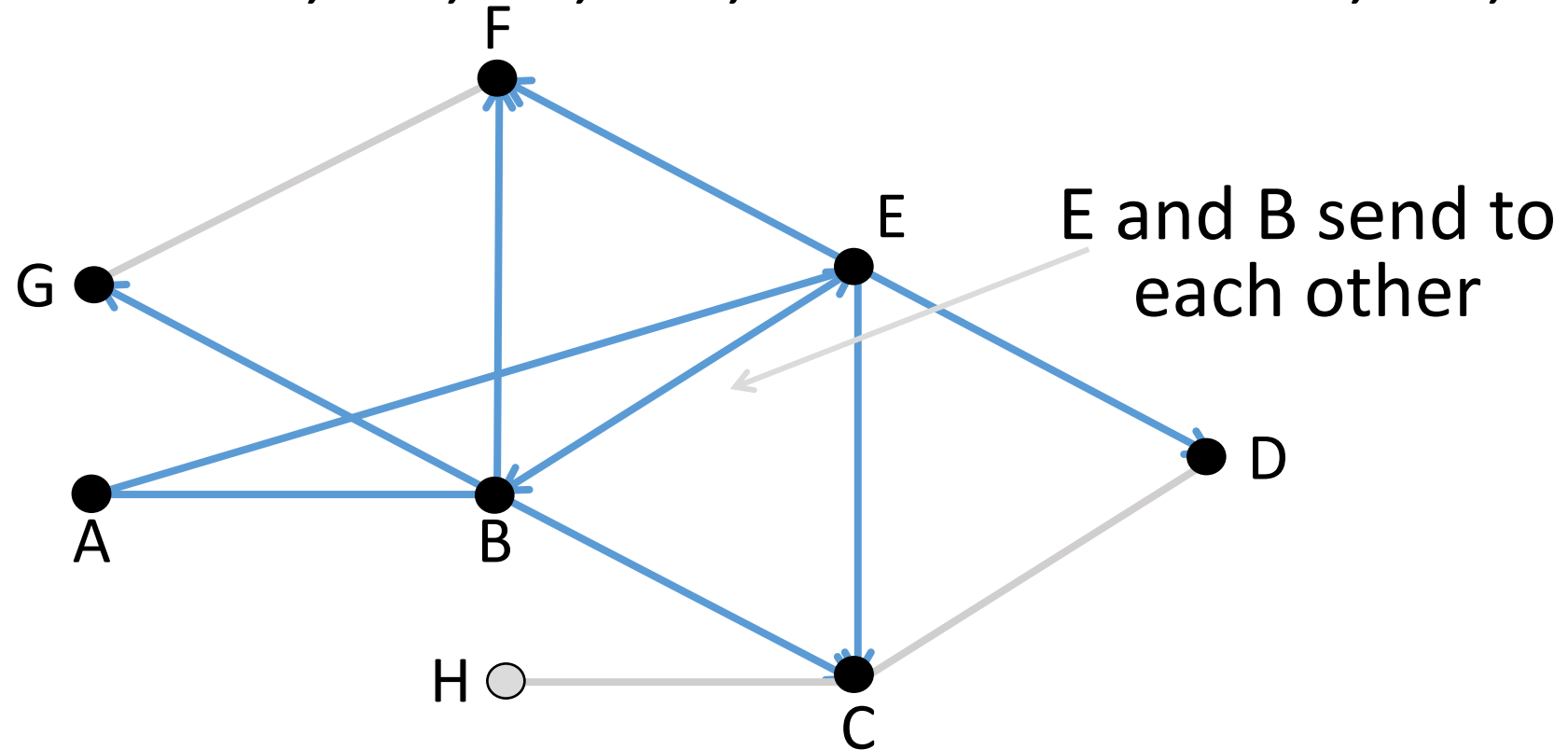
Flooding (2)

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



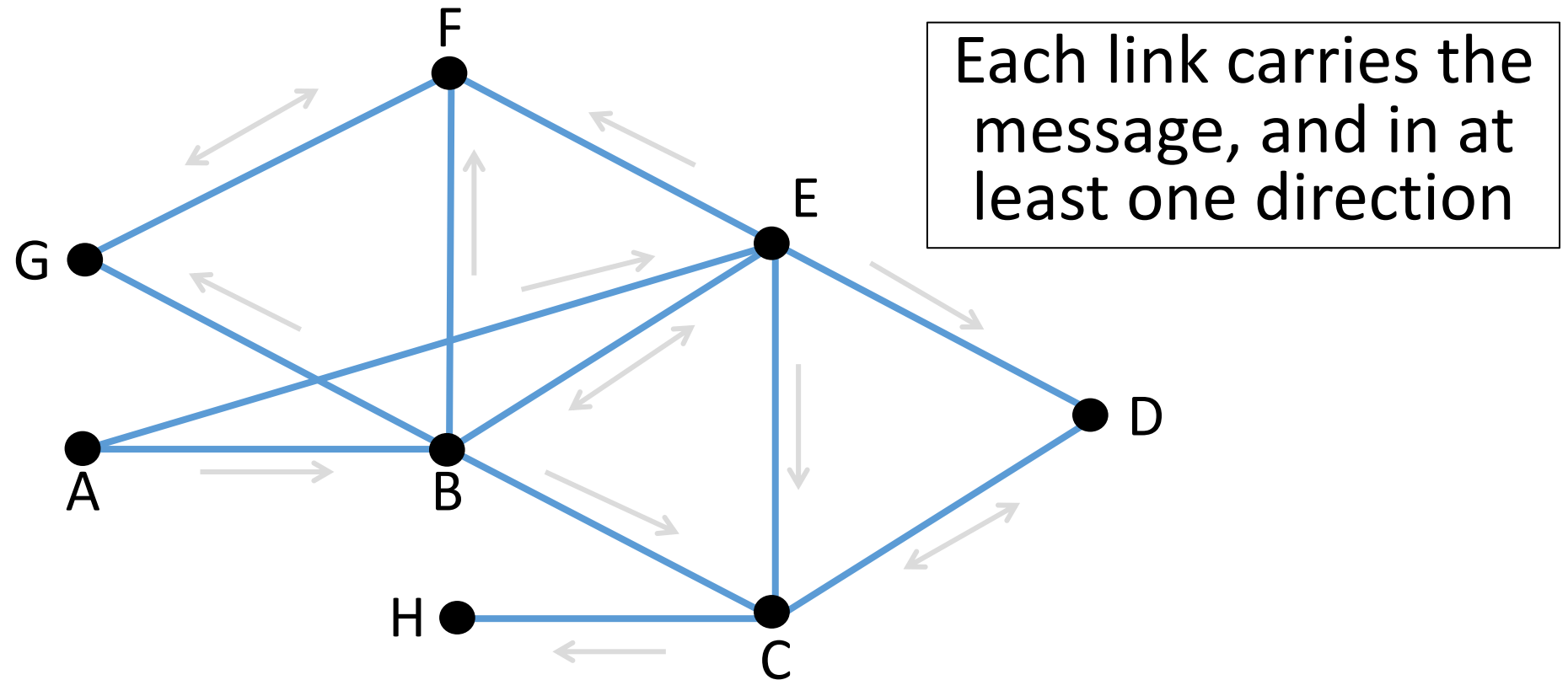
Flooding (3)

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



Flooding (5)

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



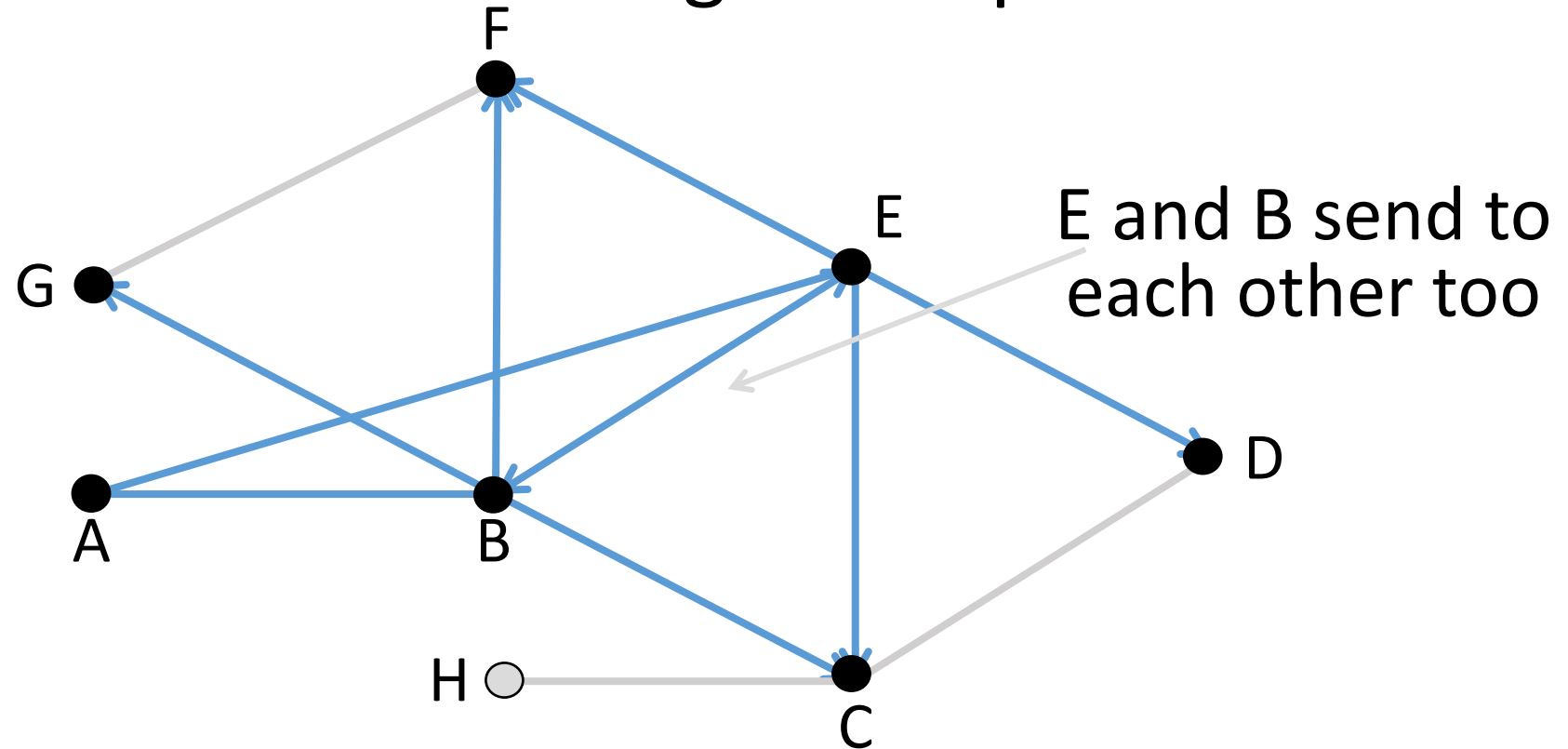
Flooding Details

- 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

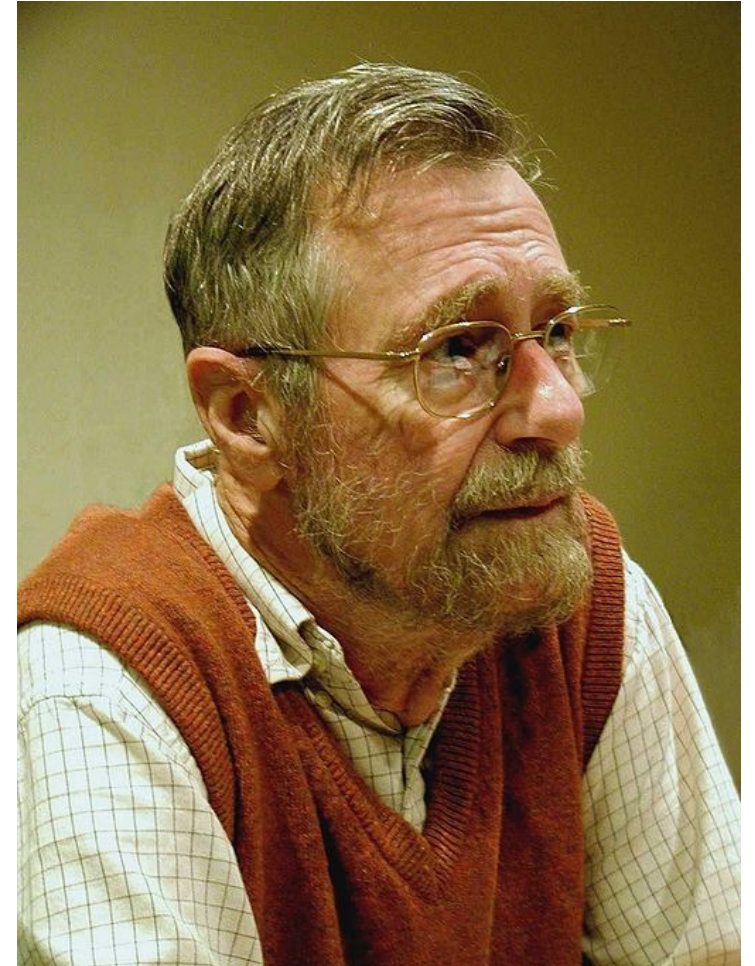
- F receives the same message multiple times



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



By Hamilton Richards, CC-BY-SA-3.0, via Wikimedia Commons

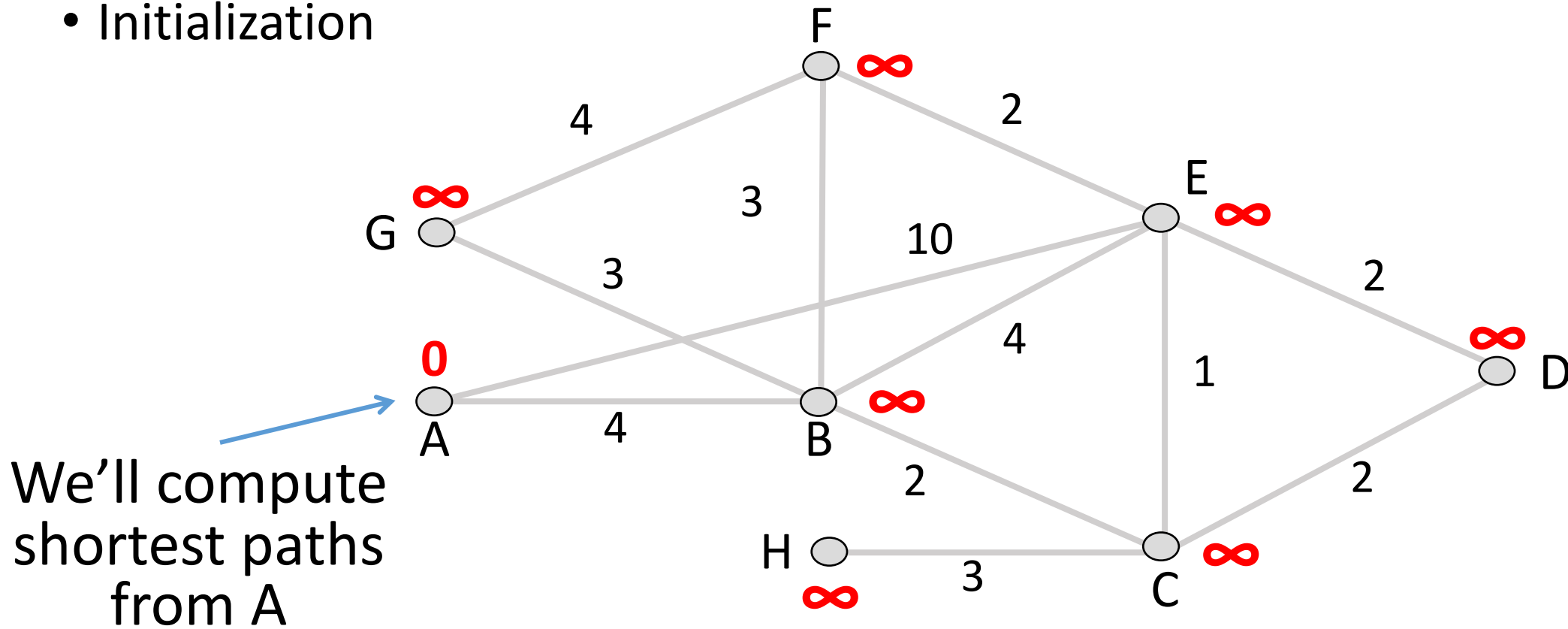
Dijkstra's Algorithm

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

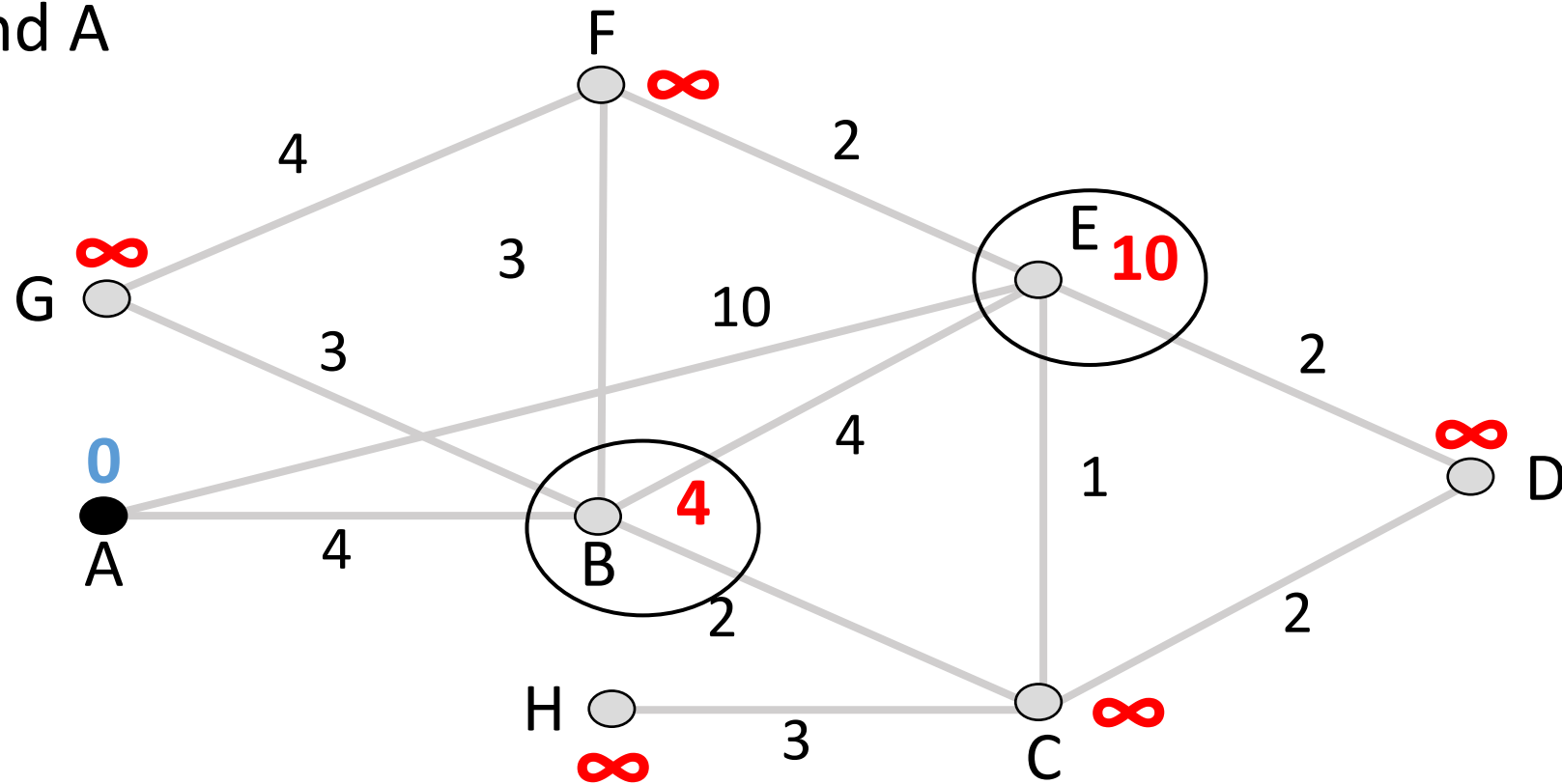
Dijkstra's Algorithm (2)

- Initialization



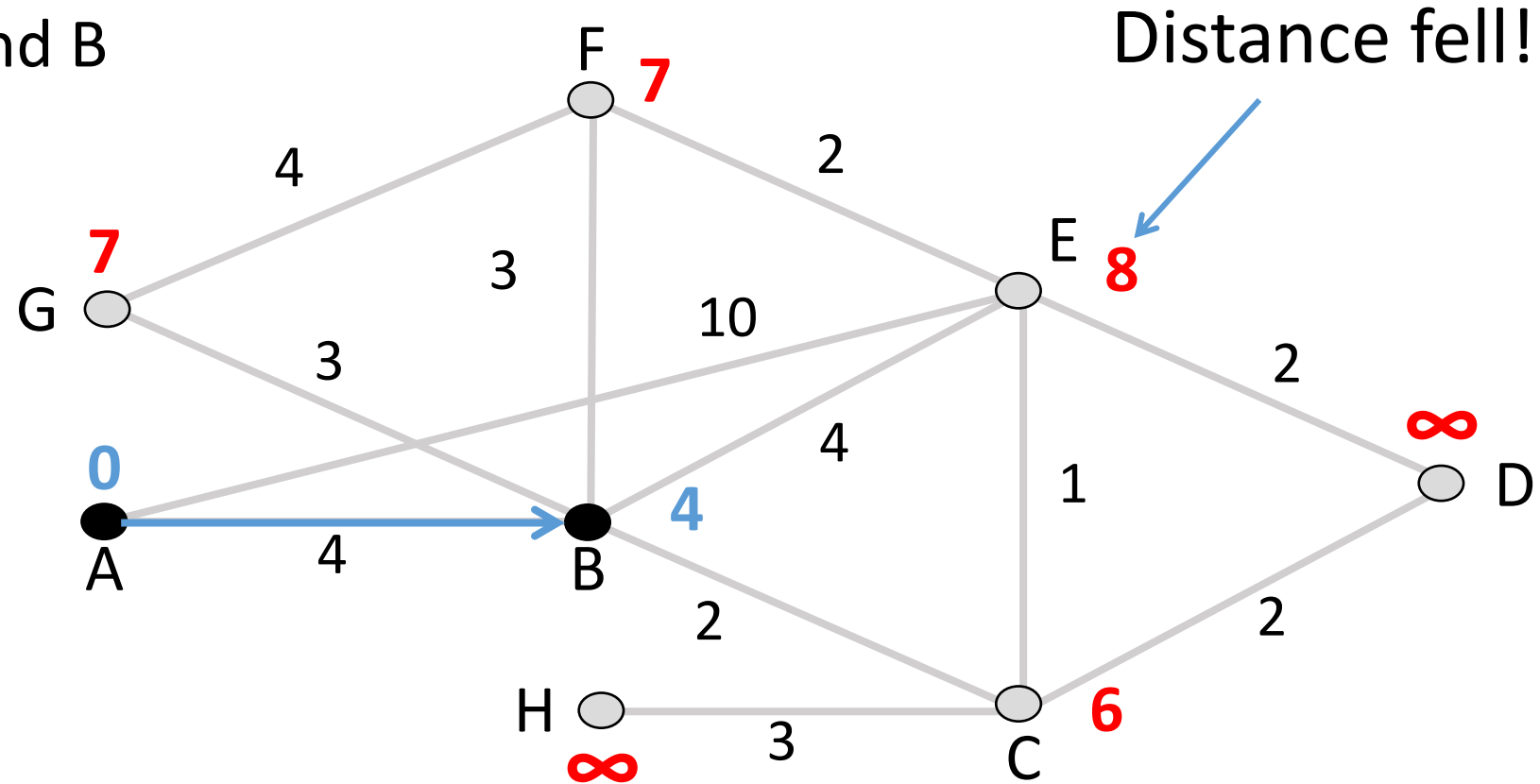
Dijkstra's Algorithm (3)

- Relax around A



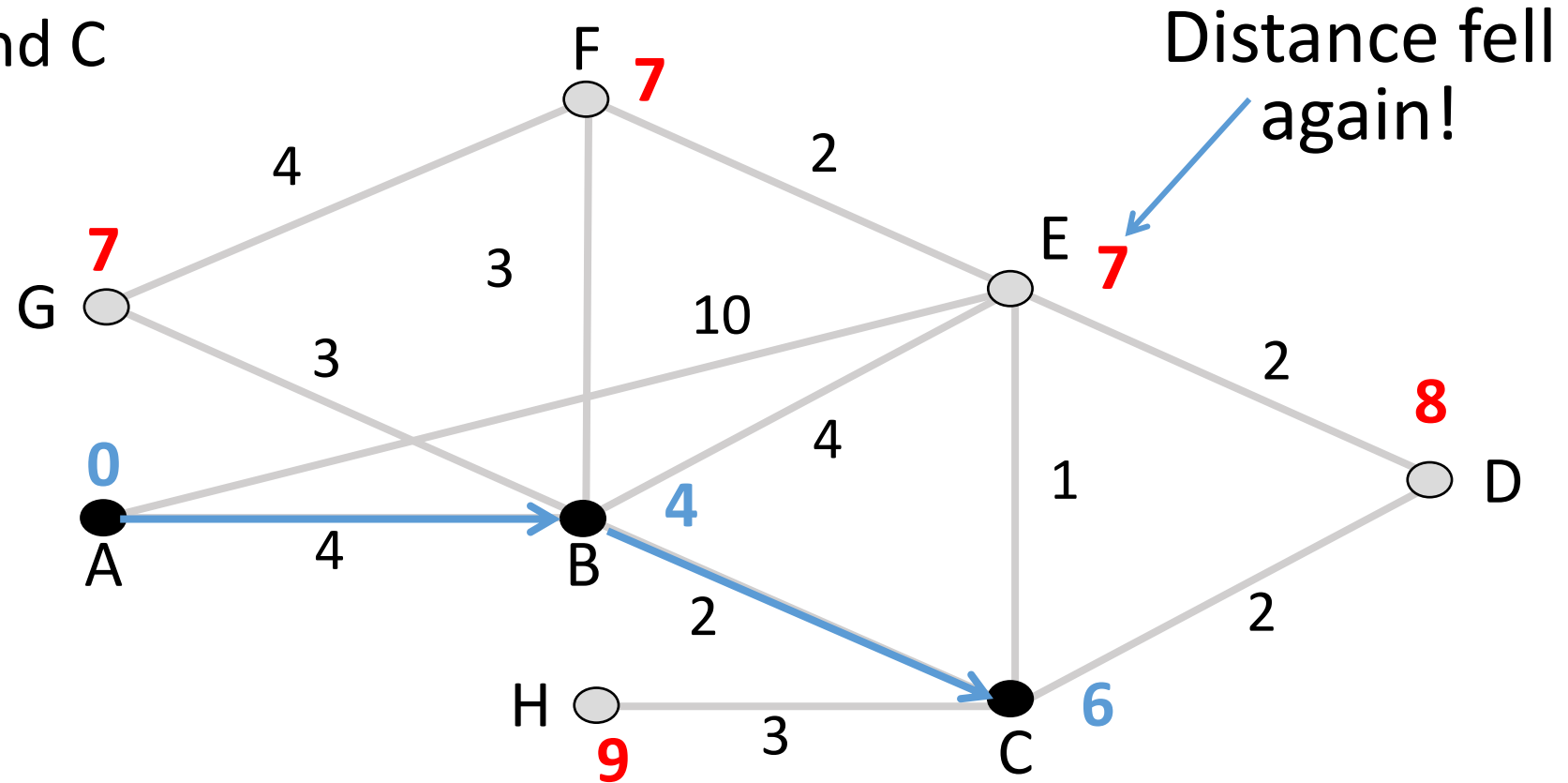
Dijkstra's Algorithm (4)

- Relax around B



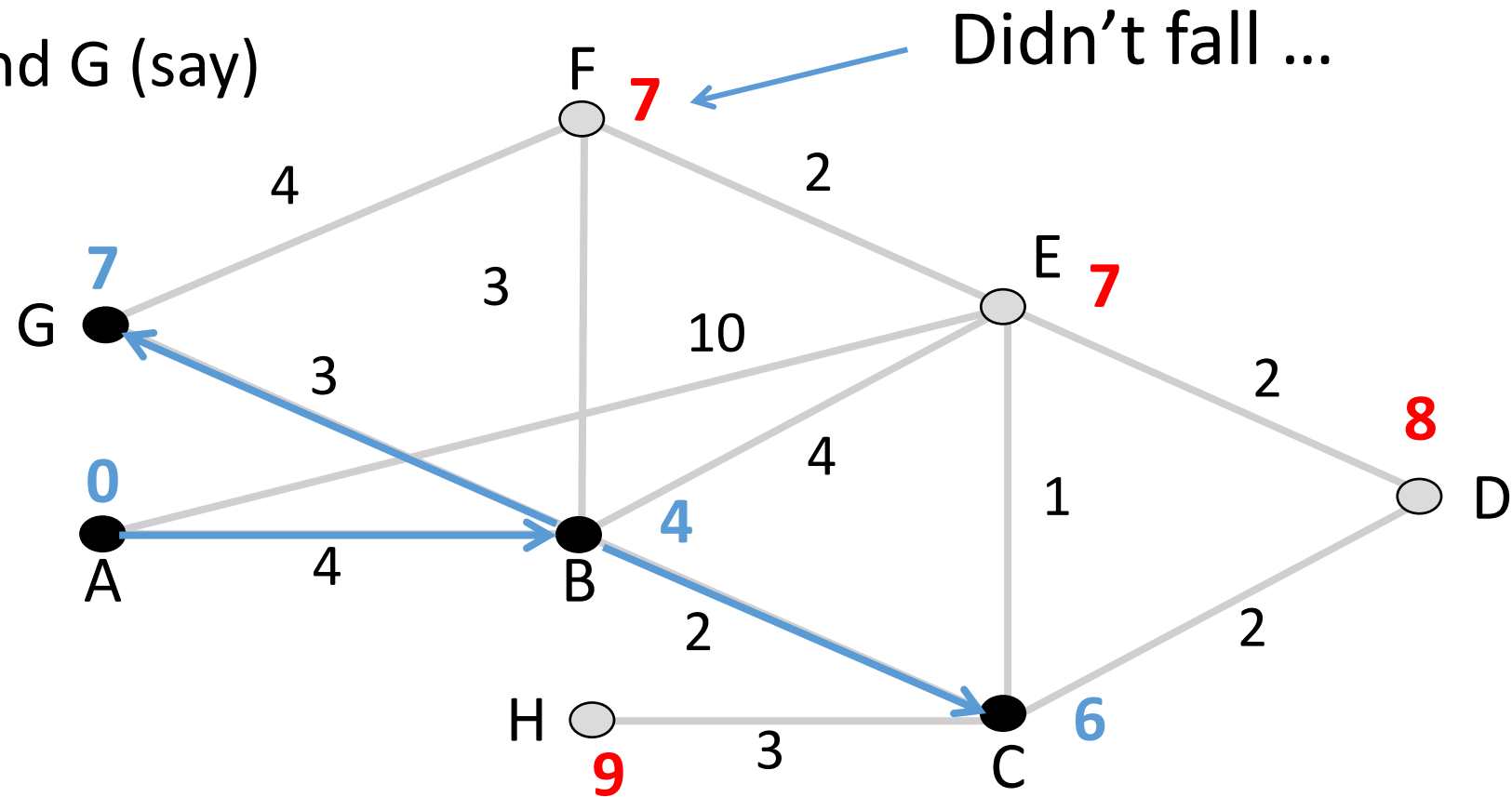
Dijkstra's Algorithm (5)

- Relax around C



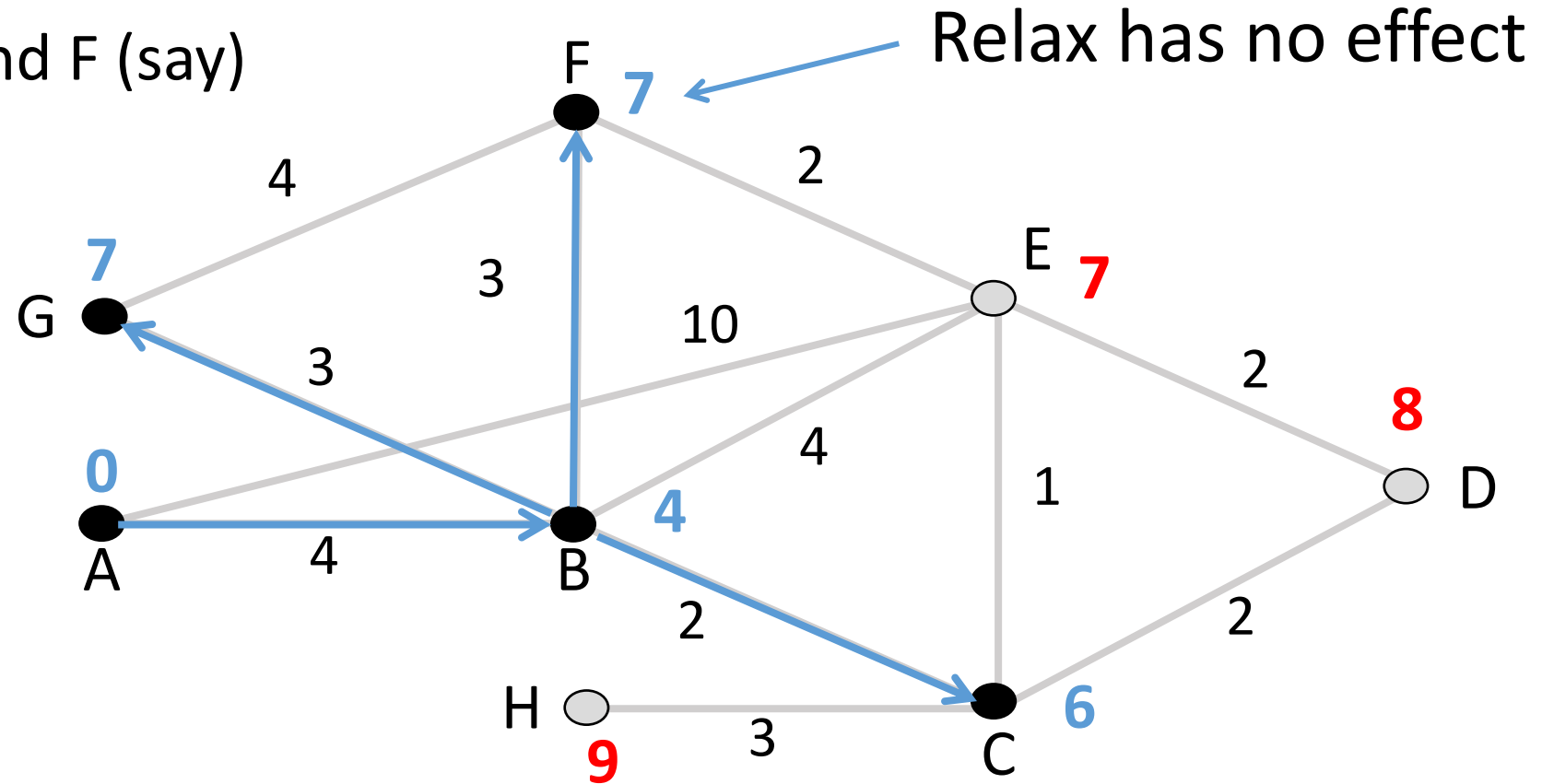
Dijkstra's Algorithm (6)

- Relax around G (say)



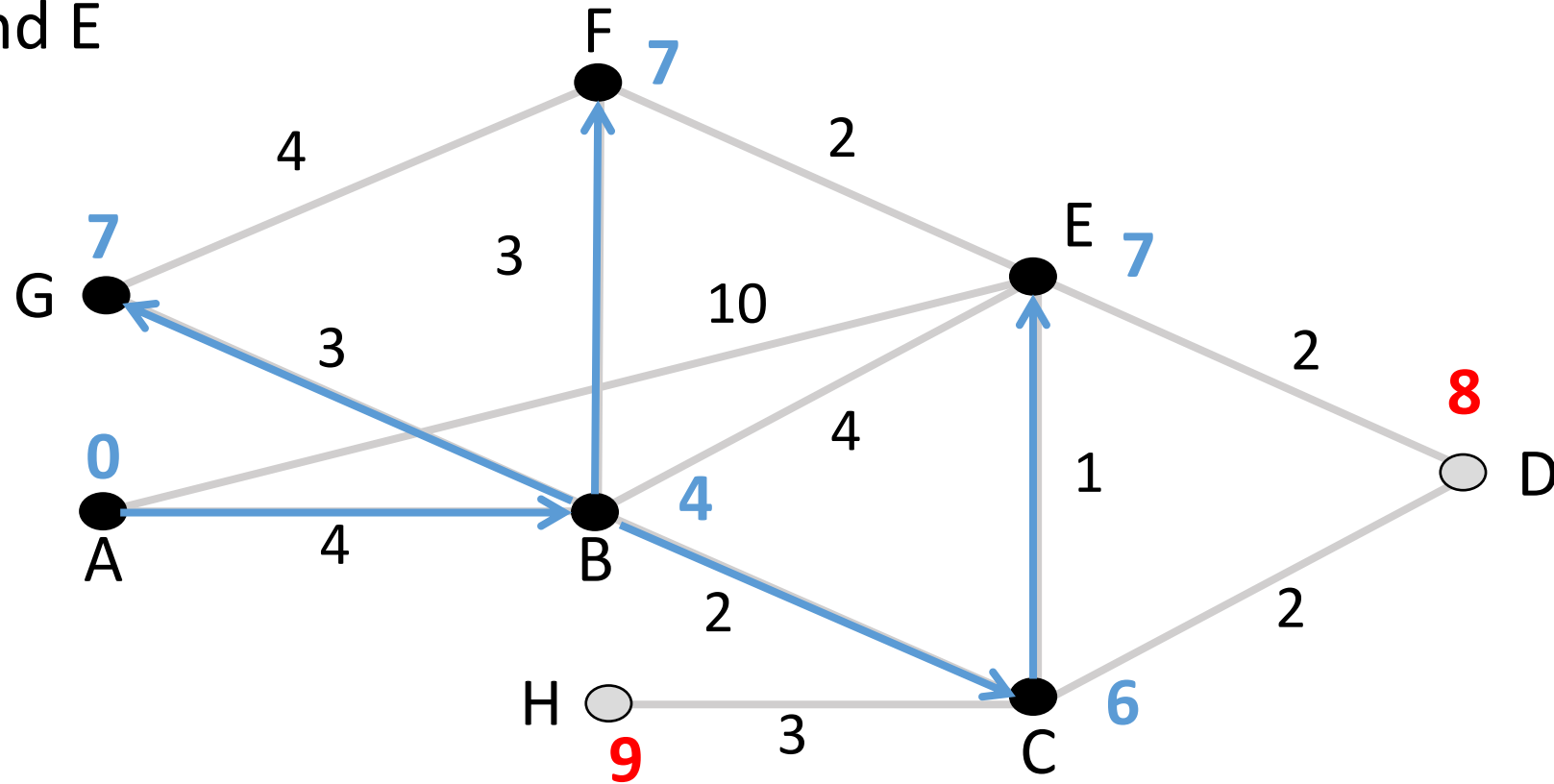
Dijkstra's Algorithm (7)

- Relax around F (say)



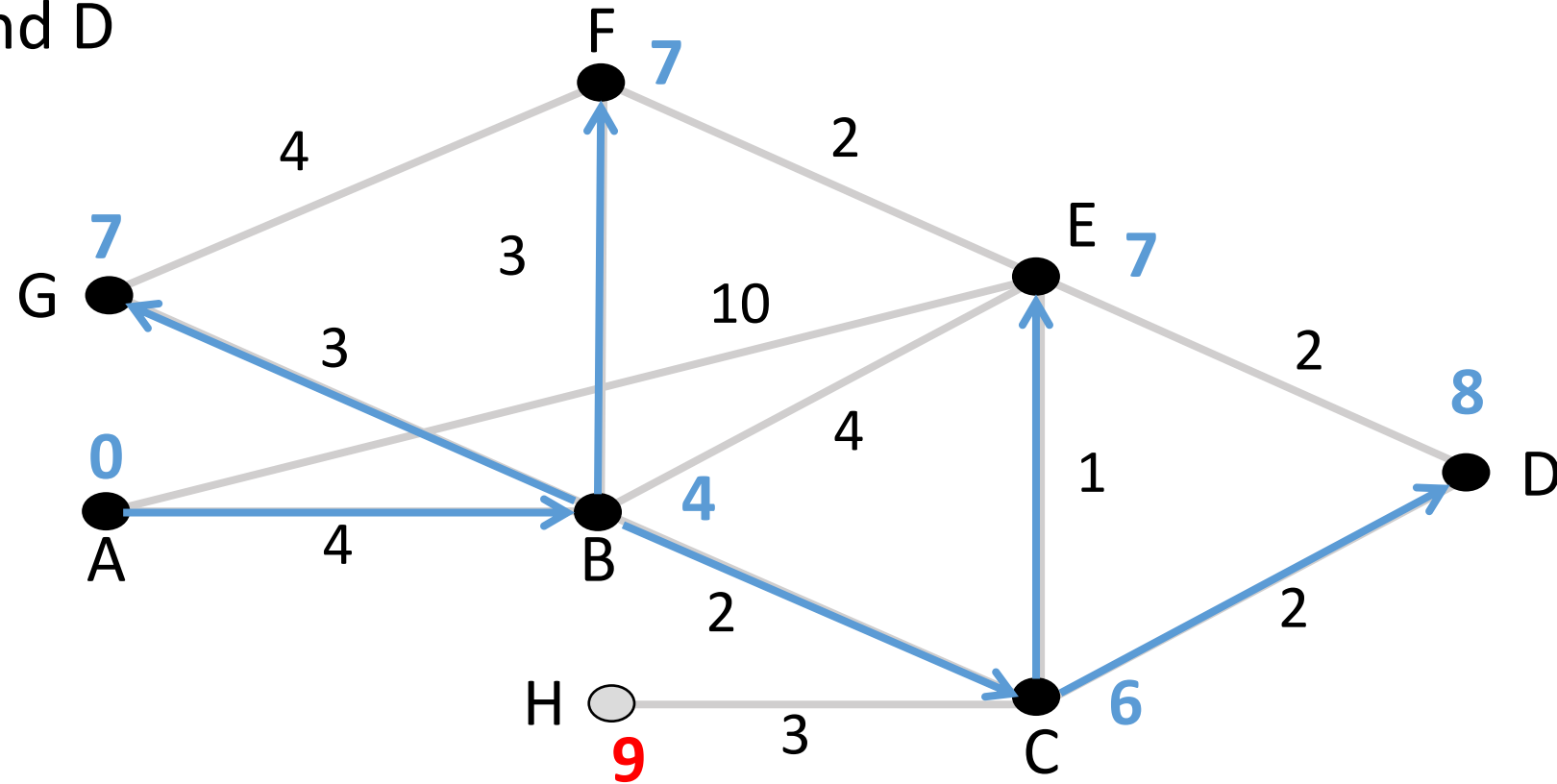
Dijkstra's Algorithm (8)

- Relax around E



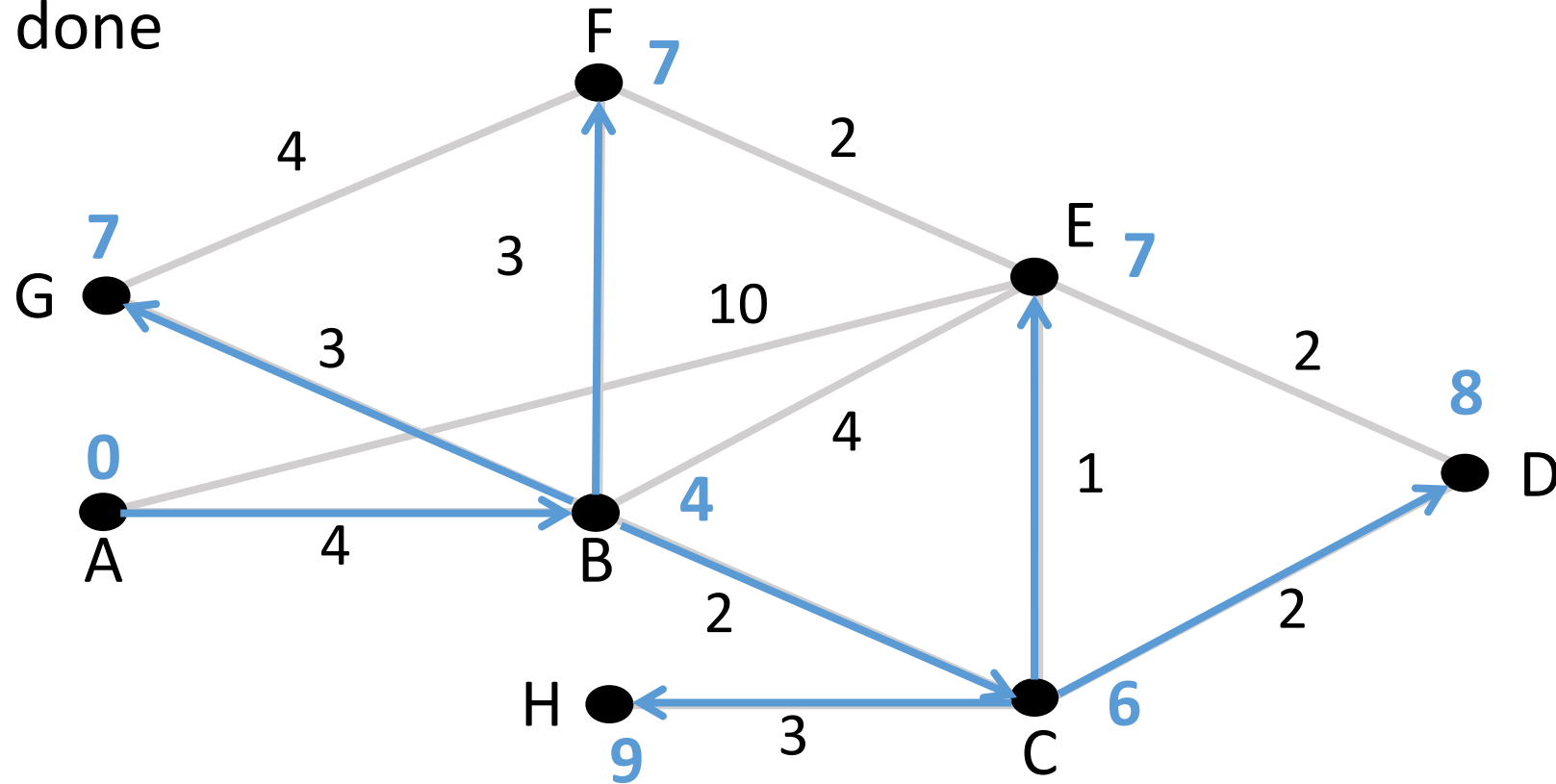
Dijkstra's Algorithm (9)

- 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

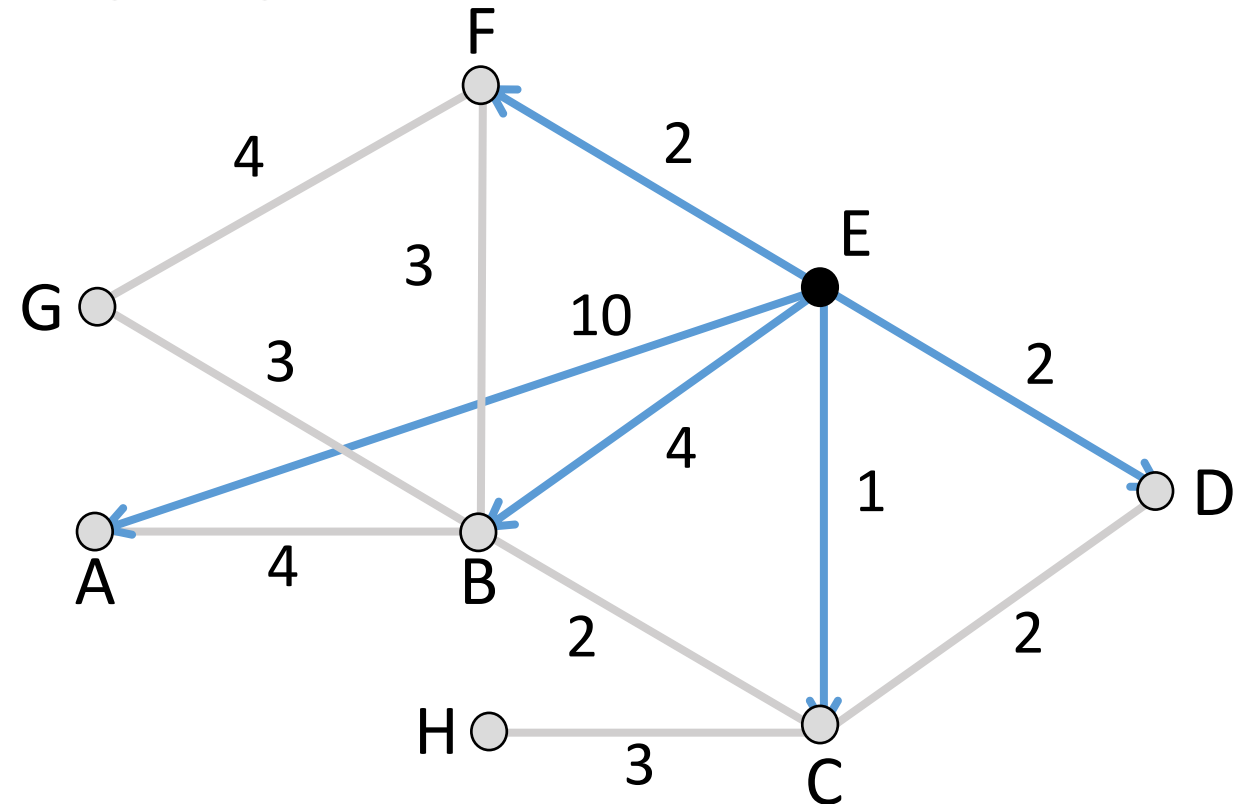
Bringing it all together...

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

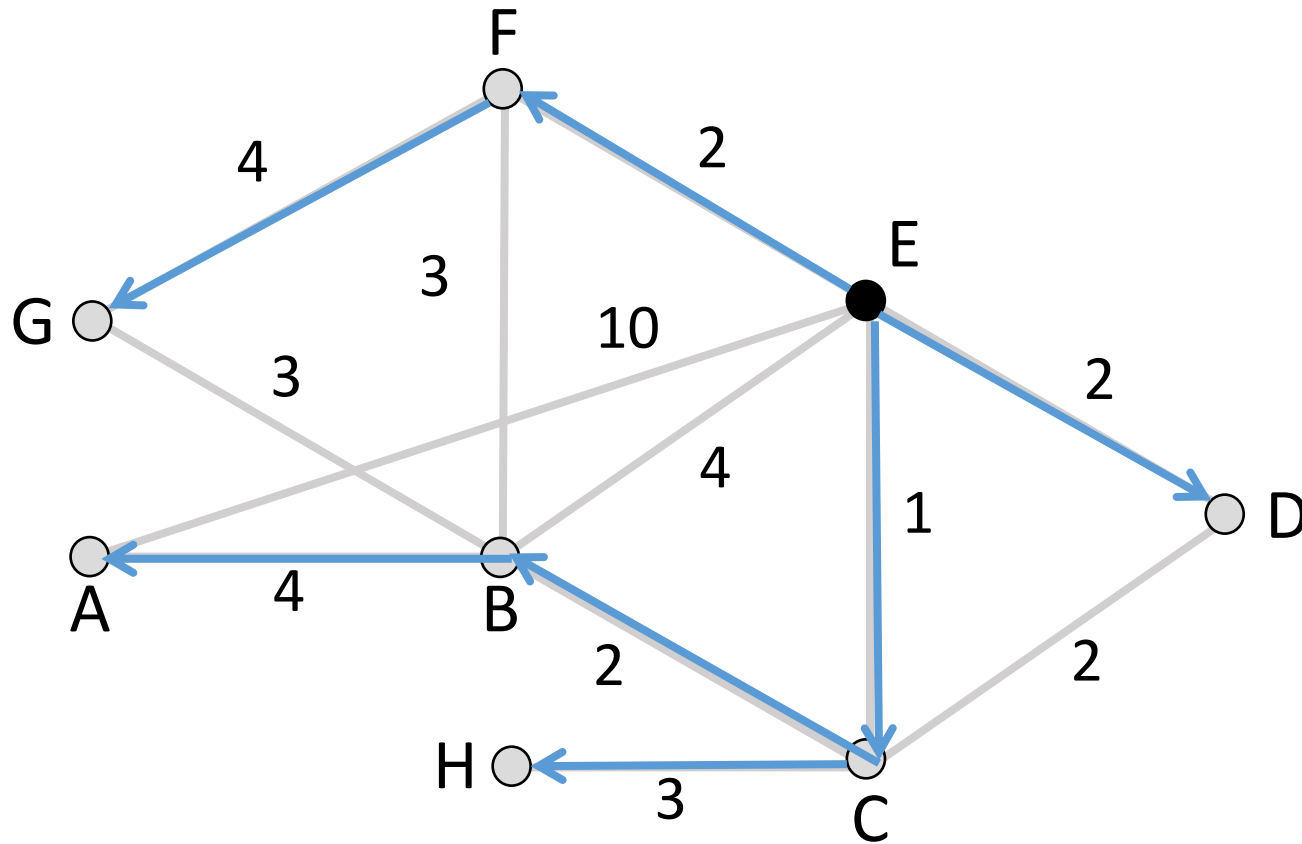


Phase 2: Route Computation

- 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

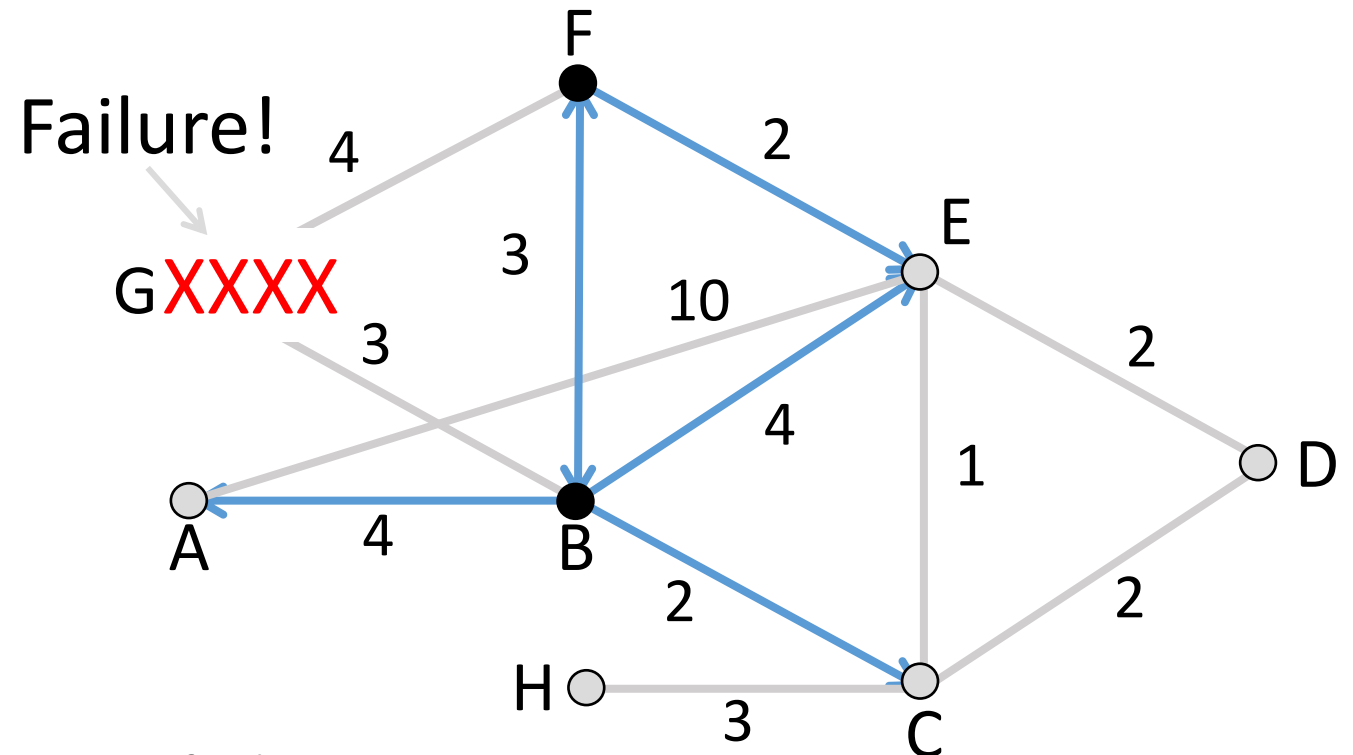
- On change, flood updated LSPs, re-compute routes
 - E.g., nodes adjacent to failed link or node initiate

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