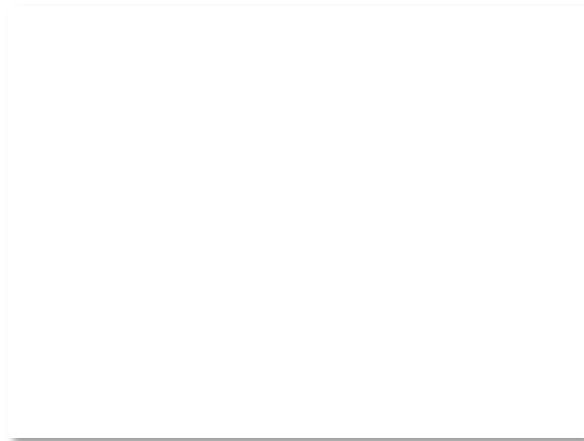
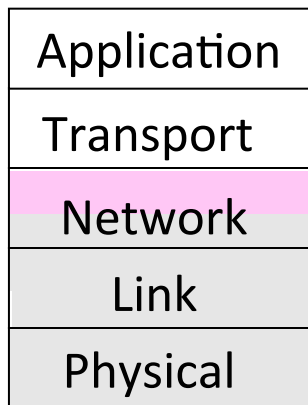


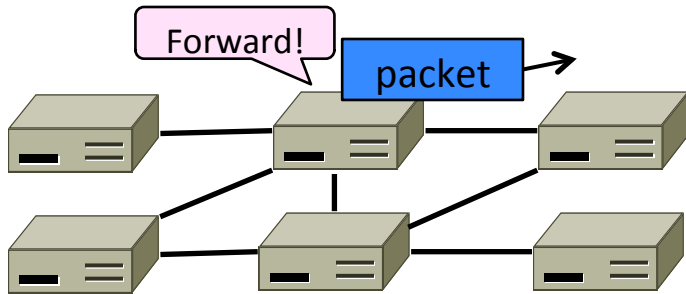
Where we are in the Course

- More fun in the Network Layer!
 - We've covered packet forwarding
 - Now we'll learn about routing

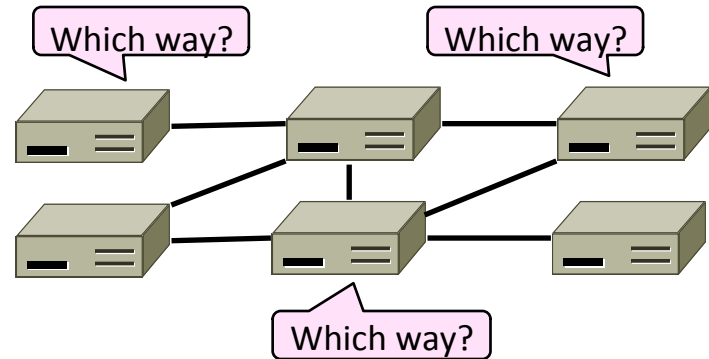


Routing versus Forwarding

- Forwarding is the process of sending a packet on its way

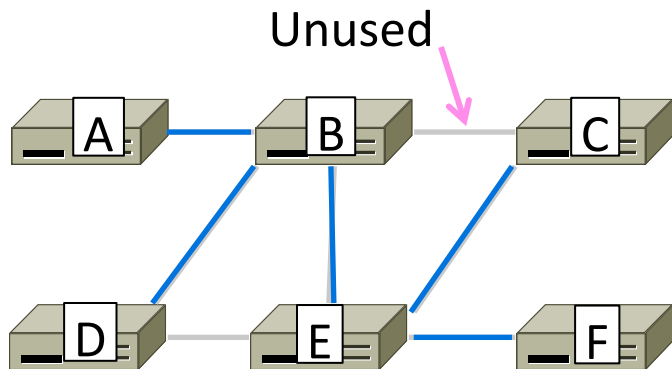


- Routing is the process of deciding in which direction to send traffic

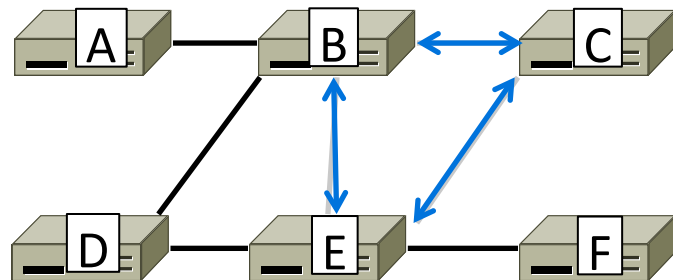


Improving on the Spanning Tree

- Spanning tree provides basic connectivity
 - e.g., some path $B \rightarrow C$



- Routing uses all links to find “best” paths
 - e.g., use BC, BE, and CE



Perspective on Bandwidth Allocation

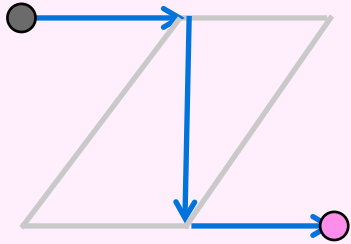
- Routing allocates network bandwidth adapting to failures; other mechanisms used at other timescales

| Mechanism | Timescale / Adaptation |
|------------------------|-------------------------------|
| Load-sensitive routing | Seconds / Traffic hotspots |
| Routing | Minutes / Equipment failures |
| Traffic Engineering | Hours / Network load |
| Provisioning | Months / Network customers |

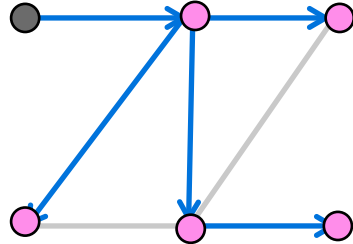
Delivery Models

- Different routing used for different delivery models

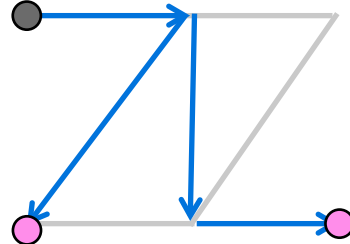
Unicast
(§5.2)



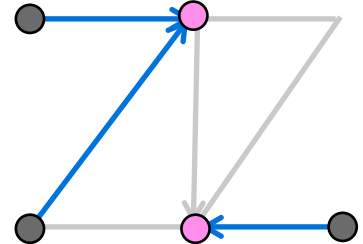
Broadcast
(§5.2.7)



Multicast
(§5.2.8)



Anycast
(§5.2.9)



Goals of Routing Algorithms

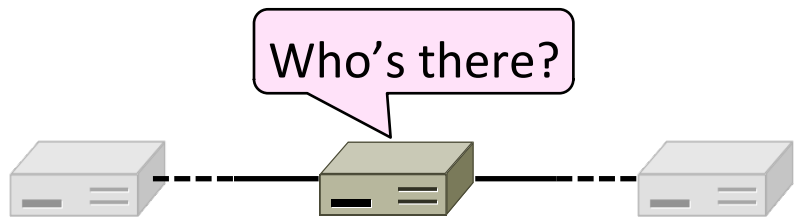
- We want several properties of any routing scheme:

| 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 |



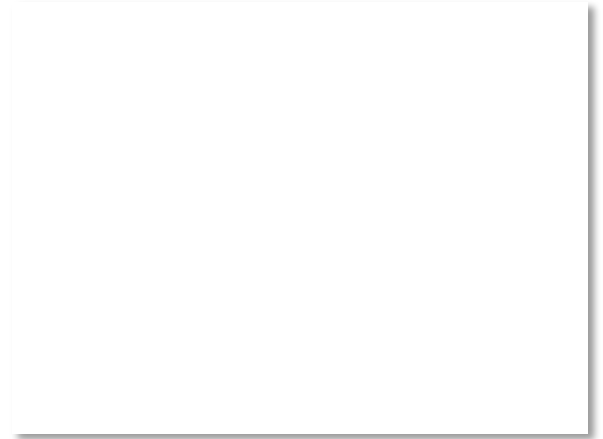
Rules of Routing Algorithms

- 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 failures



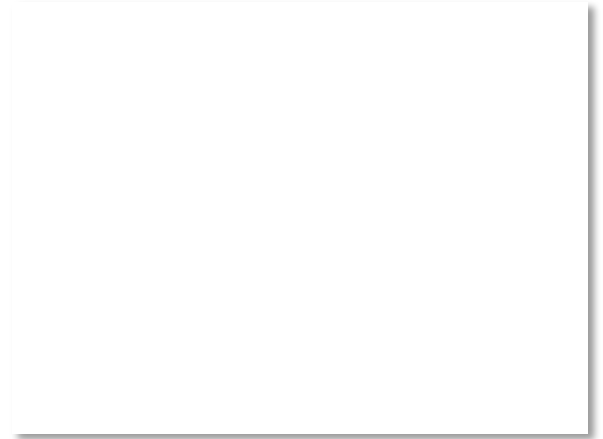
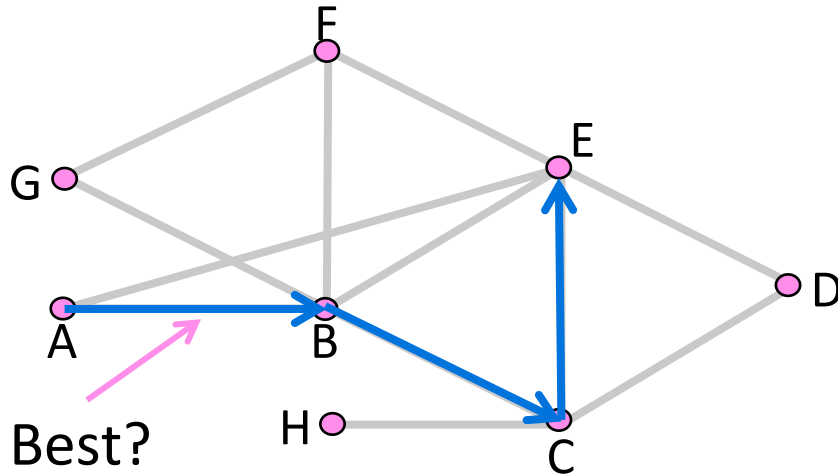
Topics

- IPv4, IPv6, NATs and all that } Last time
- Shortest path routing
- Distance Vector routing
- Flooding
- Link-state routing
- Equal-cost multi-path
- Inter-domain routing (BGP) } This time



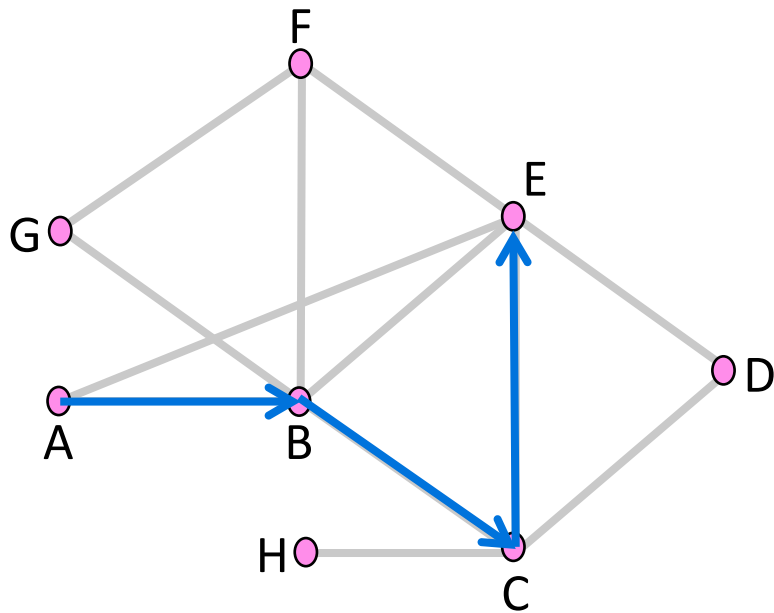
Topic

- Defining “best” paths with link costs
 - These are shortest path routes



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

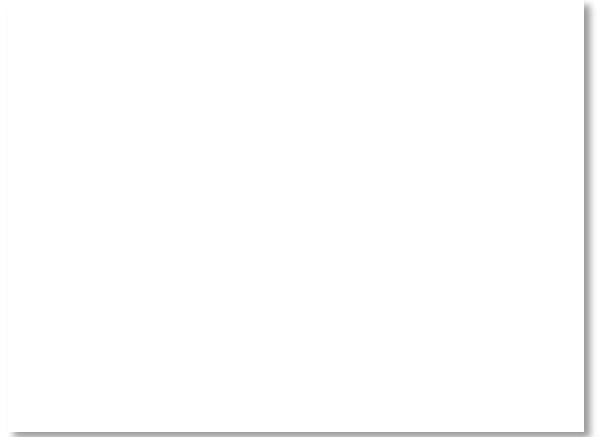


Shortest Paths

We'll approximate “best” by a cost function that captures the factors

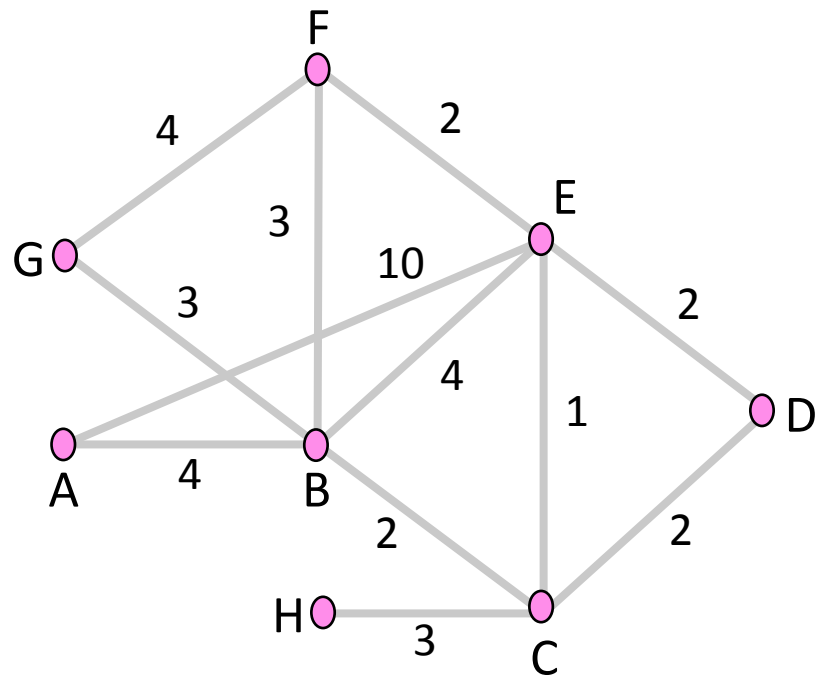
- Often call lowest “shortest”

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 ties



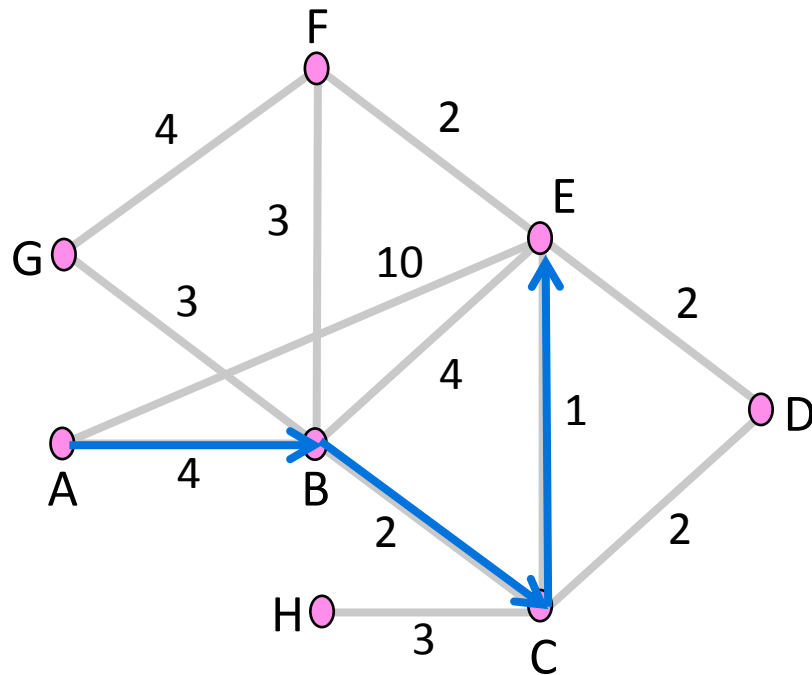
Shortest Paths (2)

- Find the shortest path $A \rightarrow E$
- All links are bidirectional, with equal costs in each direction
 - Can extend model to unequal costs if needed



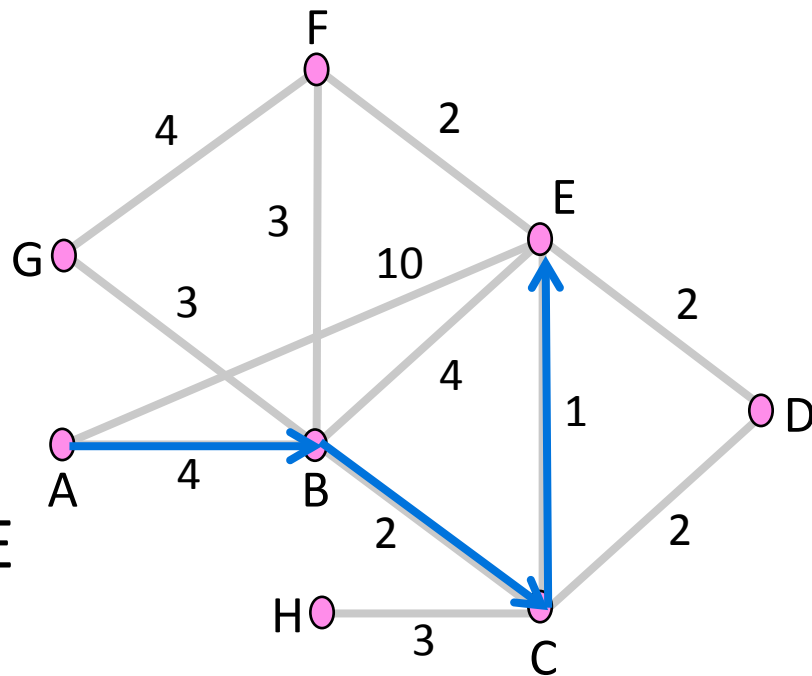
Shortest Paths (3)

- ABCE is a shortest path
- $\text{dist}(\text{ABCE}) = 4 + 2 + 1 = 7$
- This is less than:
 - $\text{dist}(\text{ABE}) = 8$
 - $\text{dist}(\text{ABFE}) = 9$
 - $\text{dist}(\text{AE}) = 10$
 - $\text{dist}(\text{ABCDE}) = 10$



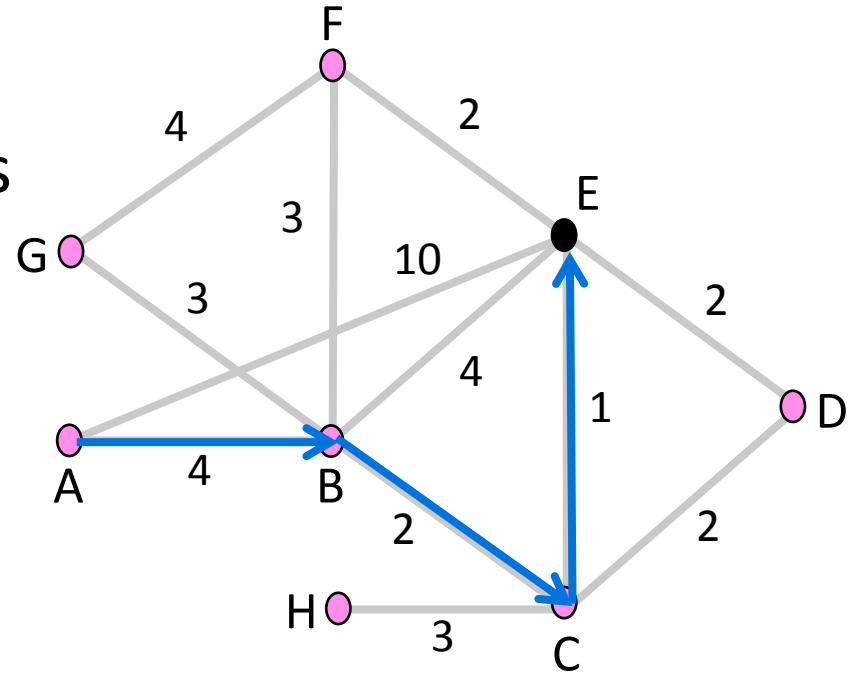
Shortest Paths (4)

- Optimality property:
 - Subpaths of shortest paths are also shortest paths
- ABCE is a shortest path
 - So are ABC, AB, BCE, BC, CE



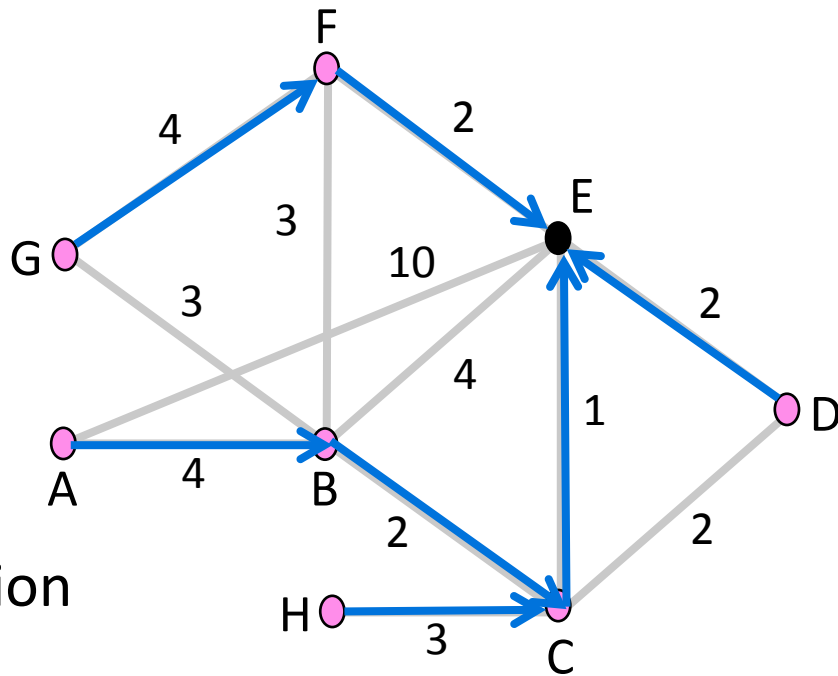
Sink Trees

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



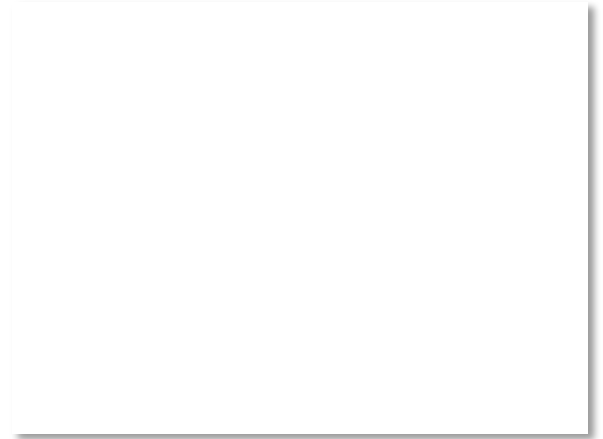
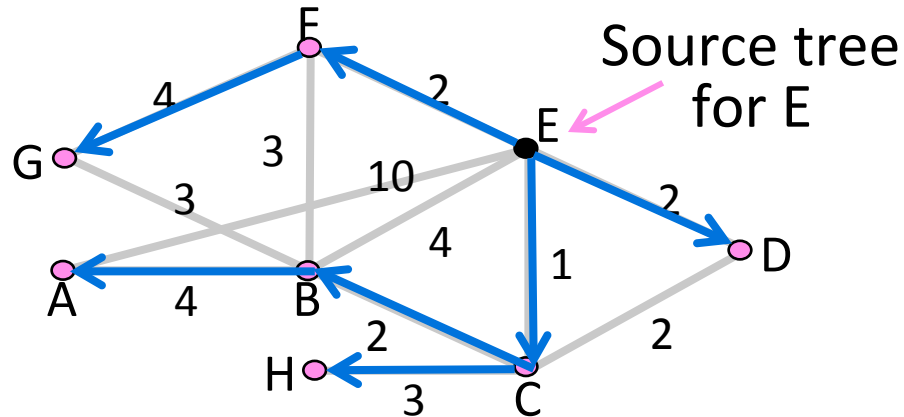
Sink Trees (2)

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



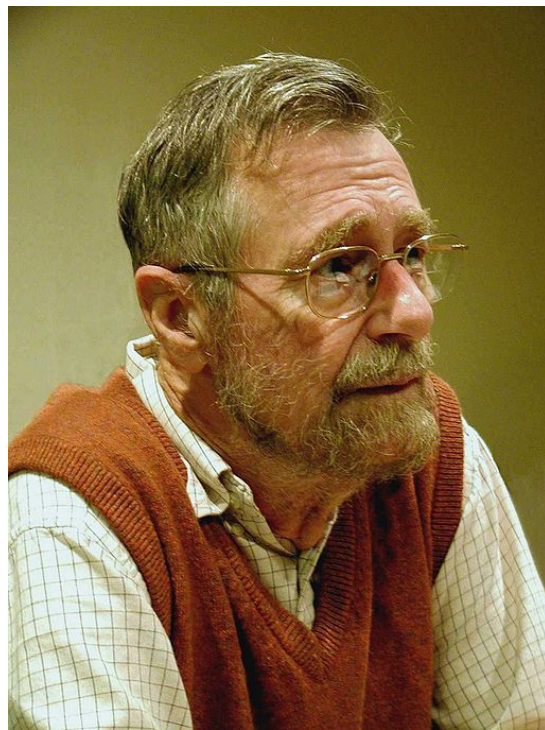
Topic

- How to compute shortest paths given the network topology
 - With 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

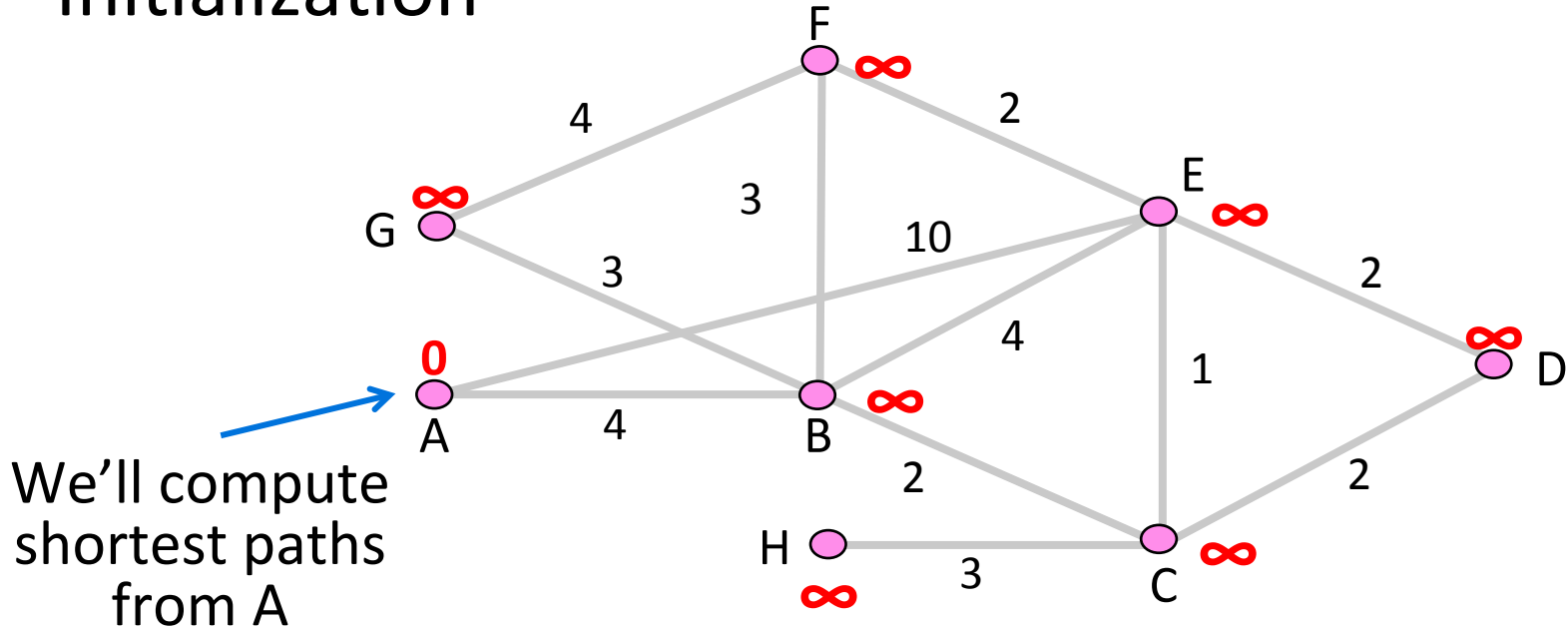
Algorithm:

- 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



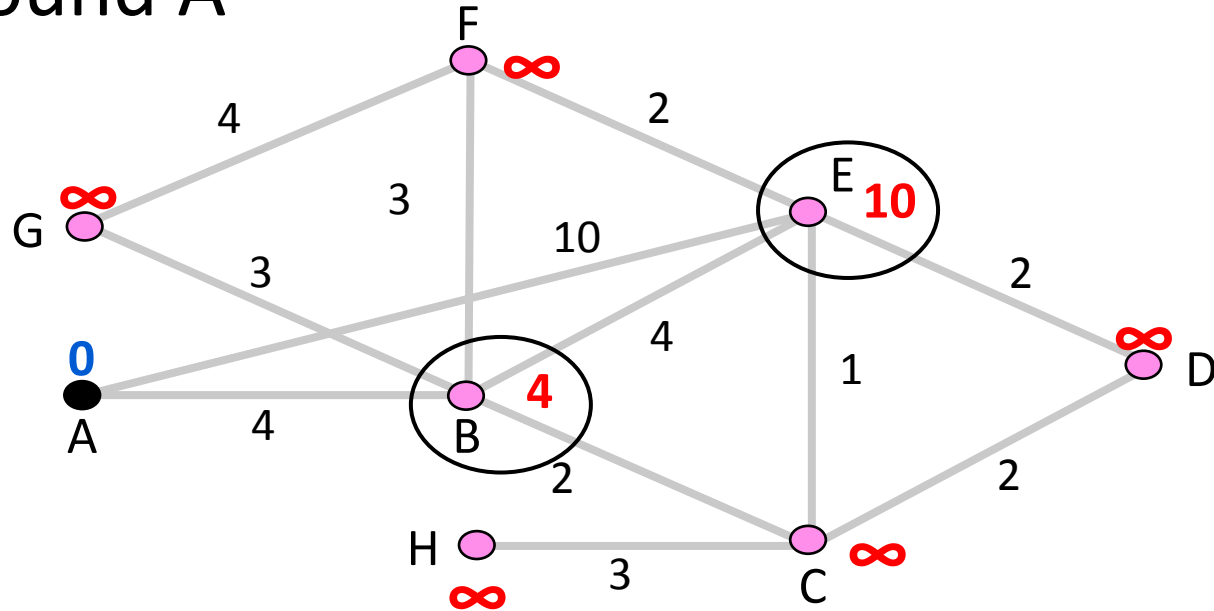
Dijkstra's Algorithm (2)

- Initialization



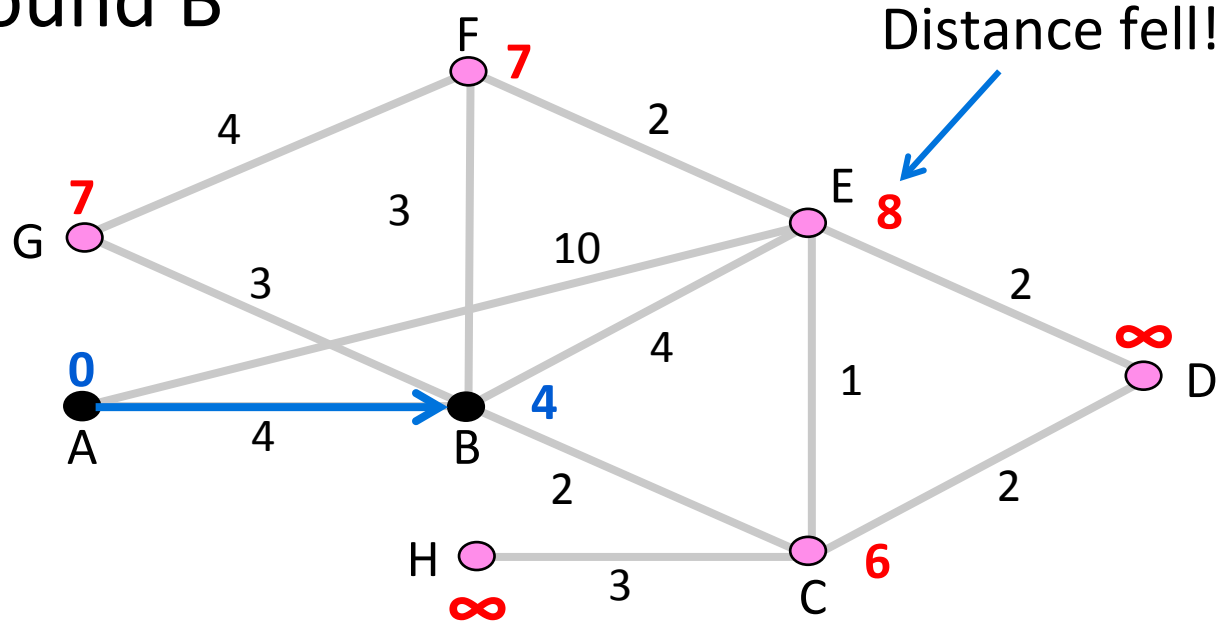
Dijkstra's Algorithm (3)

- Relax around A



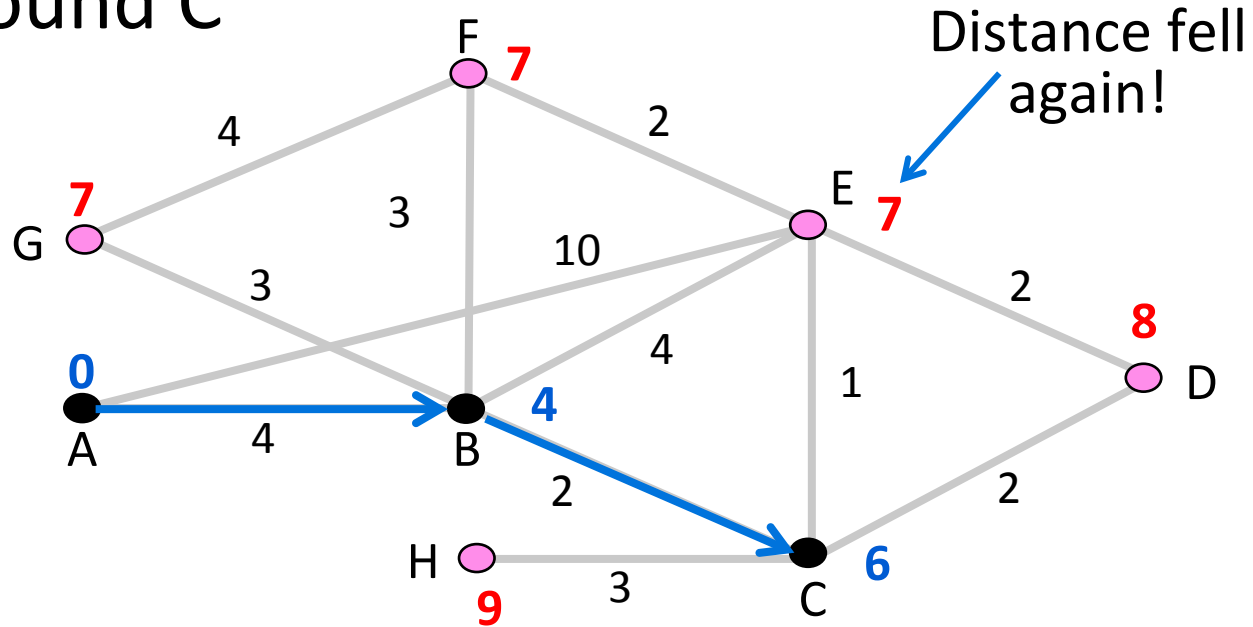
Dijkstra's Algorithm (4)

- Relax around B



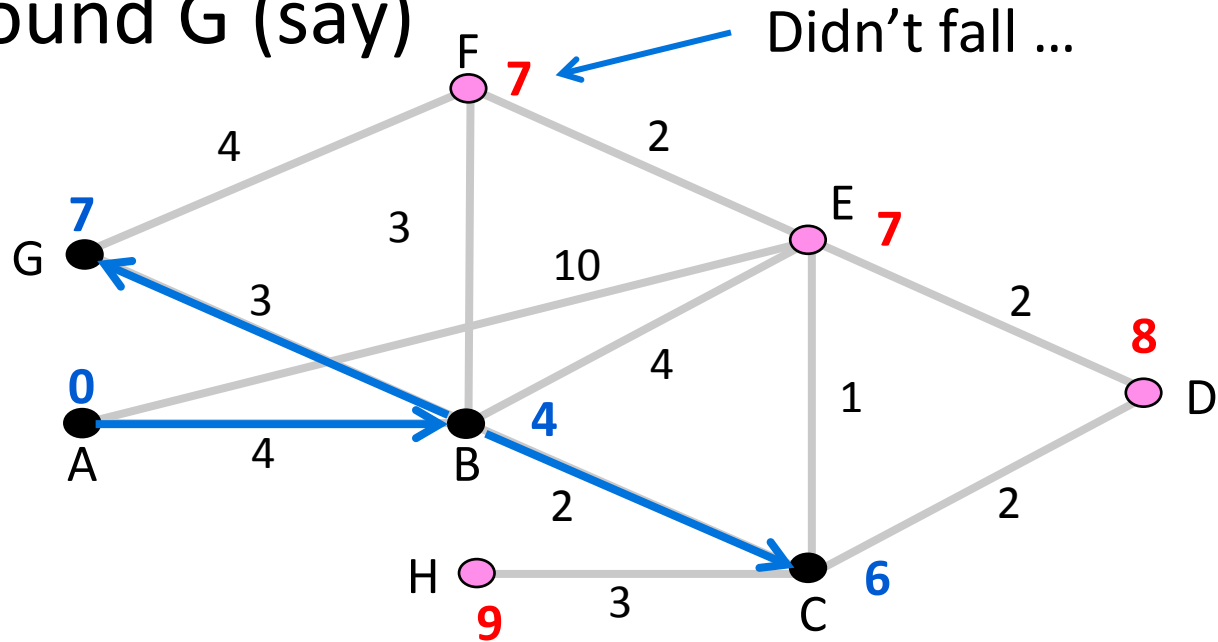
Dijkstra's Algorithm (5)

- Relax around C



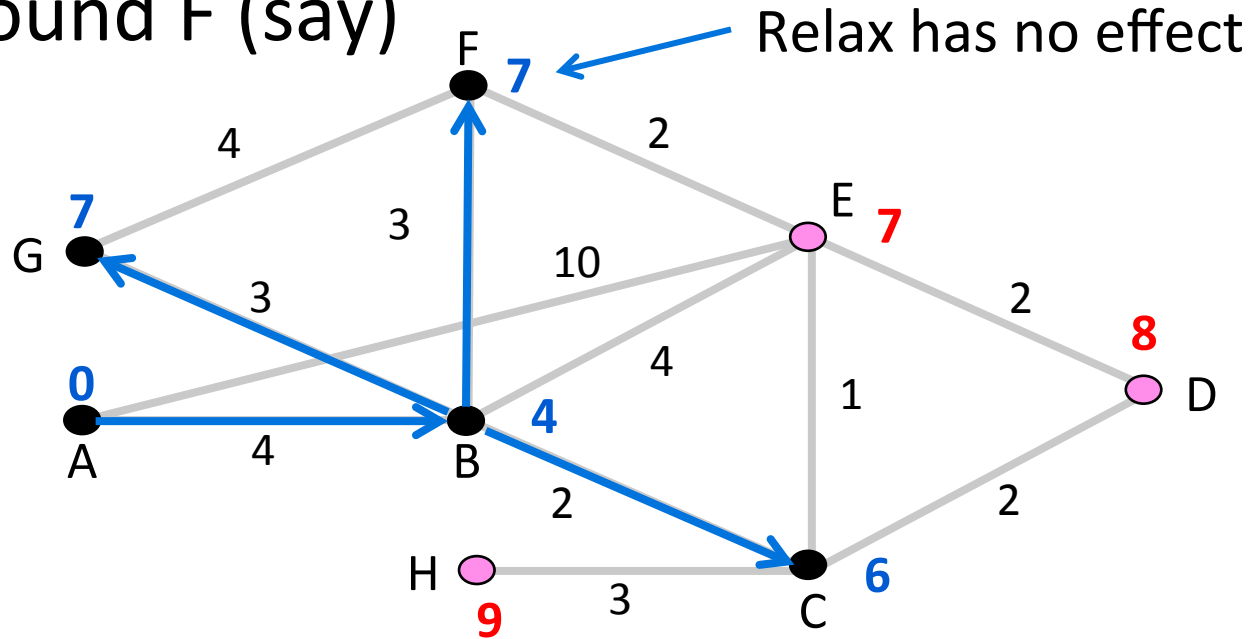
Dijkstra's Algorithm (6)

- Relax around G (say)



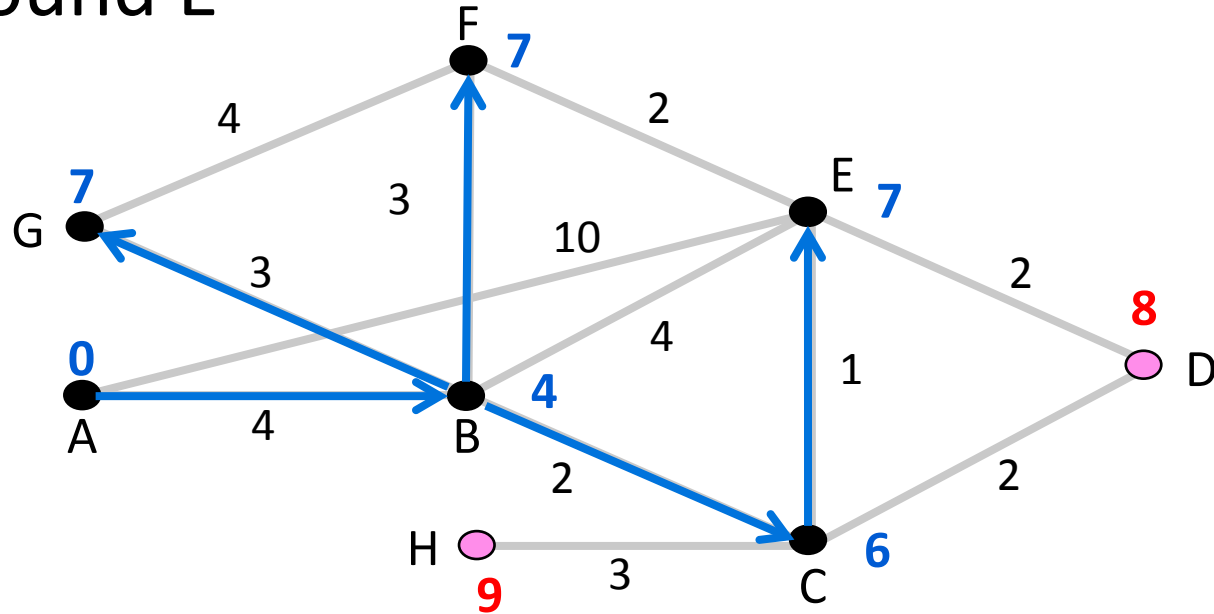
Dijkstra's Algorithm (7)

- Relax around F (say)



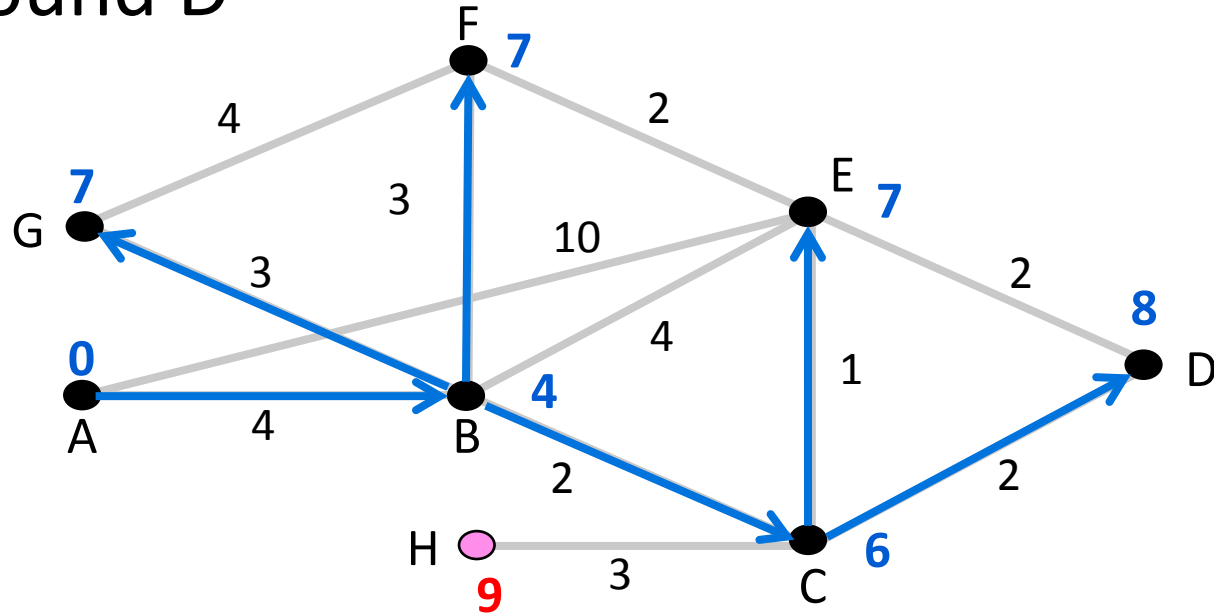
Dijkstra's Algorithm (8)

- Relax around E



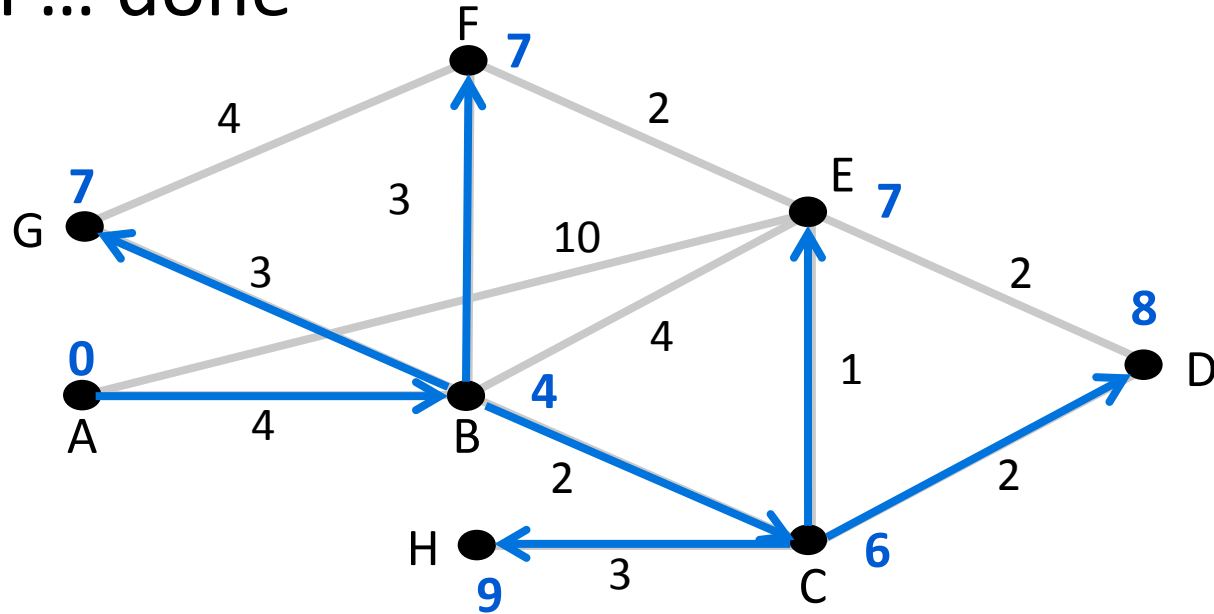
Dijkstra's Algorithm (9)

- Relax around D



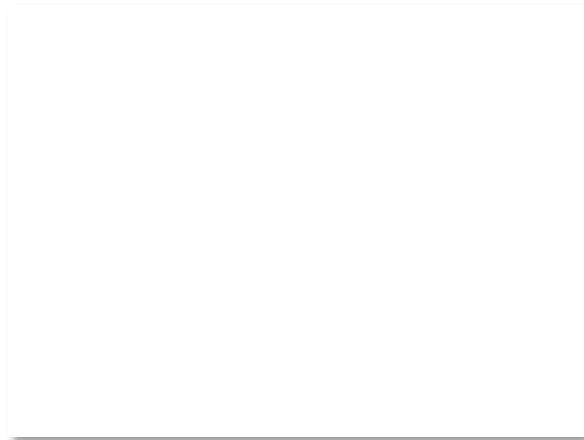
Dijkstra's Algorithm (10)

- Finally, H ... done



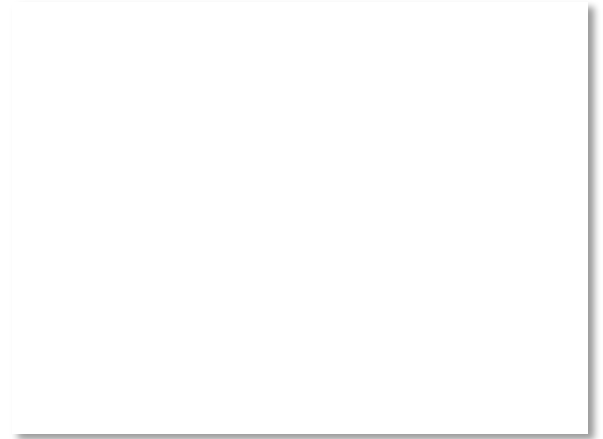
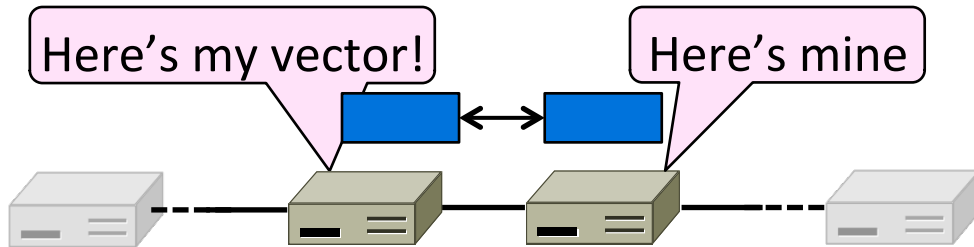
Dijkstra Comments

- Finds shortest paths in order of increasing distance from source
 - Leverages optimality property
- Runtime depends on 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



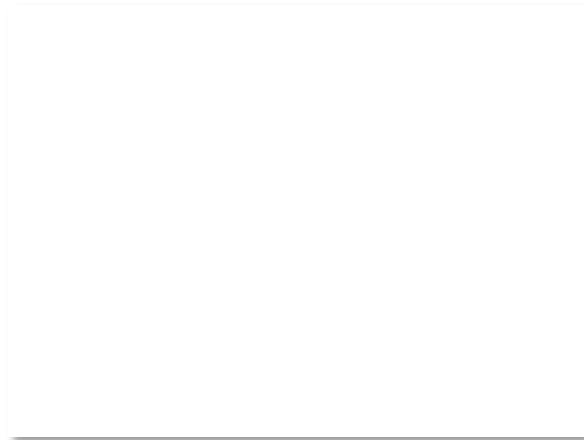
Topic

- How to compute shortest paths in a distributed network
 - The Distance Vector (DV) approach



Distance Vector Routing

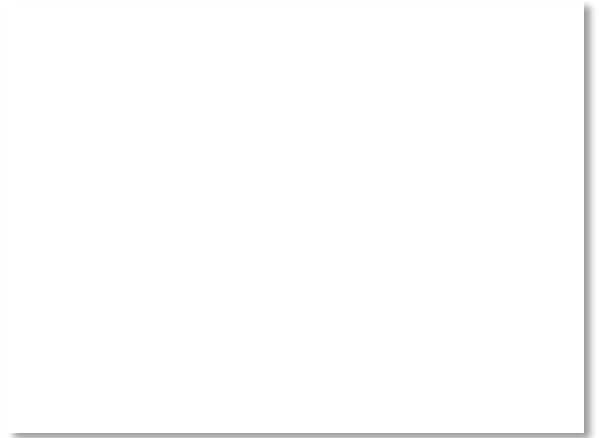
- Simple, early routing approach
 - Used in ARPANET, and RIP
- One of two main approaches to routing
 - Distributed version of Bellman-Ford
 - Works, but very slow convergence after some failures
- Link-state algorithms are now typically used in practice
 - More involved, better behavior



Distance Vector Setting

Each node computes its forwarding table in a distributed setting:

1. Nodes know only the cost to their neighbors; not 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

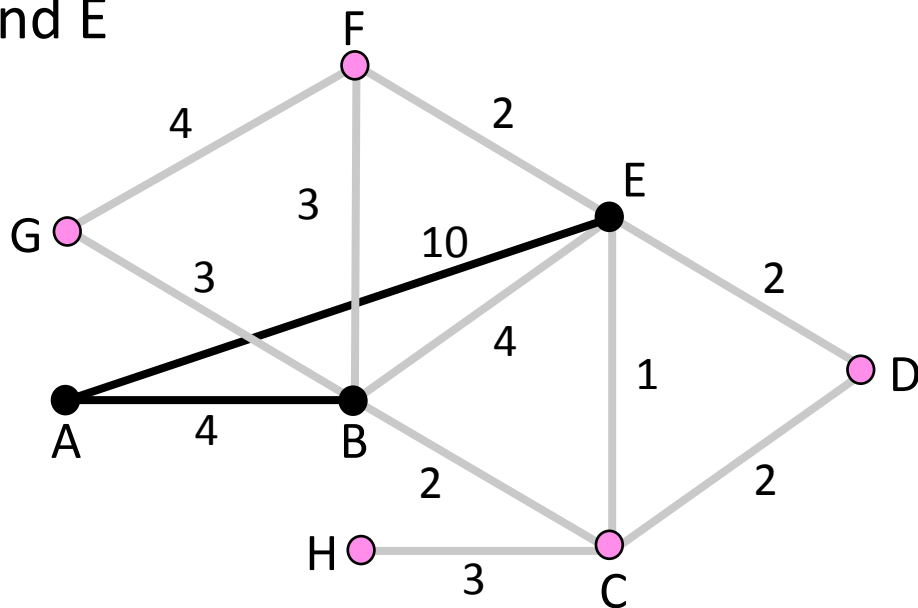
1. Initialize vector with 0 (zero) cost to self, ∞ (infinity) to other destinations
2. Periodically send vector to neighbors
3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
 - Use the best neighbor for forwarding

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 | ∞ | ∞ |

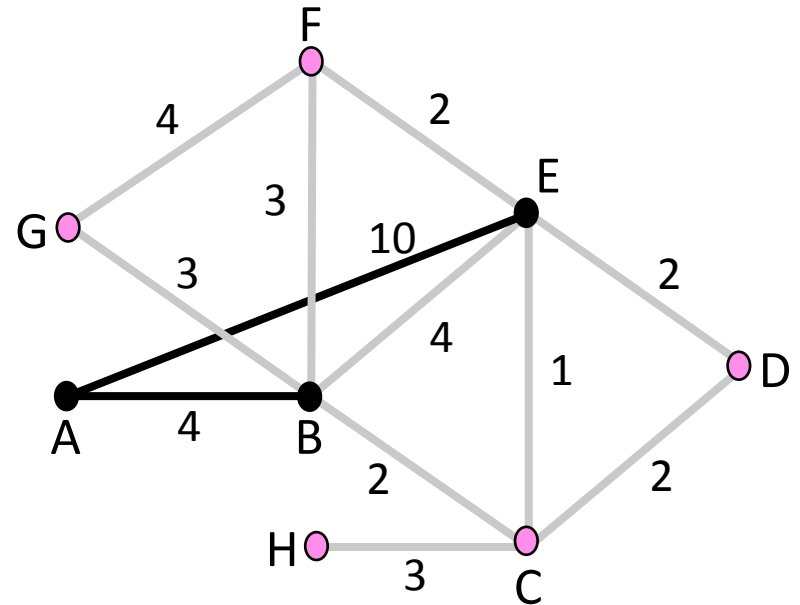
→

| 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

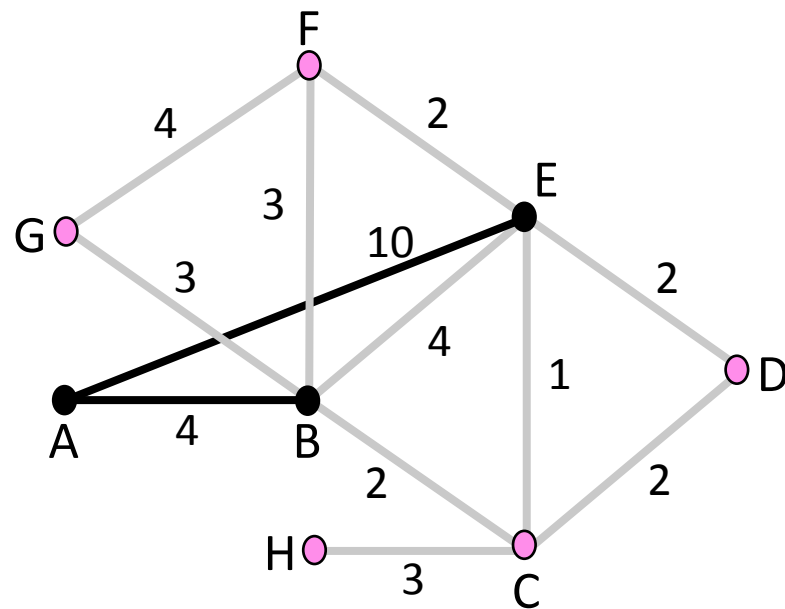
| To | B says | E says |
|----|-----------|-----------|
| A | 4 | 10 |
| B | 0 | 4 |
| C | 2 | 1 |
| D | ∞ | 2 |
| E | 4 | 0 |
| F | 3 | 2 |
| G | 3 | ∞ |
| H | ∞ | ∞ |



| 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

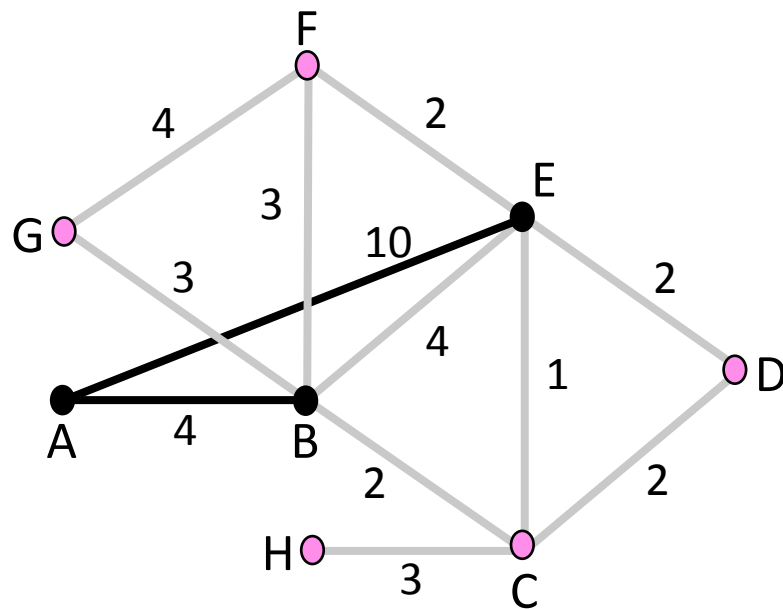
| To | B says | E says |
|----|-----------|-----------|
| A | 4 | 8 |
| B | 0 | 3 |
| C | 2 | 1 |
| D | 4 | 2 |
| E | 3 | 0 |
| F | 3 | 2 |
| G | 3 | 6 |
| H | 5 | 4 |



| 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

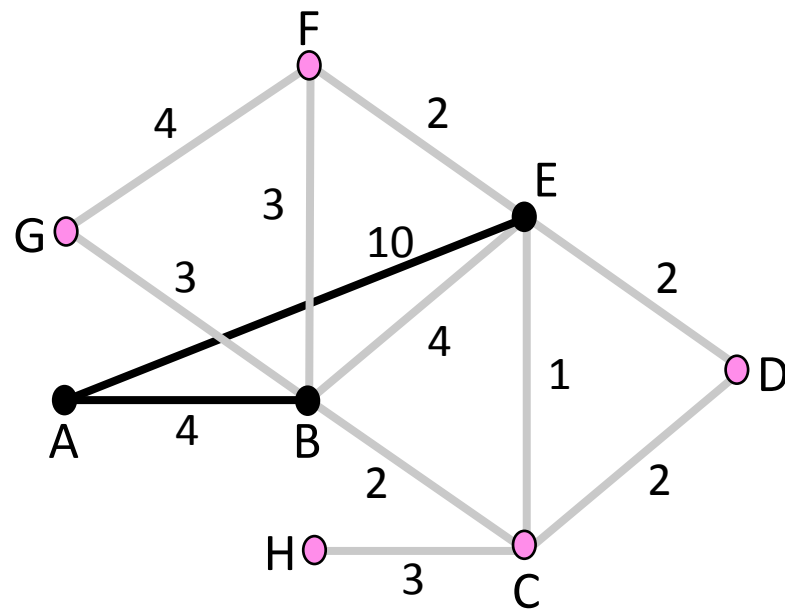
| To | B says | E says |
|----|-----------|-----------|
| A | 4 | 7 |
| B | 0 | 3 |
| C | 2 | 1 |
| D | 4 | 2 |
| E | 3 | 0 |
| F | 3 | 2 |
| G | 3 | 6 |
| H | 5 | 4 |



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

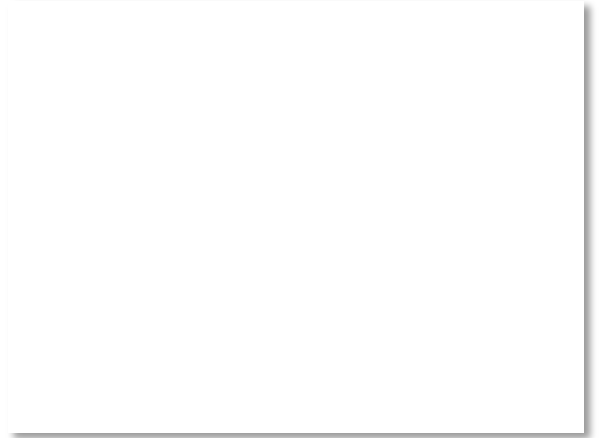


| A's Cost | A's Next |
|-------------|-------------|
| 0 | -- |
| 4 | B |
| 6 | B |
| 8 | B |
| 8 | B |
| 7 | B |
| 7 | B |
| 9 | B |



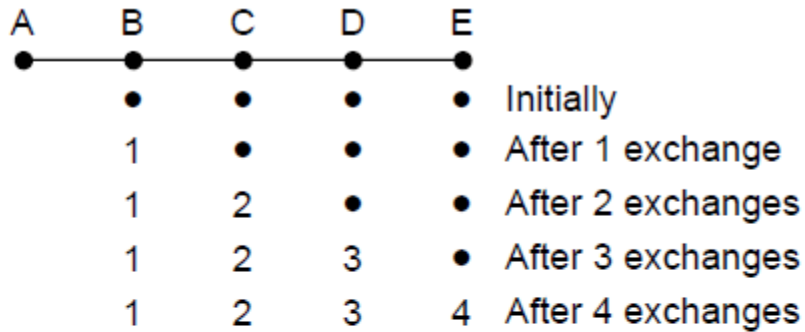
Distance Vector Dynamics

- Adding routes:
 - News travels one hop per exchange
- Removing routes
 - When a node fails, no more exchanges, other nodes forget
- But partitions (unreachable nodes in divided network) are a problem
 - “Count to infinity” scenario

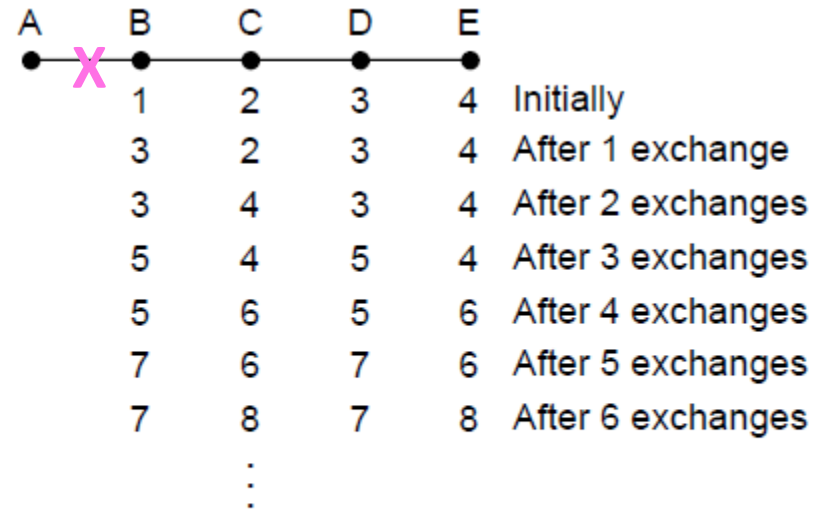


DV Dynamics (2)

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



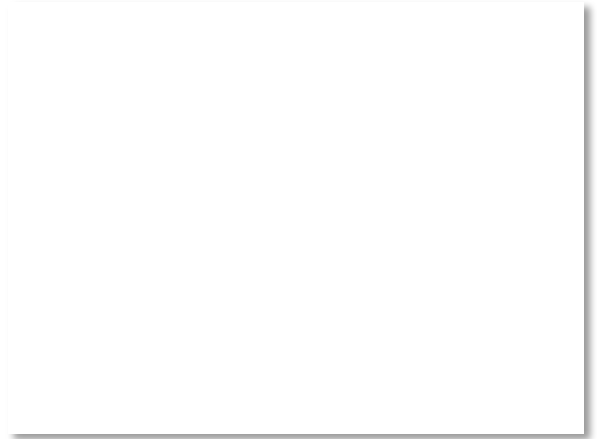
Desired convergence



“Count to infinity” scenario

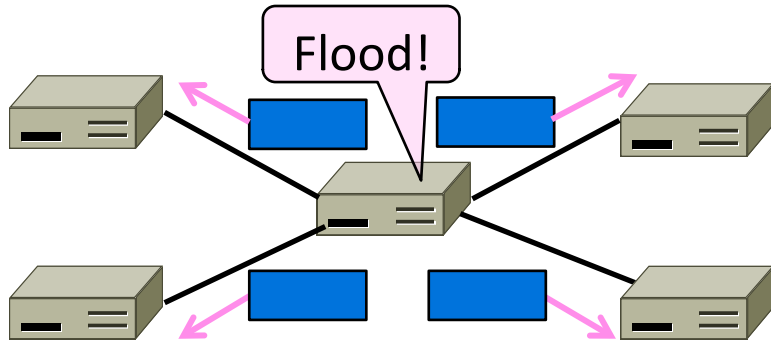
DV Dynamics (3)

- 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



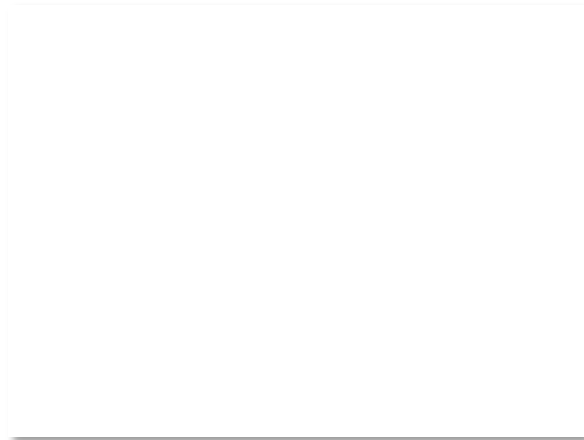
Topic

- How to broadcast a message to all nodes in the network with flooding
 - Simple mechanism, but inefficient



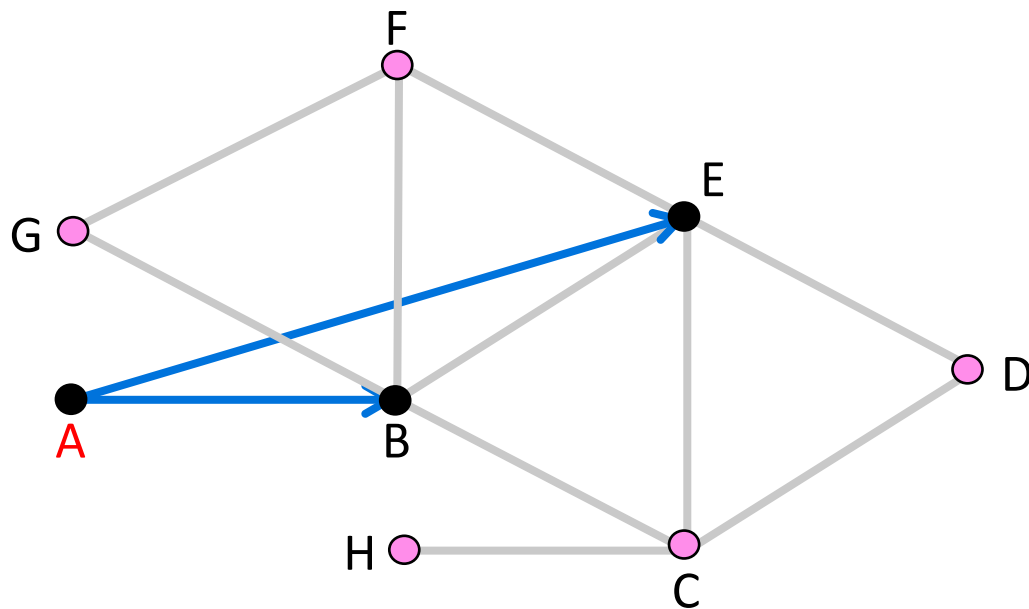
Flooding

- Rule used at each node:
 - Sends an incoming message on to all other neighbors
 - Remember the message so that it is only flood once
- Inefficient because one node may receive multiple copies of message



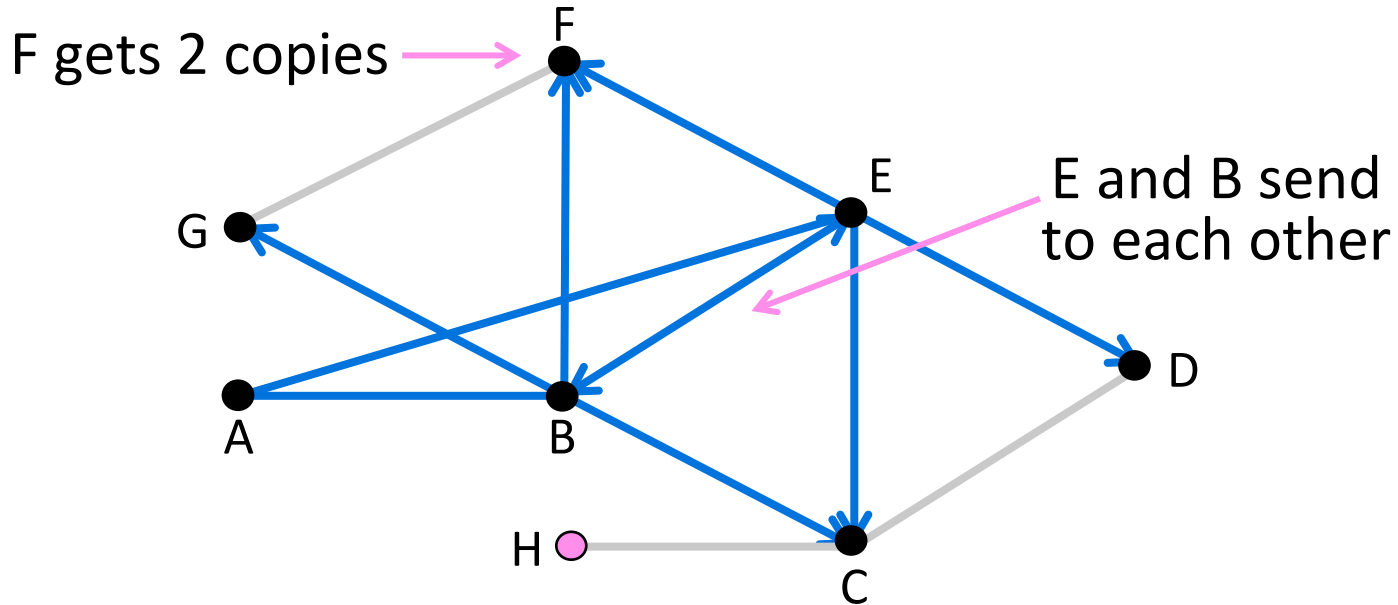
Flooding (2)

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



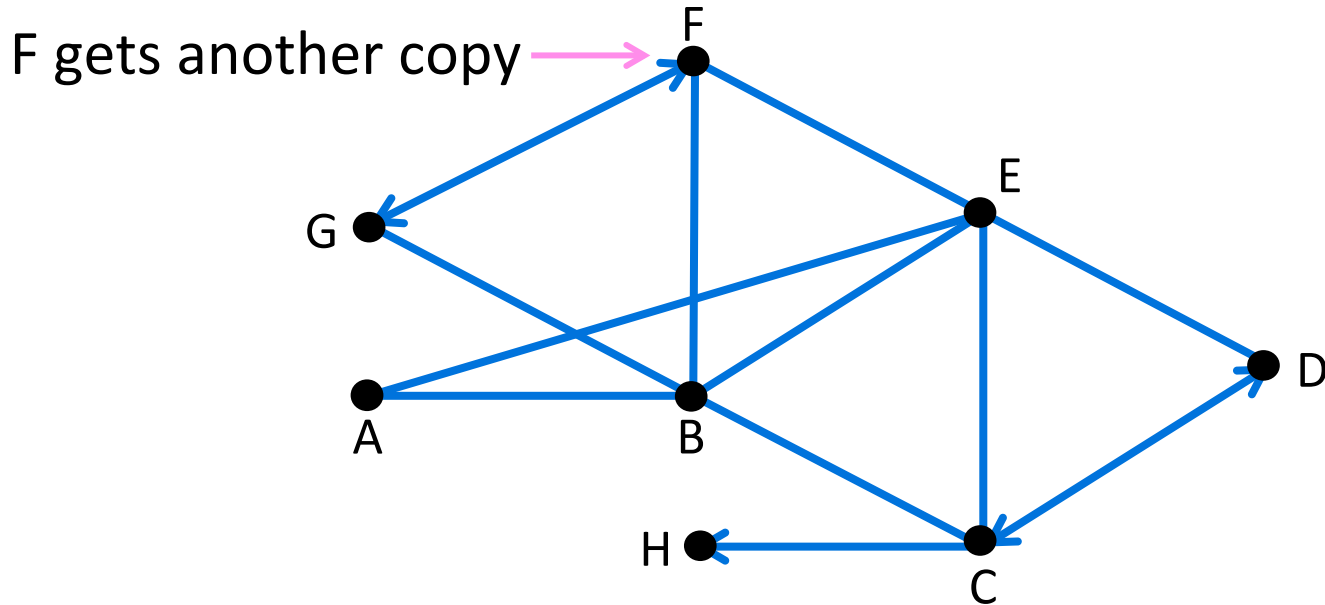
Flooding (3)

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



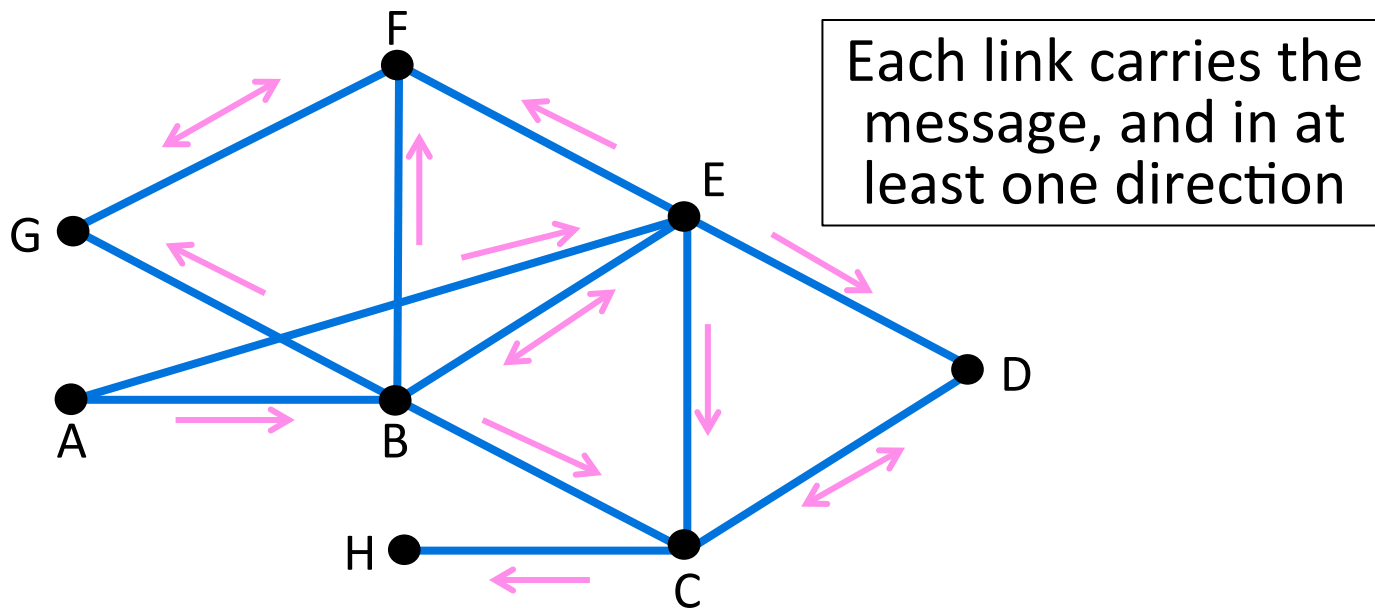
Flooding (4)

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



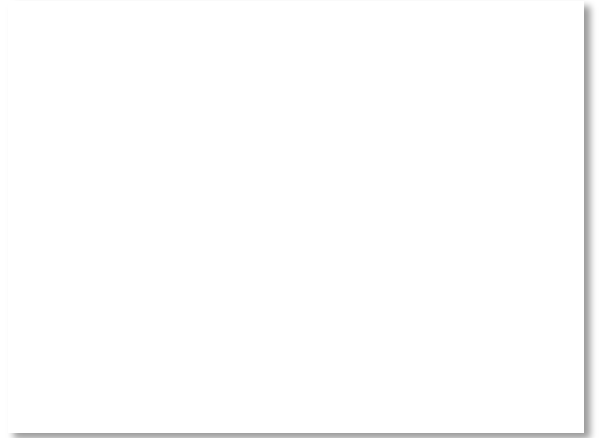
Flooding (5)

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



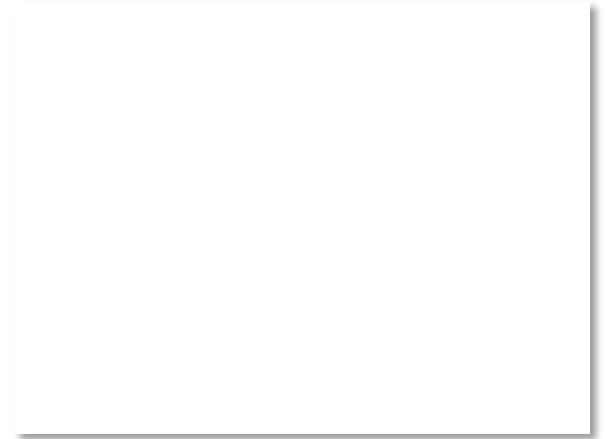
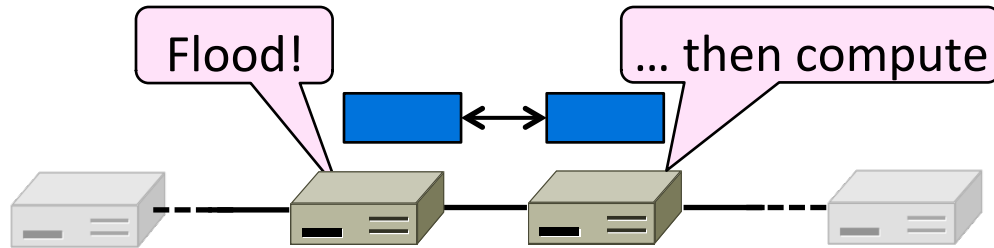
Flooding Details

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



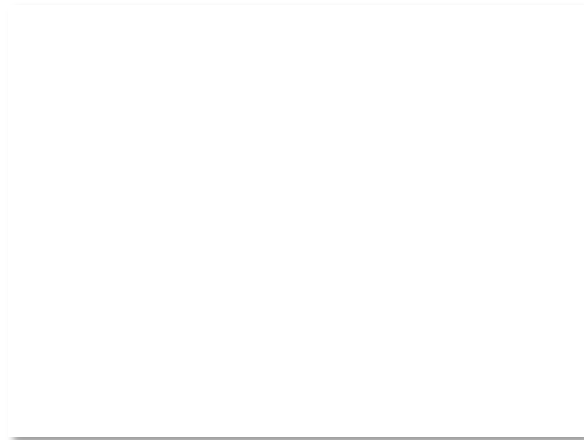
Topic

- How to compute shortest paths in a distributed network
 - The Link-State (LS) approach



Link-State Routing

- One of two approaches to routing
 - Trades more computation than distance vector for better dynamics
- Widely used in practice
 - Used in Internet/ARPANET from 1979
 - Modern networks use OSPF and IS-IS



Link-State Setting

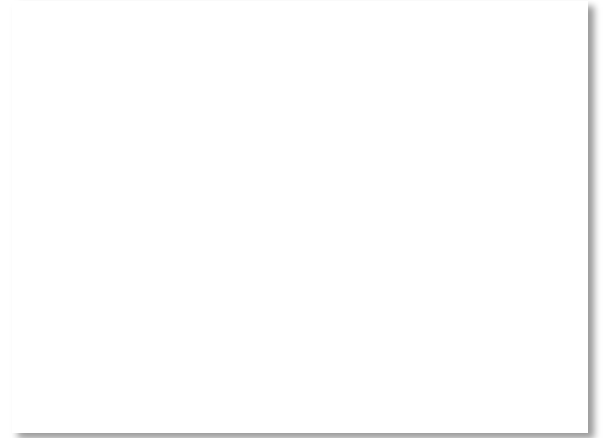
Nodes compute their forwarding table in the same distributed setting as for distance vector:

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 (or equivalent)

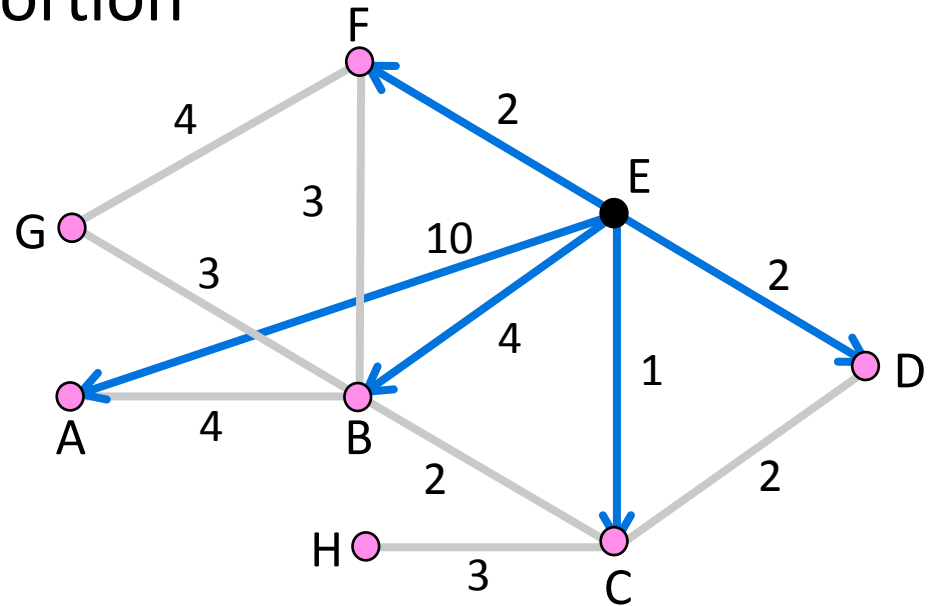


Phase 1: Topology Dissemination

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

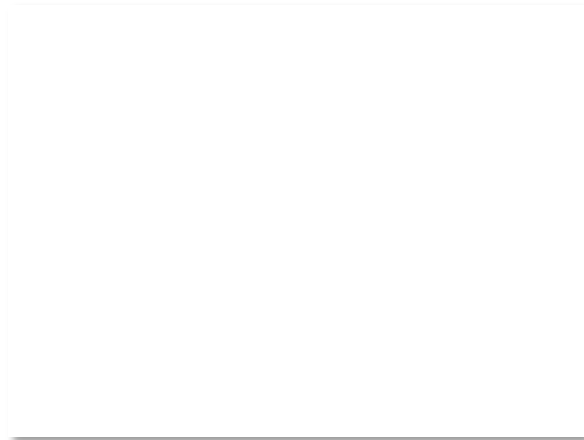
Node E's LSP
flooded to A, B,
C, D, and F

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



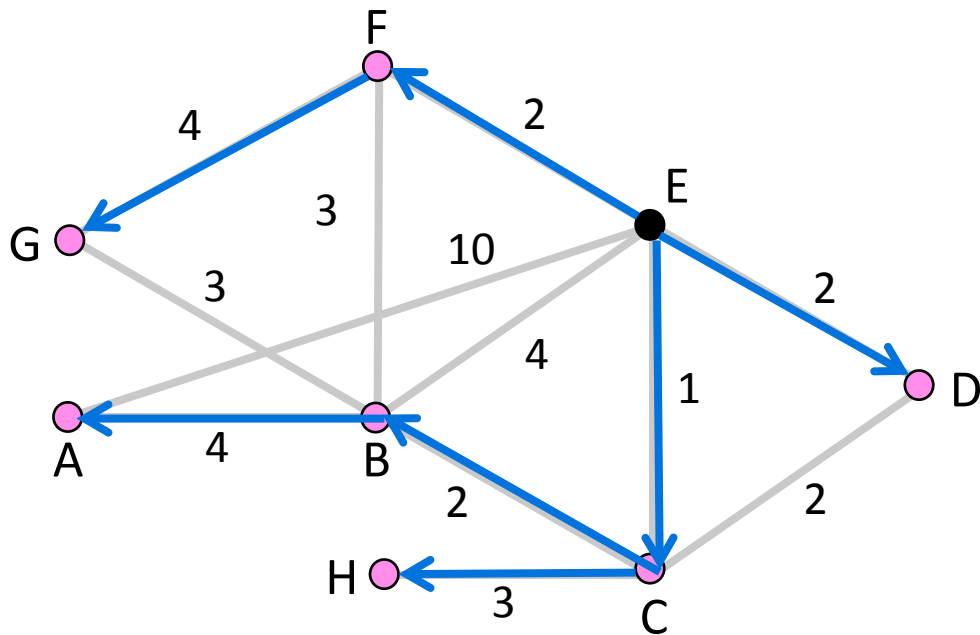
Phase 2: Route Computation

- 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'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

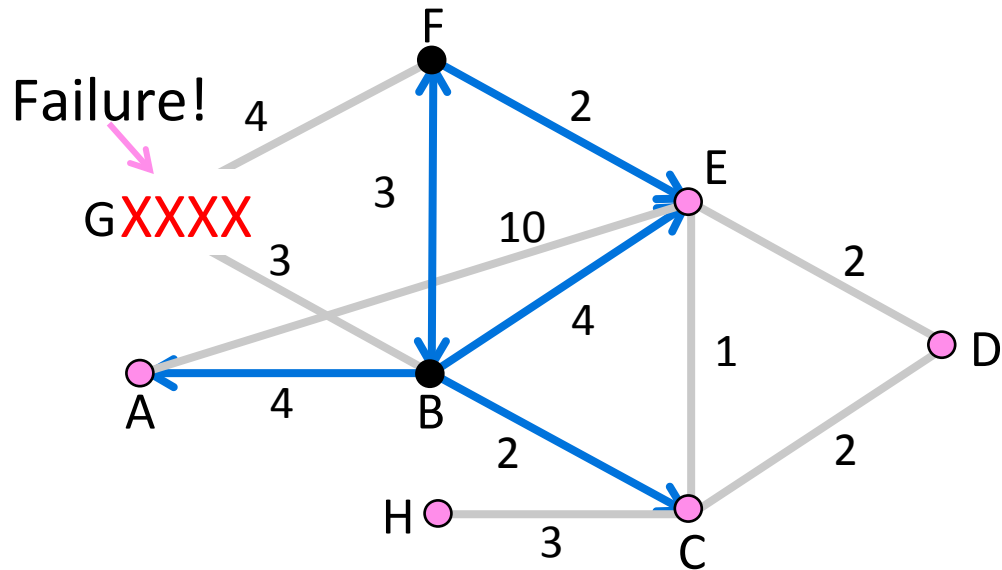
- On change, flood updated LSPs, and re-compute routes
 - E.g., nodes adjacent to failed link or node initiate

B's LSP

| | Seq. # |
|---|----------|
| A | 4 |
| C | 2 |
| E | 4 |
| F | 3 |
| G | ∞ |

F's LSP

| | Seq. # |
|---|----------|
| B | 3 |
| E | 2 |
| G | ∞ |



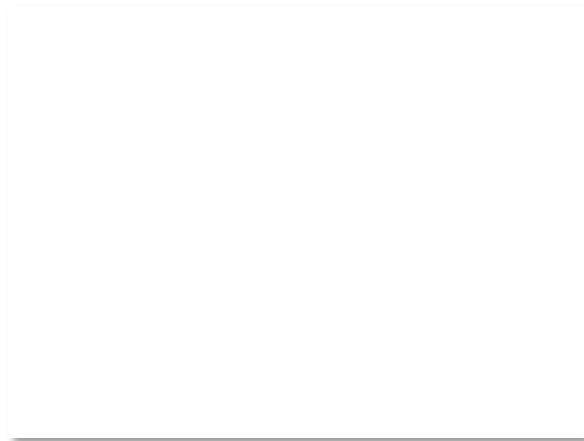
Handling Changes (2)

- Link failure
 - Both nodes notice, send updated LSPs
 - Link is removed from topology
- Node failure
 - All neighbors notice a link has failed
 - Failed node can't update its own LSP
 - But it is OK: all links to node removed



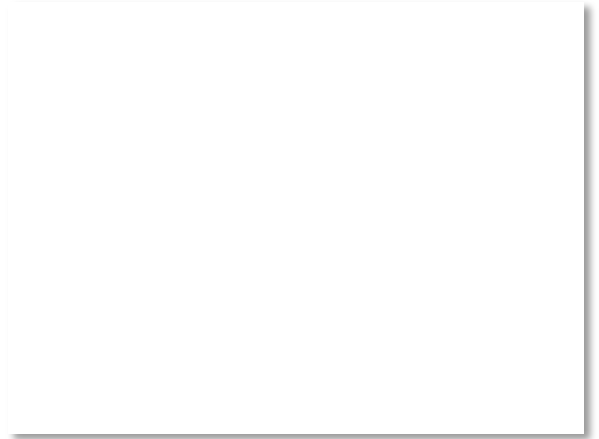
Handling Changes (3)

- 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 (as usual!)



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 |