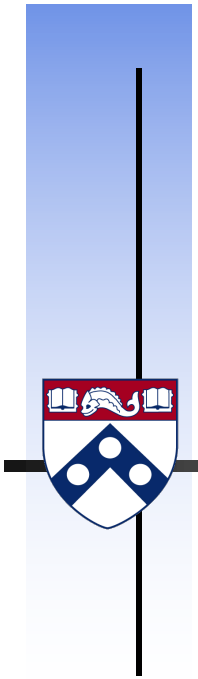


# CIS 553: Networked Systems

Interdomain Routing

February 24, 2021

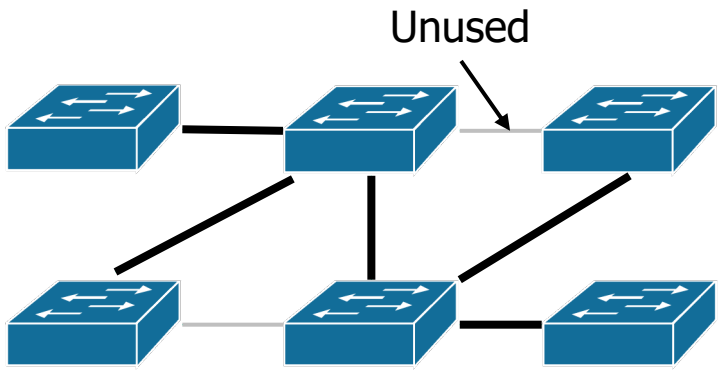
# Agenda

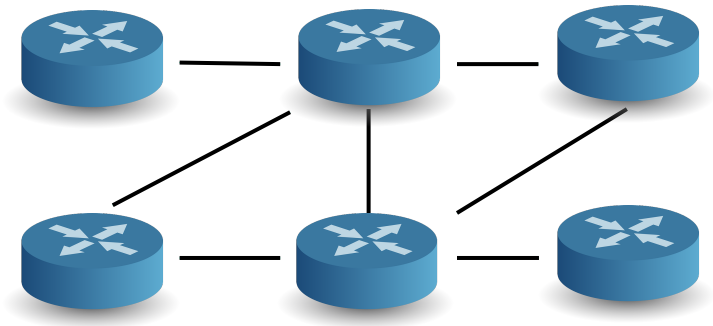
- Discovery ✓
  - DNS ✓
  - DHCP ✓
- Intradomain Routing 

  - Distance Vector
  - Link State
- Interdomain Routing

# Improving on the Spanning Tree

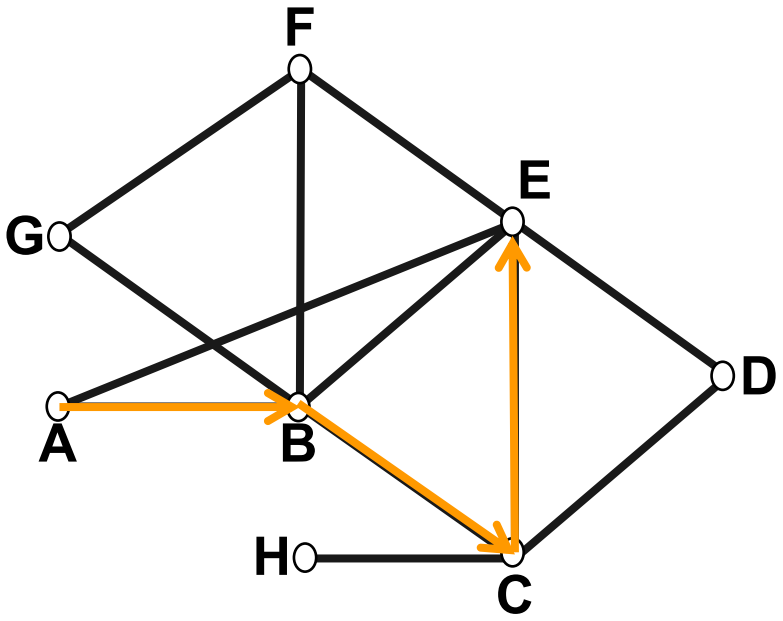
- Spanning tree provides basic connectivity
  - Wastes capacity
  - No guarantee of optimality
- Routing implies intelligence
  - Use all links to find "best" paths



# Routing

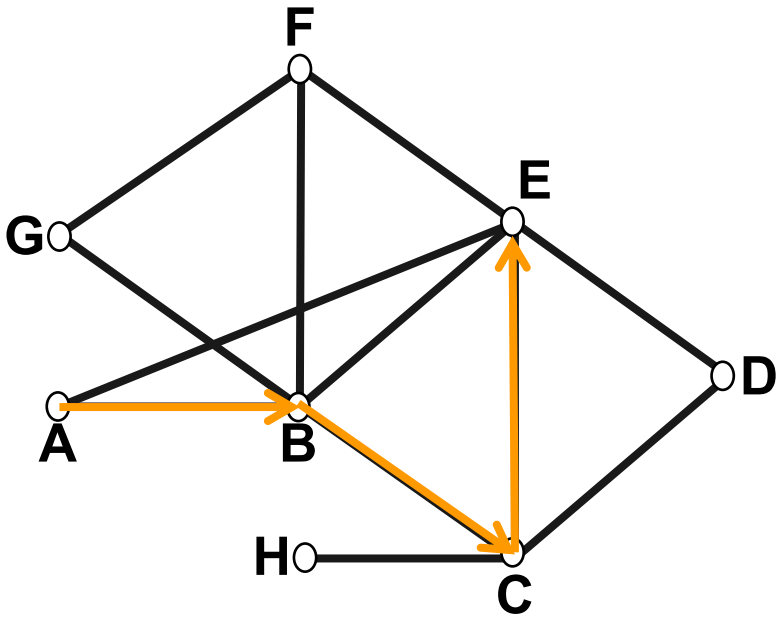
- Network modeled as a graph
  - Routers → nodes
  - Link →edges
- Goal: determine the "best" path through the network from source to destination




What does "best" mean?

# What are "best" paths anyhow?

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



# Shortest Paths

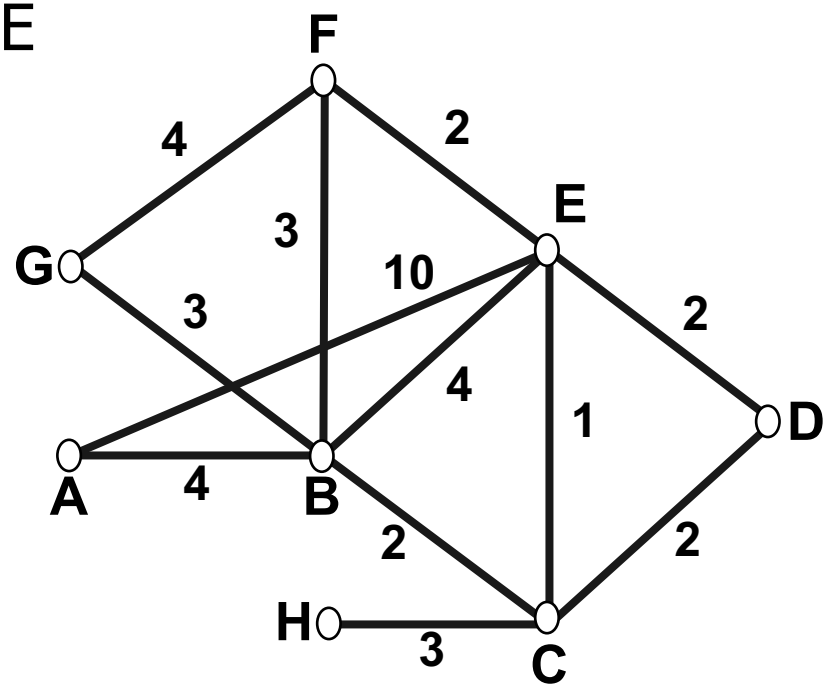
We'll approximate "best" by a cost function that captures the factors

- 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)

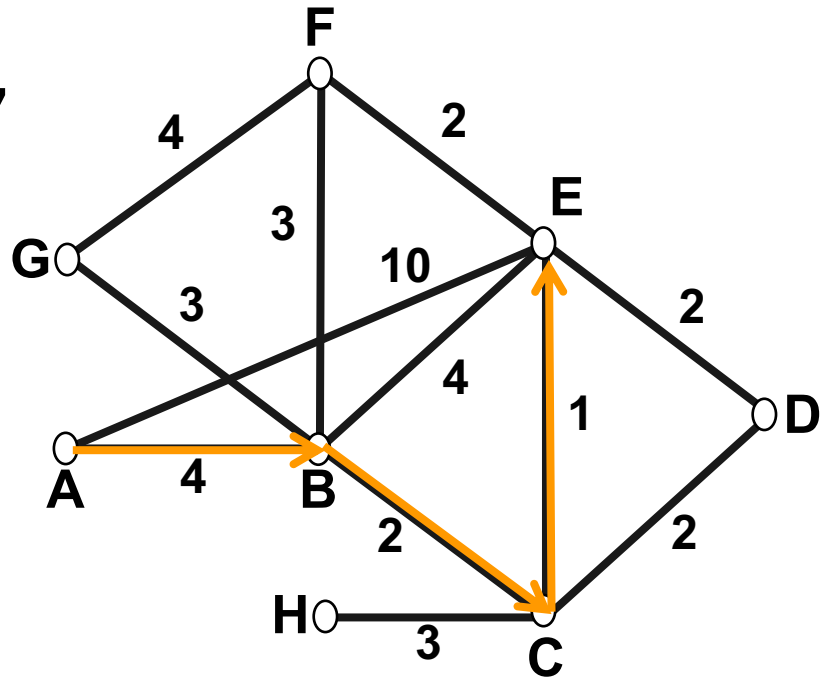
- Find the shortest path A → E
- All links are bidirectional, with equal costs in each direction
  - Can extend model to unequal costs if needed



# Shortest Paths (3)

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

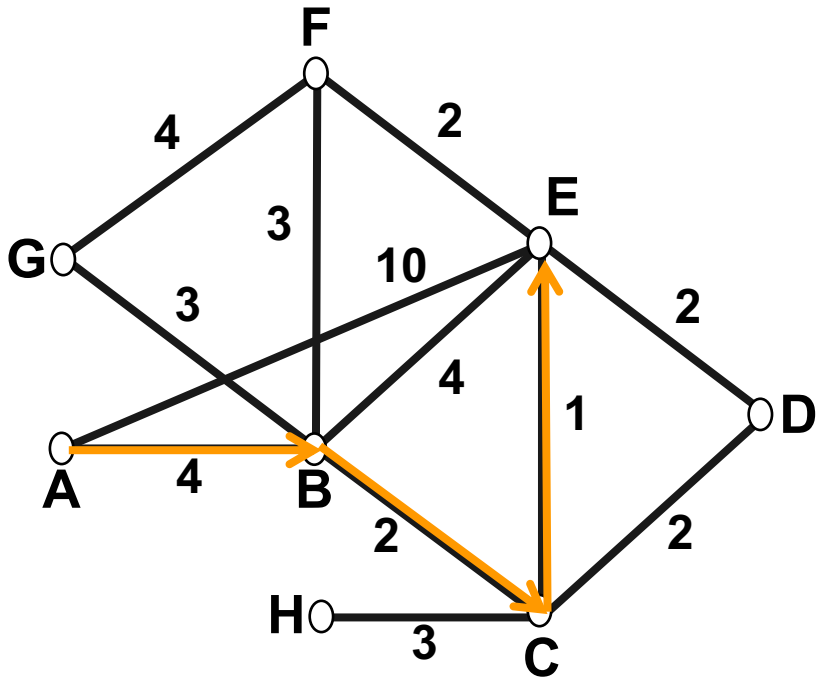
- This is less than:
  - dist(ABE) = 8
  - dist(ABFE) = 9
  - dist(AE) = 10
  - dist(ABCDE) = 10



# Shortest Paths (4)

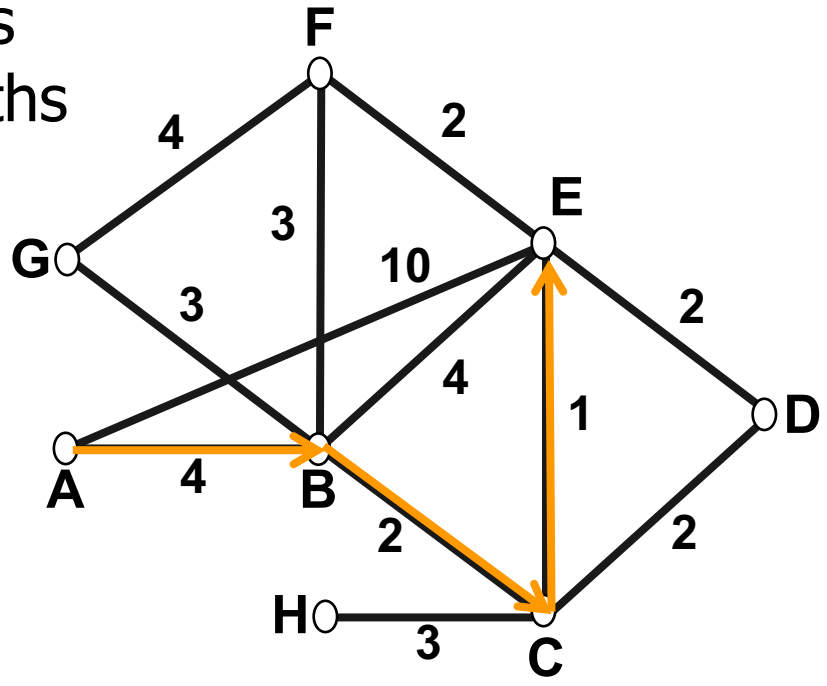
- Optimality property:
  - Subpaths of shortest paths are also shortest paths
- ABCE is a shortest path

  →So are ABC, AB, BCE, BC, CE



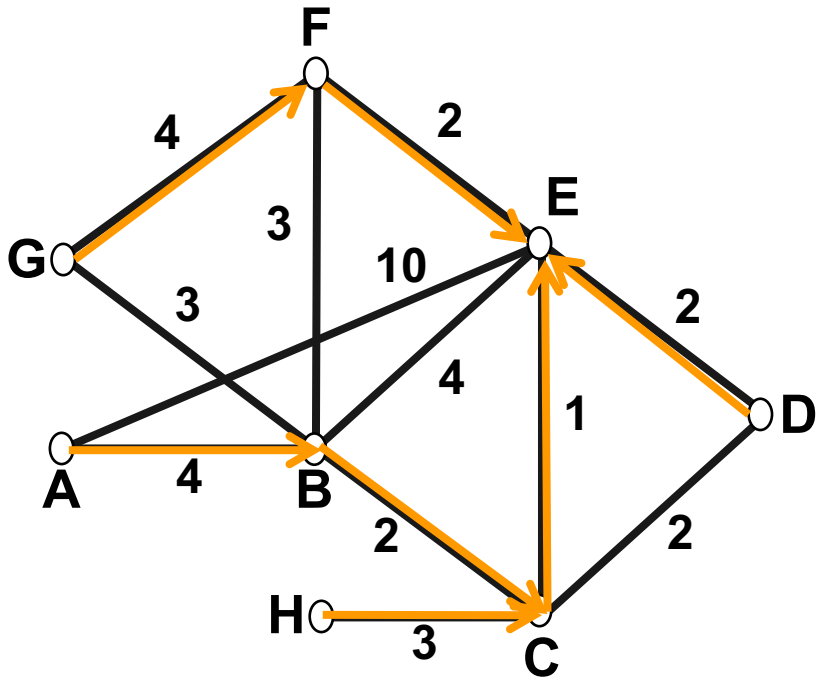
# Sink Trees

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



# Agenda

- Intradomain Routing 
  - Distance Vector 
    
  - Link State
- Interdomain Routing

# Distance Vector

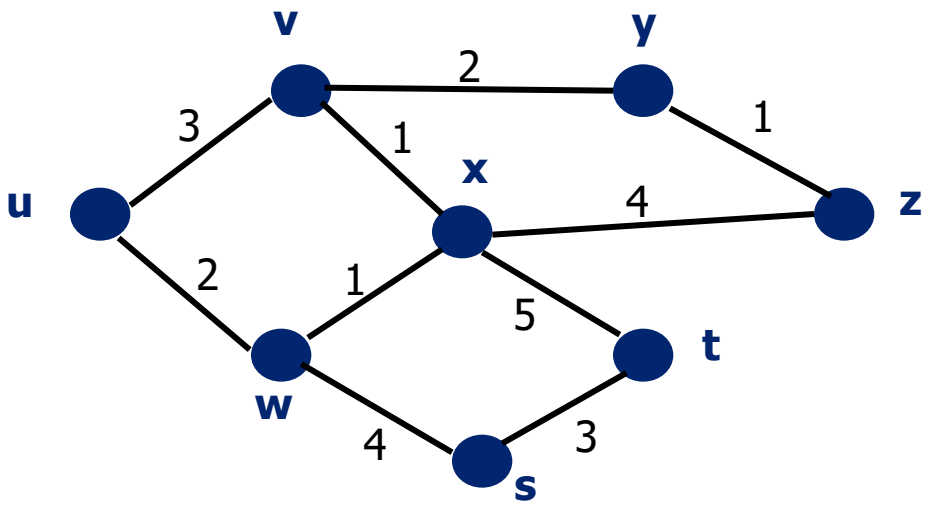
key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV

- under minor, natural conditions, routing will eventually converge

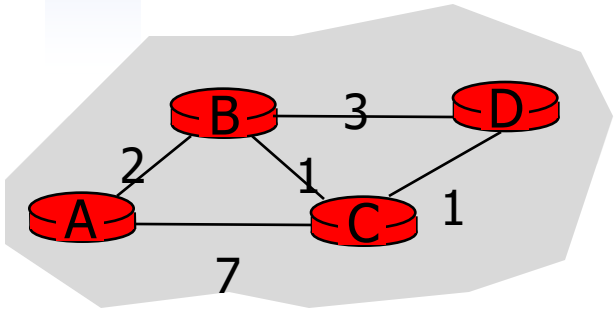
# Distance Vector: Bellman-Ford

- Define distances at each node x
  - $d_x(y)$ = cost of least-cost path from x to y
- Update distances based on neighbors
  - $d_x(y) = \min \{c(x,v) + d_v(y)\}$ over all neighbors v




$$d_u(z) = \min\{\, c(u,v) + d_v(z),\; c(u,w) + d_w(z)\}$$

# Example: Distance Vector Algorithm




Node A

| Dest. | Cost | NextHop |
|---|---|---|
| B | ∞ | - |
| C | ∞ | - |
| D | ∞ | - |

Node B

| Dest. | Cost | NextHop |
|---|---|---|
| A | ∞ | - |
| C | ∞ | - |
| D | ∞ | - |

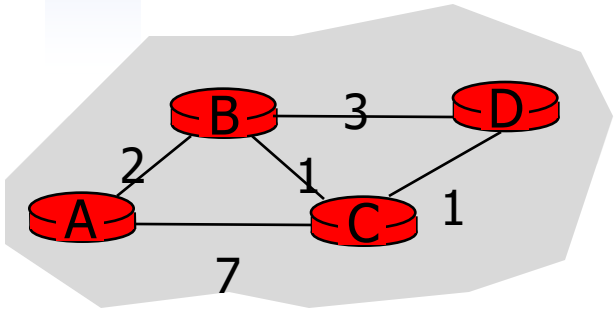
Node C

| Dest. | Cost | NextHop |
|---|---|---|
| A | ∞ | - |
| B | ∞ | - |
| D | ∞ | - |

Node D

| Dest. | Cost | NextHop |
|---|---|---|
| A | ∞ | - |
| B | ∞ | - |
| C | ∞ | - |

# Example: 0th Iteration




Node A

| Dest. | Cost | NextHop |
|---|---|---|
| B | 2 | B |
| C | 7 | C |
| D | ∞ | - |

Node B

| Dest. | Cost | NextHop |
|---|---|---|
| A | 2 | A |
| C | 1 | C |
| D | 3 | D |

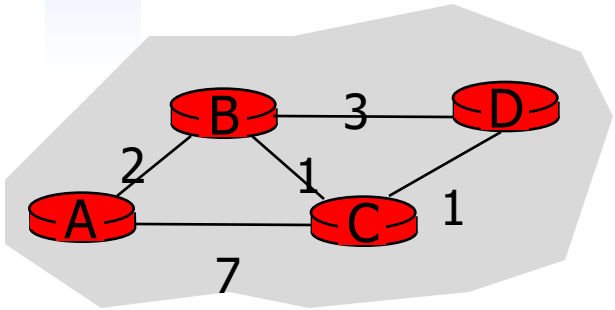
Node C

| Dest. | Cost | NextHop |
|---|---|---|
| A | 7 | A |
| B | 1 | B |
| D | 1 | D |

Node D

| Dest. | Cost | NextHop |
|---|---|---|
| A | ∞ | - |
| B | 3 | B |
| C | 1 | C |

# Example: 1st Iteration (A's Updates)




Node A

| Dest. | Cost | NextHop |
|---|---|---|
| B | 2 | B |
| C | 3 | B |
| D | 5 | B |

Node B

| Dest. | Cost | NextHop |
|---|---|---|
| A | 2 | A |
| C | 1 | C |
| D | 3 | D |

D(A,D) = D(A,B) + D(B,D) = 2 + 3 = 5

D(A,C) = D(A,B) + D(B,C) = 2 + 1 = 3

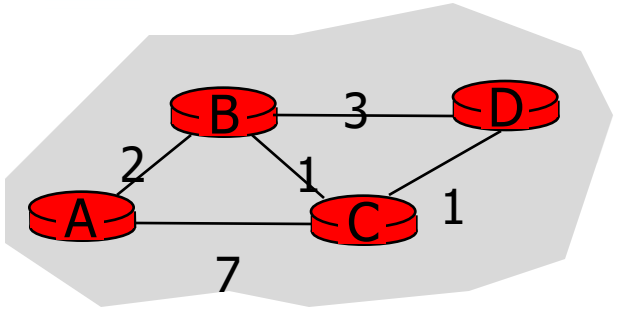
Node C

| Dest. | Cost | NextHop |
|---|---|---|
| A | 7 | A |
| B | 1 | B |
| D | 1 | D |

Node D

| Dest. | Cost | NextHop |
|---|---|---|
| A | ∞ | - |
| B | 3 | B |
| C | 1 | C |

# Example: End of 1st Iteration




Node A

| Dest. | Cost | NextHop |
|---|---|---|
| B | 2 | B |
| C | 3 | B |
| D | 5 | B |

Node B

| Dest. | Cost | NextHop |
|---|---|---|
| A | 2 | A |
| C | 1 | C |
| D | 2 | C |

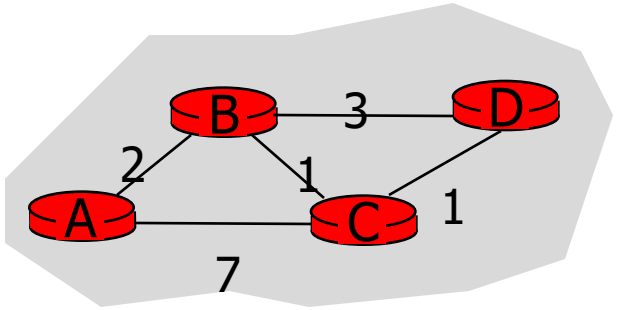
Node C

| Dest. | Cost | NextHop |
|---|---|---|
| A | 3 | B |
| B | 1 | B |
| D | 1 | D |

Node D

| Dest. | Cost | NextHop |
|---|---|---|
| A | 5 | B |
| B | 2 | C |
| C | 1 | C |

# Example: End of 2nd Iteration




Node A

| Dest. | Cost | NextHop |
|---|---|---|
| B | 2 | B |
| C | 3 | B |
| D | 4 | B |

Node B

| Dest. | Cost | NextHop |
|---|---|---|
| A | 2 | A |
| C | 1 | C |
| D | 2 | C |

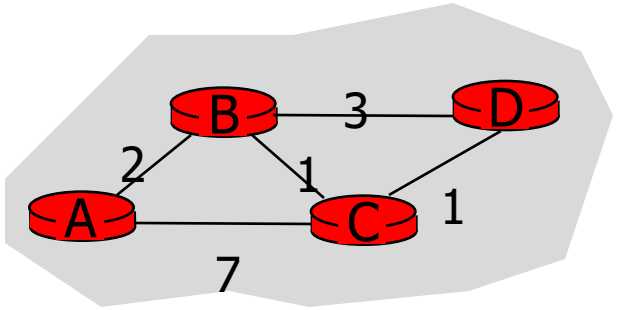
Node C

| Dest. | Cost | NextHop |
|---|---|---|
| A | 3 | B |
| B | 1 | B |
| D | 1 | D |

Node D

| Dest. | Cost | NextHop |
|---|---|---|
| A | 4 | C |
| B | 2 | C |
| C | 1 | C |

# Example: End of 3rd Iteration




Node A

| Dest. | Cost | NextHop |
|---|---|---|
| B | 2 | B |
| C | 3 | B |
| D | 4 | B |

Node B

| Dest. | Cost | NextHop |
|---|---|---|
| A | 2 | A |
| C | 1 | C |
| D | 2 | C |

Node C

| Dest. | Cost | NextHop |
|---|---|---|
| A | 3 | B |
| B | 1 | B |
| D | 1 | D |

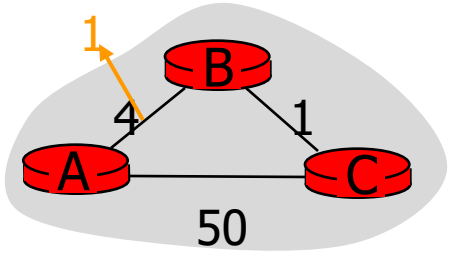
Node D

| Dest. | Cost | NextHop |
|---|---|---|
| A | 4 | C |
| B | 2 | C |
| C | 1 | C |

Nothing changes → algorithm has converged

# Distance Vector: Link Cost Changes

- Link cost changes:
  - node detects local link cost change
  - updates routing info, recalculates distance vector
  - if DV changes, notify neighbors




Node B

| D | C | N |
|---|---|---|
| A | 4 | A |
| C | 1 | B |

Node C

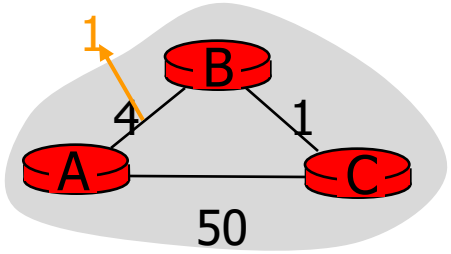
| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

time

Link cost changes here

# Distance Vector: Link Cost Changes

- Link cost changes:
  - node detects local link cost change
  - updates routing info, recalculates distance vector
  - if DV changes, notify neighbors




Node B

| D | C | N |
|---|---|---|
| A | 4 | A |
| C | 1 | B |

Node C

| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 1 | A |
| C | 1 | B |
| D | C | N |
| A | 5 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 1 | A |
| C | 1 | B |
| D | C | N |
| A | 2 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 1 | A |
| C | 1 | B |
| D | C | N |
| A | 2 | B |
| B | 1 | B |

time

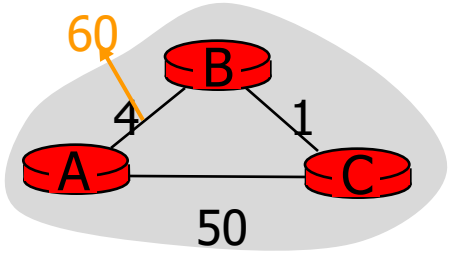
Link cost changes here

Algorithm terminates

"good news travels fast"

# Distance Vector: Count to Infinity Problem

- Link cost changes:
  - node detects local link cost change
  - updates routing info, recalculates distance vector
  - if DV changes, notify neighbors




Node B

| D | C | N |
|---|---|---|
| A | 4 | A |
| C | 1 | B |

Node C

| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 6 | C |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 6 | C |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 7 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 8 | C |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 7 | B |
| B | 1 | B |

"bad news travels slowly"

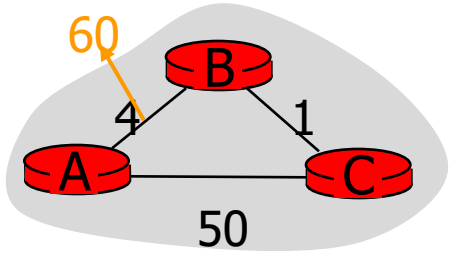
...

time

Link cost changes here; (A,4,A) at node B is invalidated.
Node C advertises D(C,A)=5 to node B. Thus D(B, A) becomes 6!

# Distance Vector: Poisoned Reverse

- If C routes through B to get to A:
  - C tells B its (C's) distance to A is infinite (so B won't route to A via C)
  - Will this solve count to infinity problem?




Node B

| D | C | N |
|---|---|---|
| A | 4 | A |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 60 | A |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 60 | A |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 51 | C |
| C | 1 | B |

| D | C | N |
|---|---|---|
| A | 51 | C |
| C | 1 | B |

Node C

| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 50 | A |
| B | 1 | B |

| D | C | N |
|---|---|---|
| A | 50 | A |
| B | 1 | B |

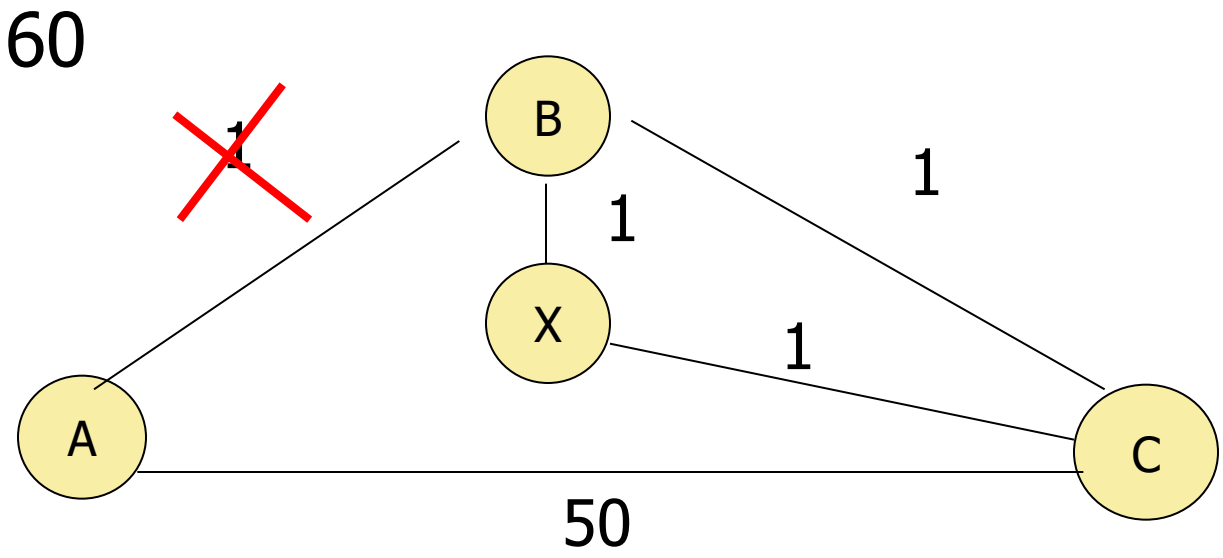
| D | C | N |
|---|---|---|
| A | 50 | A |
| B | 1 | B |

time

Link cost changes here;B updates D(B, A) = 60 as C has advertised D(C, A) = ∞ instead of D(C,A)=5.
At node C, (A,5,B) is replaced by (A,60,B). Node C switches to use (A,50,A).

Algorithm terminates

# Does Poison Reverse Always Work?




Node B

| D | C | N |
|---|---|---|
| A | 4 | A |
| C | 1 | B |

Node C

| D | C | N |
|---|---|---|
| A | 5 | B |
| B | 1 | B |

# Example: Routing Information Protocol

- Earliest IP routing protocol (1982 BSD)
  - Version 1: RFC 1058
  - Version 2: RFC 2453
- Features
  - Edges have unit cost
  - "Infinity" = 16
- Sending Updates
  - Router listens for updates on UDP port 520
  - Message can contain up to 25 table entries

# Agenda

- Intradomain Routing 
  - Distance Vector 
  - Link State 
    
- Interdomain Routing

# Link-State Routing

Key idea: distribute a network map

- Each node performs shortest path (SPF) computation between itself and all other nodes
- Initialization step
  - Add costs of immediate neighbors, D(v), else infinite
  - Flood costs c(u,v) to neighbors, N
- For some D(w) that is not in N
  - D(v) = min( c(u,w) + D(w), D(v) )

# Dijsktra's Algorithm

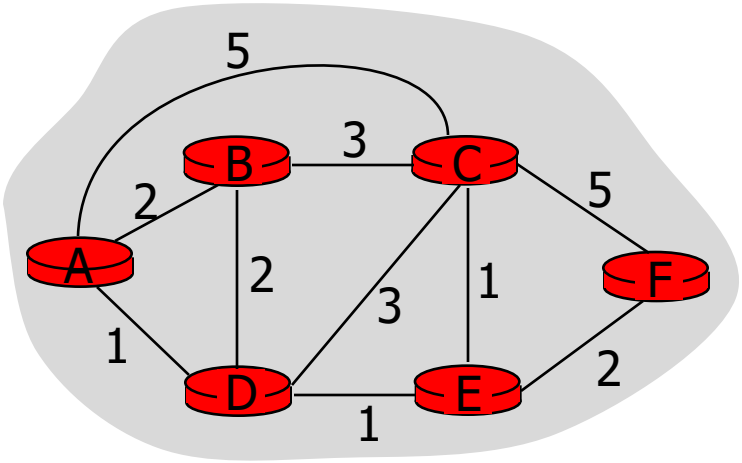
```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞;
7
8  Loop
9      find w not in S such that D(w) is a minimum;
10     add w to S;
11     update D(v) for all v adjacent to w and not in S:
12       D(v) = min( D(v), D(w) + c(w,v) );
           // new cost to v is either old cost to v or known
           // shortest path cost to w plus cost from w to v
13 until all nodes in S;
```

Notations

- c(i,j): link cost from node *i* to *j*; cost infinite if not direct neighbors
- D(v): current value of cost of path from source to destination *v*
- p(v): predecessor node along path from source to *v*, that is next to *v*
- S: set of nodes whose least cost path definitively known

# Example: Dijkstra's Algorithm

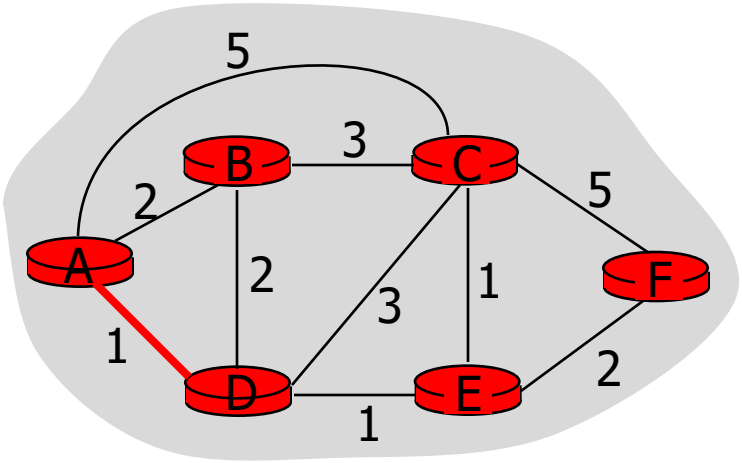
| Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞ ;
…
```

# Example: Dijkstra's Algorithm

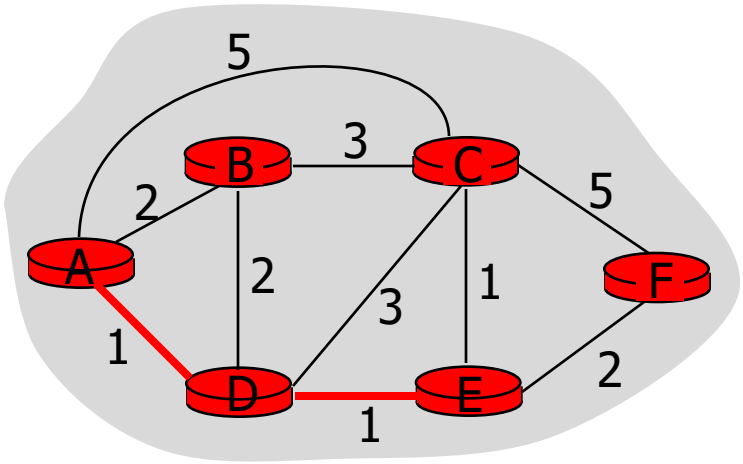
| Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | ∞ |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a
      minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12      D(v) = min( D(v), D(w) + c(w,v) );
13    until all nodes in S;
```

# Example: Dijkstra's Algorithm

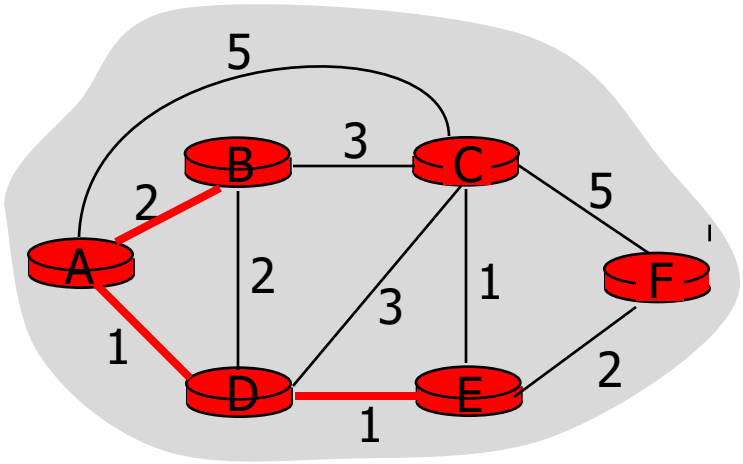
| Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | ∞ |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a
      minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12      D(v) = min( D(v), D(w) + c(w,v) );
13    until all nodes in S;
```

# Example: Dijkstra's Algorithm

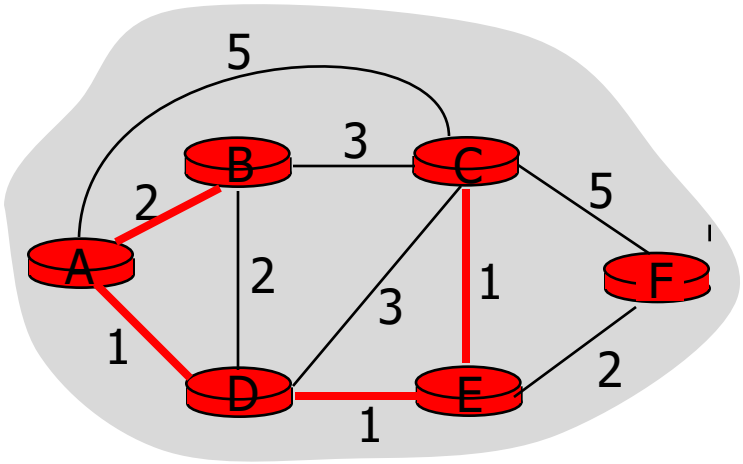
| Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | ∞ |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a
      minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12      D(v) = min( D(v), D(w) + c(w,v) );
13    until all nodes in S;
```

# Example: Dijkstra's Algorithm

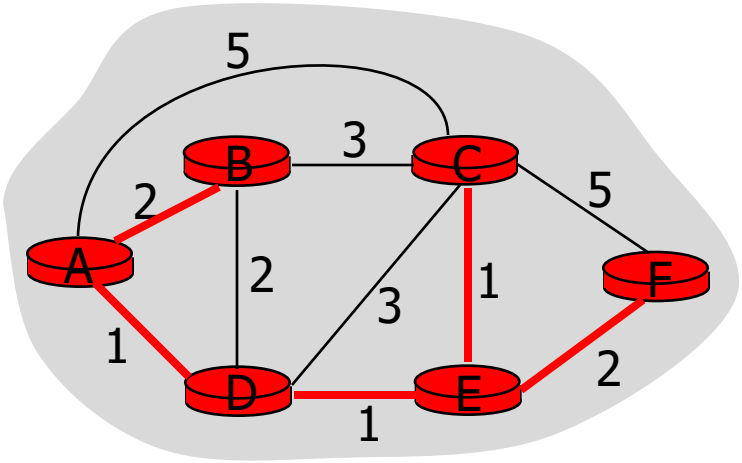
| Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | ∞ |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | ADEBC | | | | | |
| 5 | | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a
      minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12      D(v) = min( D(v), D(w) + c(w,v) );
13    until all nodes in S;
```

# Example: Dijkstra's Algorithm

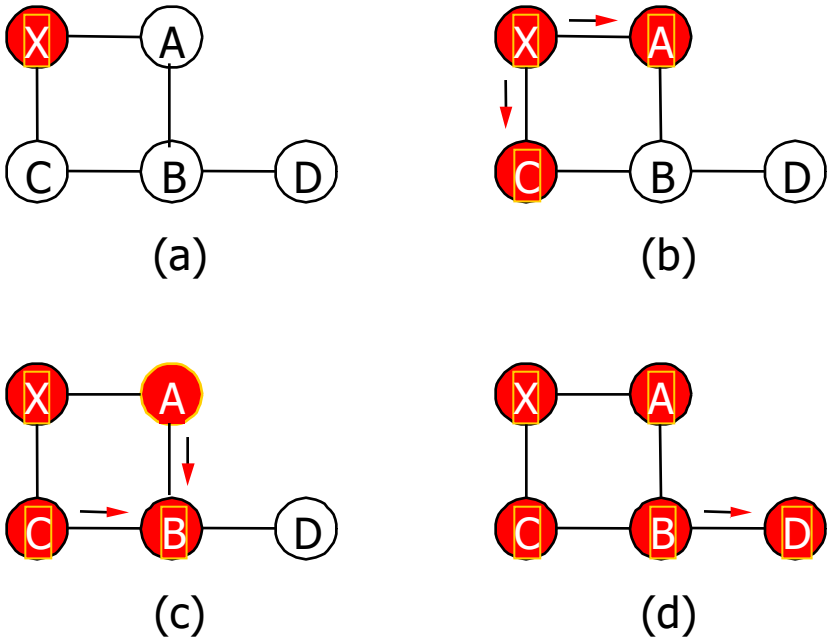
| Step | start S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | ∞ |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | ADEBC | | | | | |
| 5 | ADEBCF | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a
      minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12      D(v) = min( D(v), D(w) + c(w,v) );
13    until all nodes in S;
```

# Flooding the Link State

- Node sends link-state information out its links
- Similar to broadcasting, but not quite! Why?



# Flooding the Link State

- Challenges
  - Loops
  - Packet loss
  - Out-of-order arrival
- Solutions
  - Sequence numbers
  - Acknowledgments and retransmissions
  - Time-to-live for each packet