

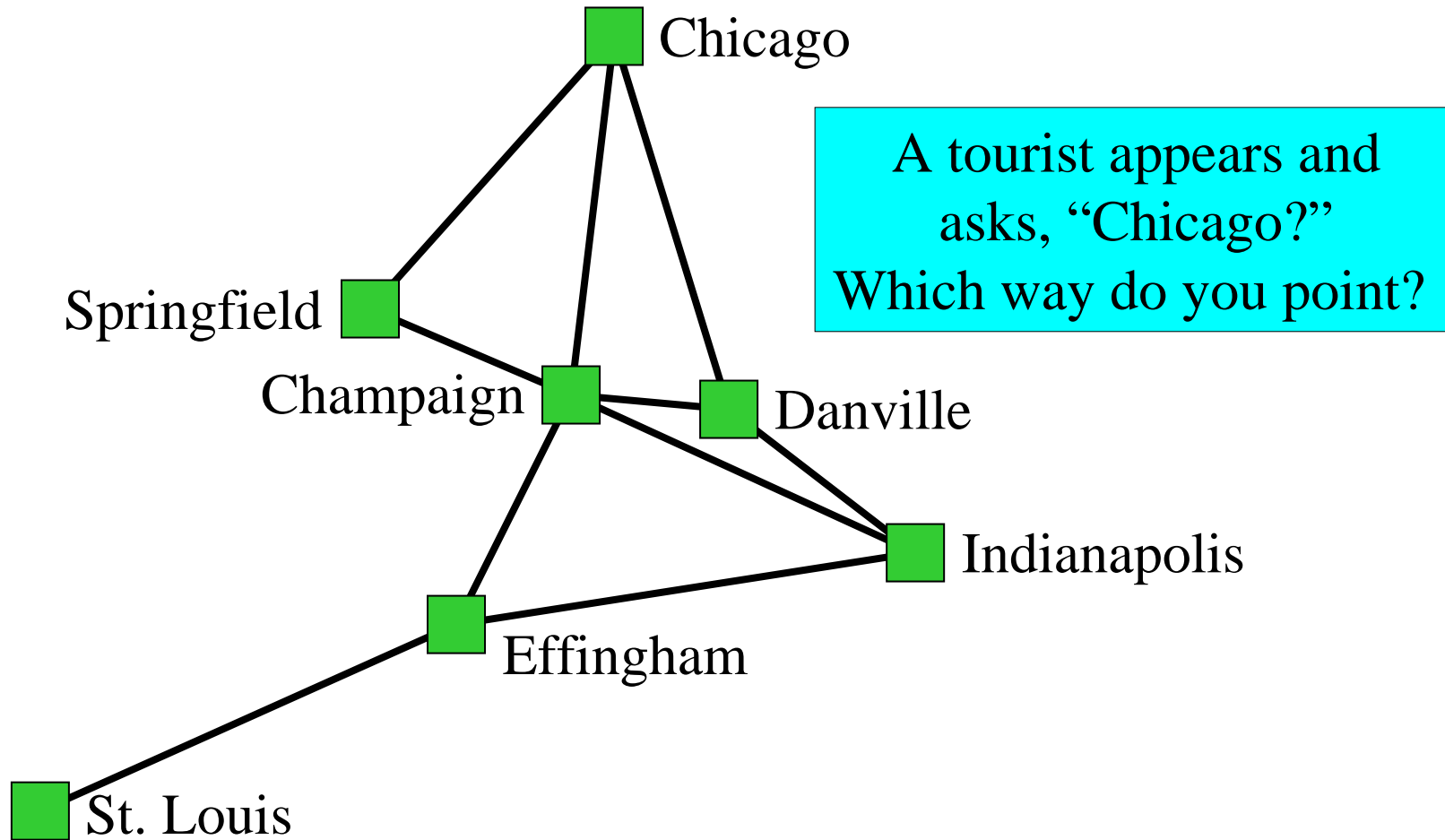
Lecture 8: Routing

CS/ECE 438: Communication Networks

Prof. Matthew Caesar

March 3, 2010

Routing



Routing

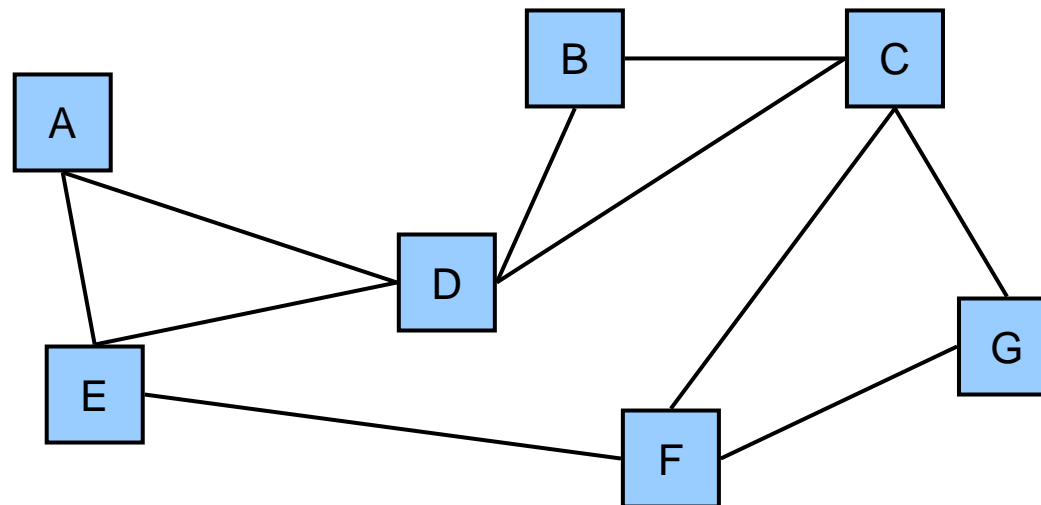
- Definition
 - The task of constructing and maintaining forwarding information (in hosts or routers)
- Goals
 - Capture the notion of “best” routes
 - Propagate changes effectively
 - Require limited information exchange
- Conceptually
 - A network can be represented as a graph where each host/router is a node and each physical connection is a link

Routing

- Factors
 - Network topology can change
 - Traffic conditions can change
- Design elements
 - Performance criteria
 - Decision time and place
 - Information source
- Goals
 - Correctness
 - Simplicity
 - Robustness
 - Fairness
 - High throughput
 - Low end-to-end latency

Routing: Ideal Approach

- Maintain information about each link
- Calculate fastest path between each directed pair



For each direction, maintain:

- Bandwidth
- Latency
- Queueing delay

Routing: Ideal Approach

- Problems
 - Unbounded amount of information
 - Queueing delay can change rapidly
 - Graph connectivity can change rapidly
- Solution
 - Dynamic
 - Periodically recalculate routes
 - Distributed
 - No single point of failure
 - Reduced computation per node
 - Abstract Metric
 - “Distance” may combine many factors
 - Use heuristics

Routing Overview

- Algorithms
 - Static shortest path algorithms
 - Bellman-Ford
 - Based on local iterations
 - Dijkstra's algorithm
 - Build tree from source
 - Distributed, dynamic routing algorithms
 - Distance vector routing
 - Distributed Bellman-Ford
 - Link state routing
 - Implement Dijkstra's algorithm at each node

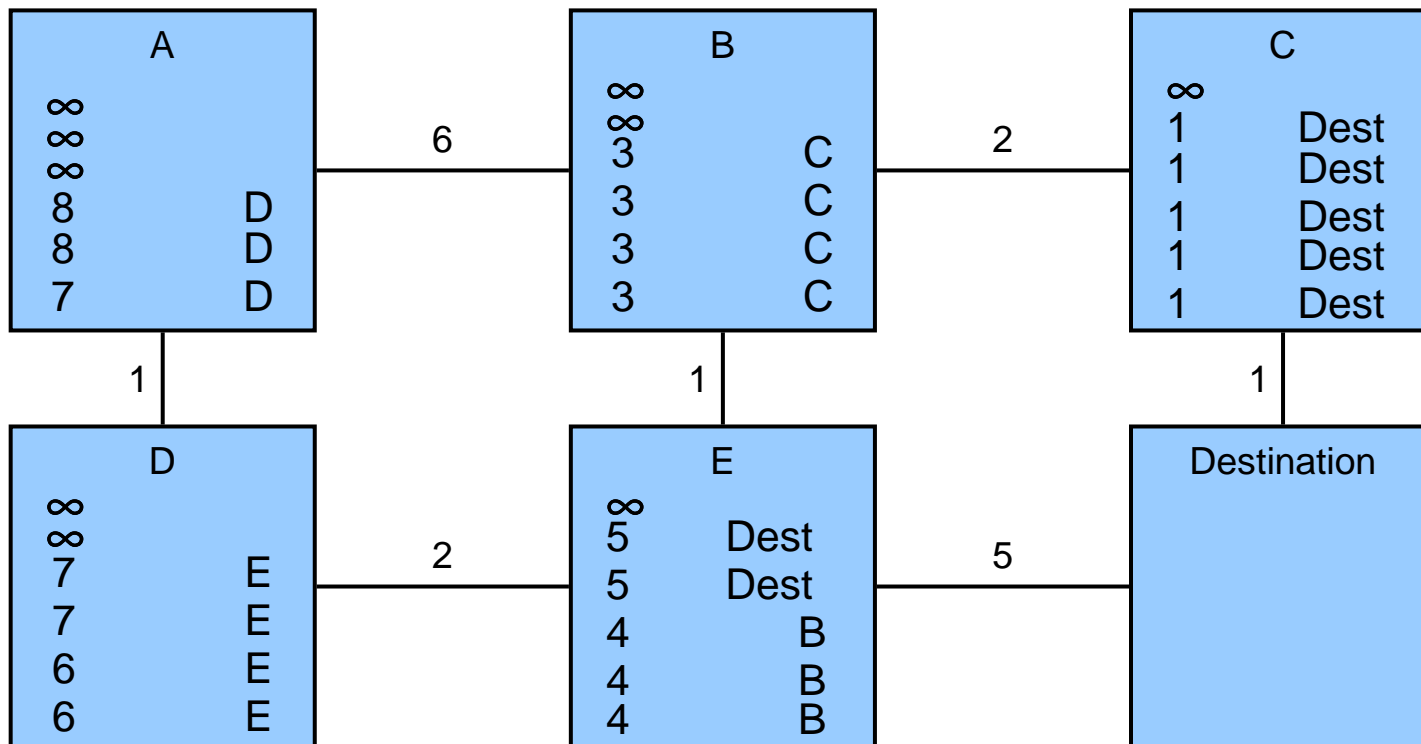
Bellman-Ford Algorithm

- Concept
 - Static centralized algorithm
- Given
 - Directed graph with edge costs and destination node
- Finds
 - Least cost path from each node to destination
- Multiple nodes
 - To find shortest paths for multiple destination nodes, run entire Bellman-Ford algorithm once per destination

Bellman-Ford Algorithm

- Based on repetition of iterations
 - For every node A and every neighbor B of A
 - Is the cost of the path (A → B → → → destination) smaller than the currently known cost from A to destination?
 - If YES
 - Make B the successor node for A
 - Update cost from A to destination
 - Can run iterations synchronously or all at once

Bellman-Ford Algorithm



Distance Vector Routing

- Distributed dynamic version of Bellman-Ford
- Each node maintains a table of
 - $\langle \text{destination}, \text{distance}, \text{successor} \rangle$
- Information acquisition
 - Assume nodes initially know cost to immediate neighbor
 - Nodes send $\langle \text{destination}, \text{distance} \rangle$ vectors to all immediate neighbors
 - Periodically – seconds, minutes
 - Whenever vector changes – triggered update

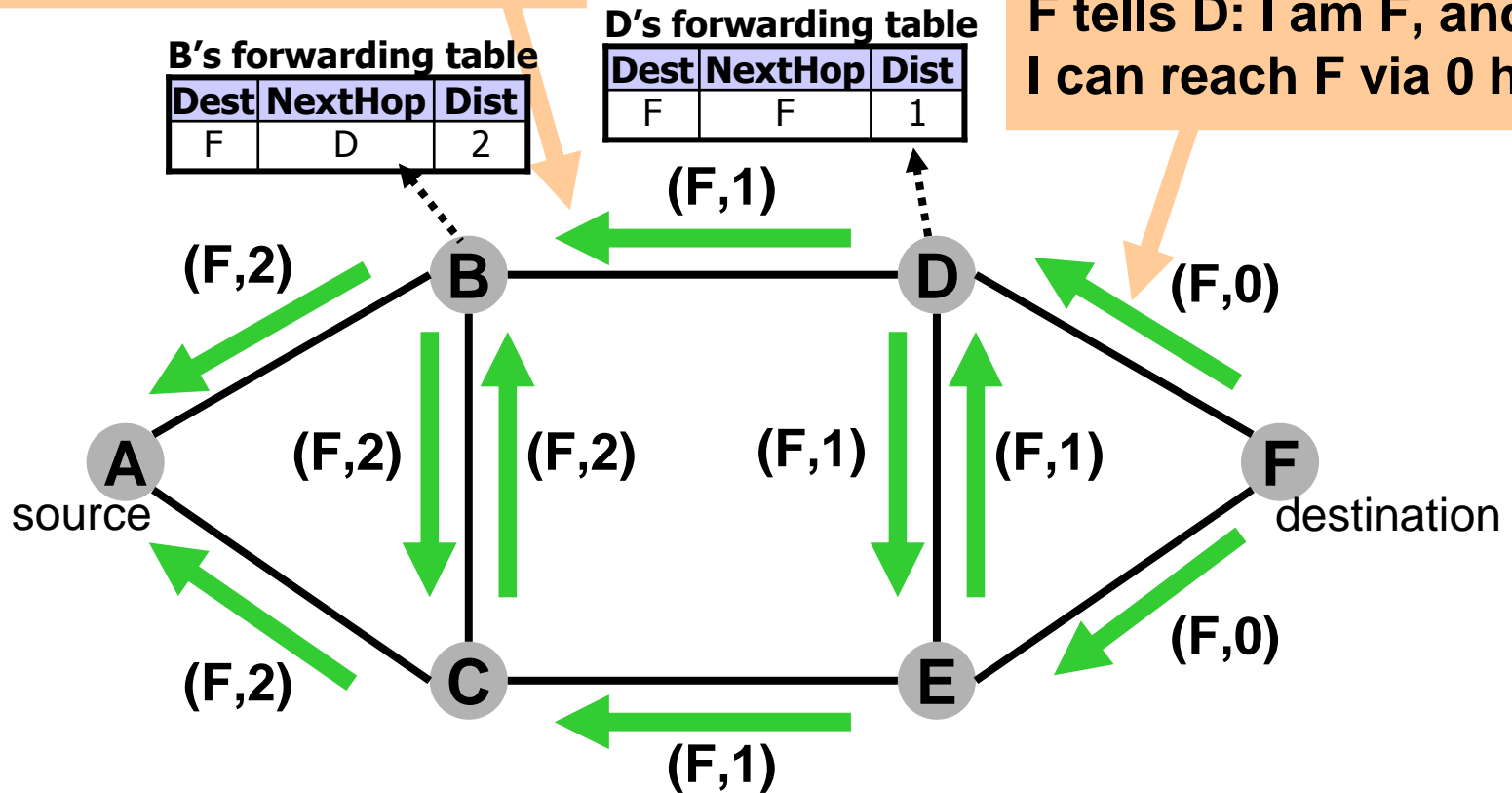
Distance Vector Routing

- When a route changes
 - Local failure detection
 - Control message not acknowledged
 - Timeout on periodic route update
 - Current route disappears
 - Newly advertised route is shorter than previous route
- Used in
 - Original ARPANET (until 1979)
 - Early Internet: Routing Information Protocol (RIP)
 - Early versions of DECnet and Novell IPX

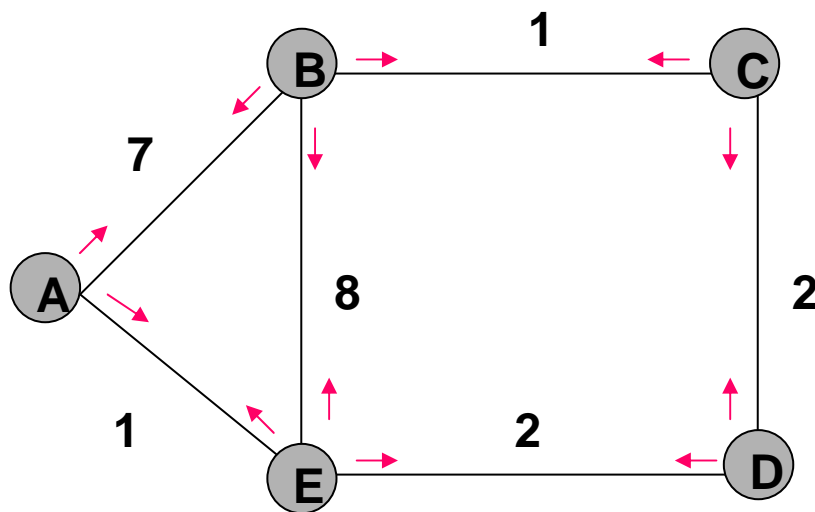
Distance vector: update propagation

D tells B: I am D, and I can reach F via 1 hop

F tells D: I am F, and I can reach F via 0 hops

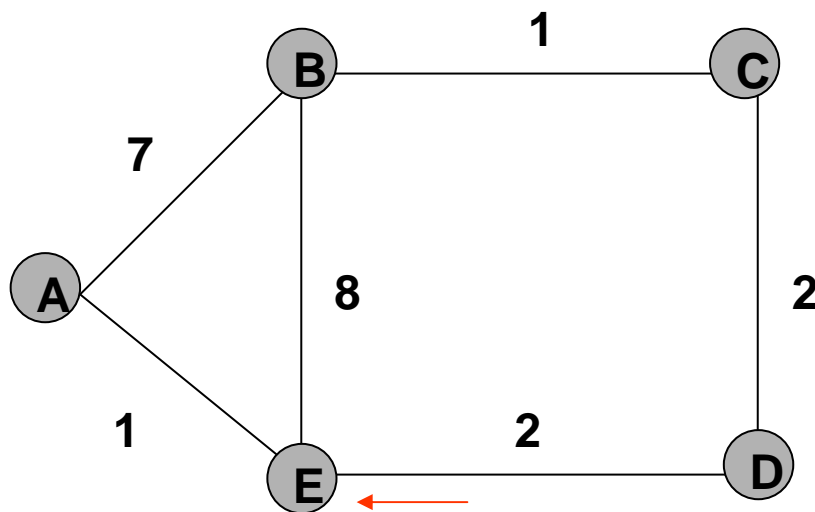


Example - Initial Distances



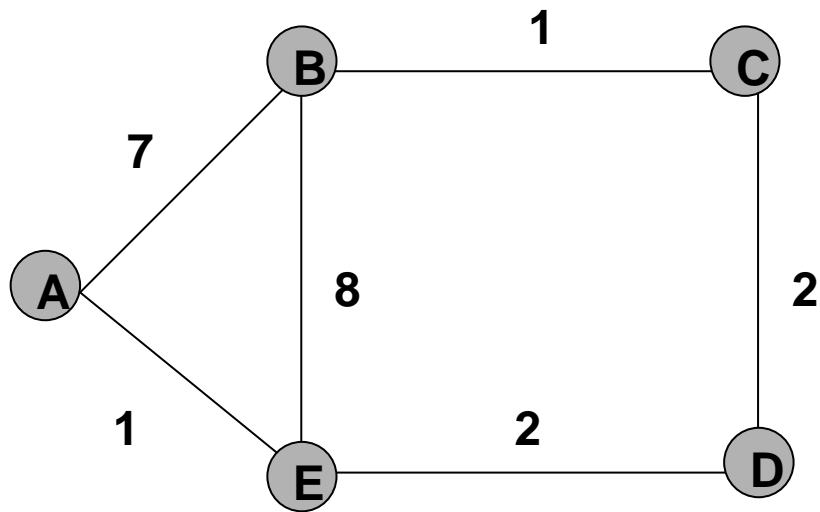
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

E Receives D's Routes



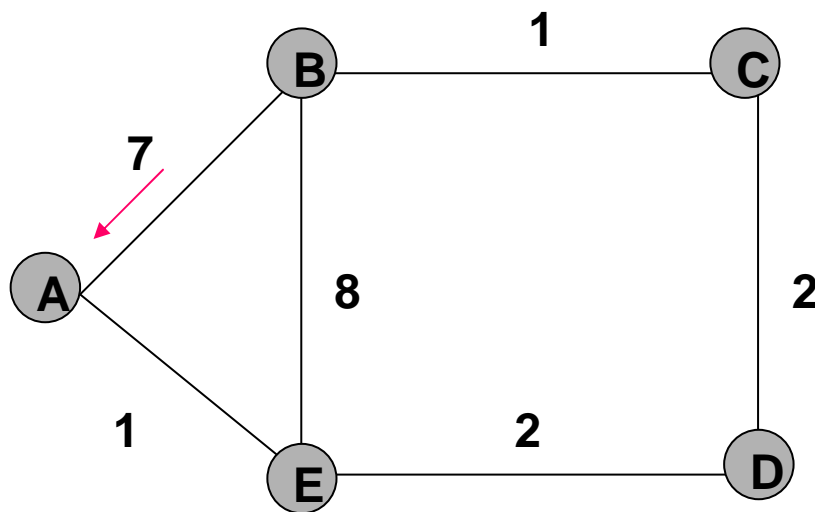
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

E Updates Cost to C



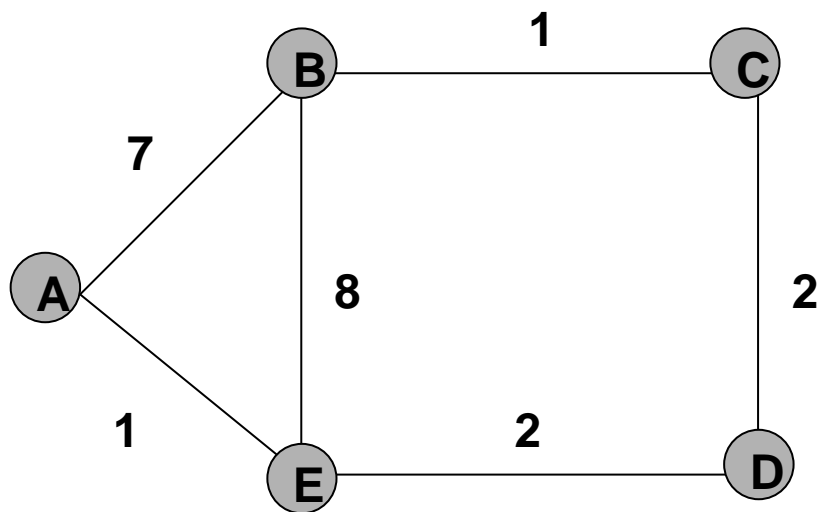
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A Receives B's Routes



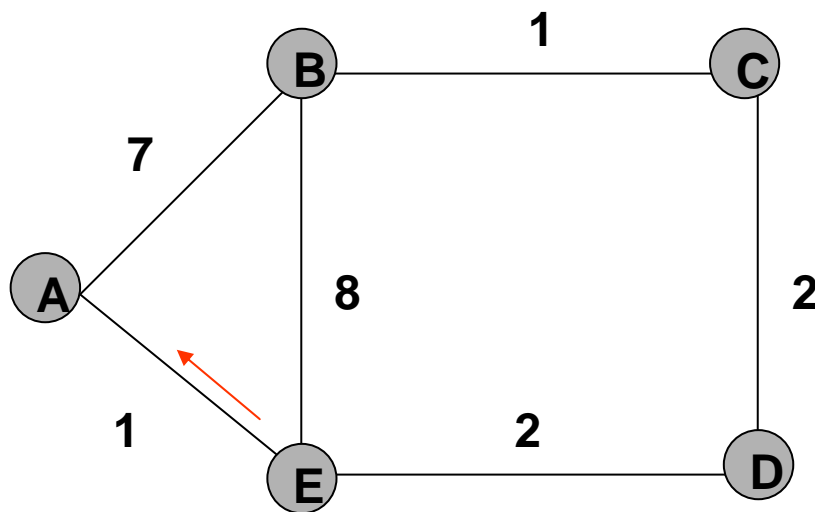
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A Updates Cost to C



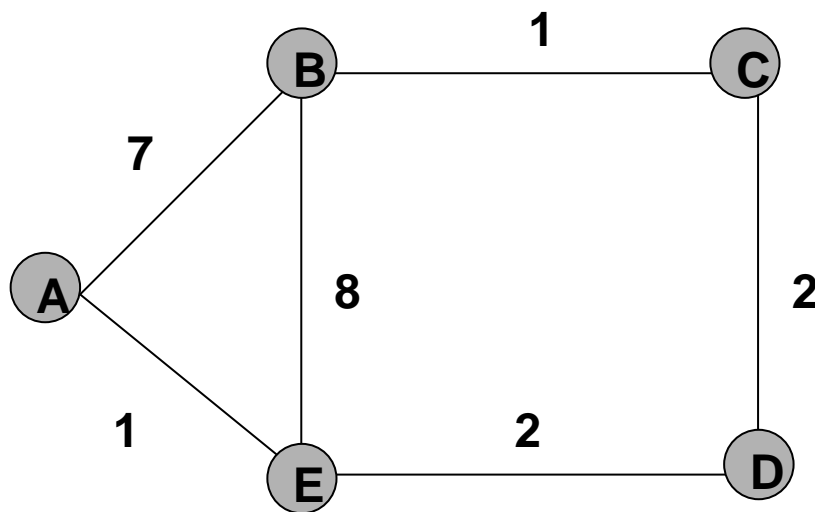
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	8	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A Receives E's Routes



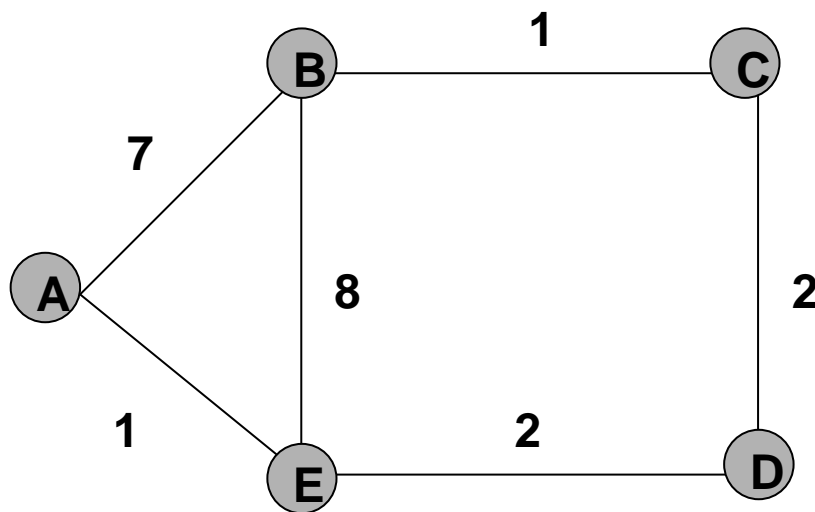
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	8	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A Updates Cost to C and D



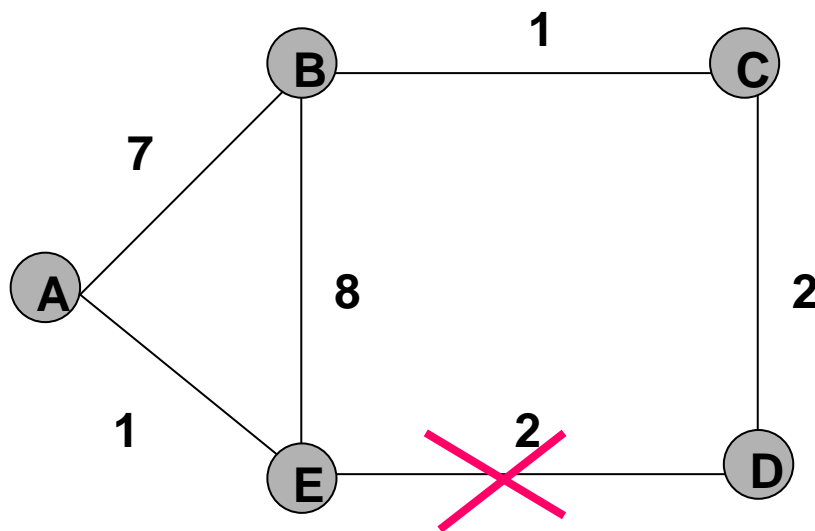
Info at node	Distance to node				
	A	B	C	D	E
A	0	7	5	3	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

Final Distances



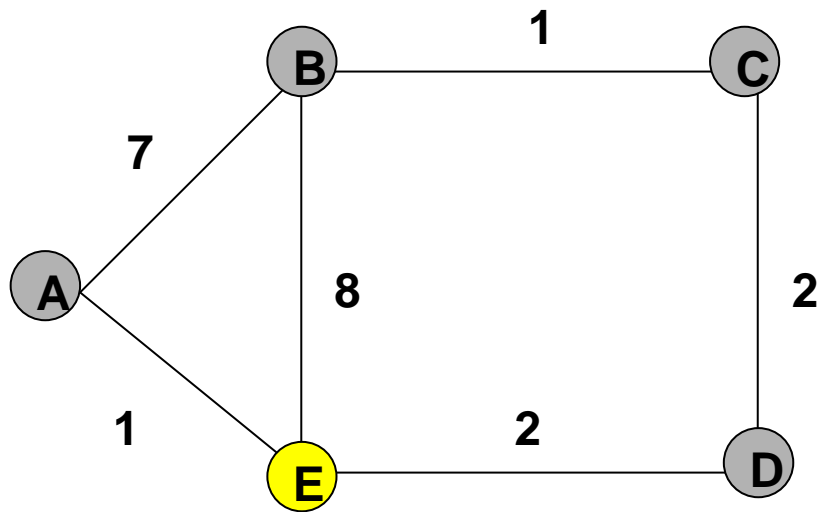
Info at node	Distance to node				
	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

Final Distances After Link Failure



Info at node	Distance to node				
	A	B	C	D	E
A	0	7	8	10	1
B	7	0	1	3	8
C	8	1	0	2	9
D	10	3	2	0	11
E	1	8	9	11	0

View From a Node

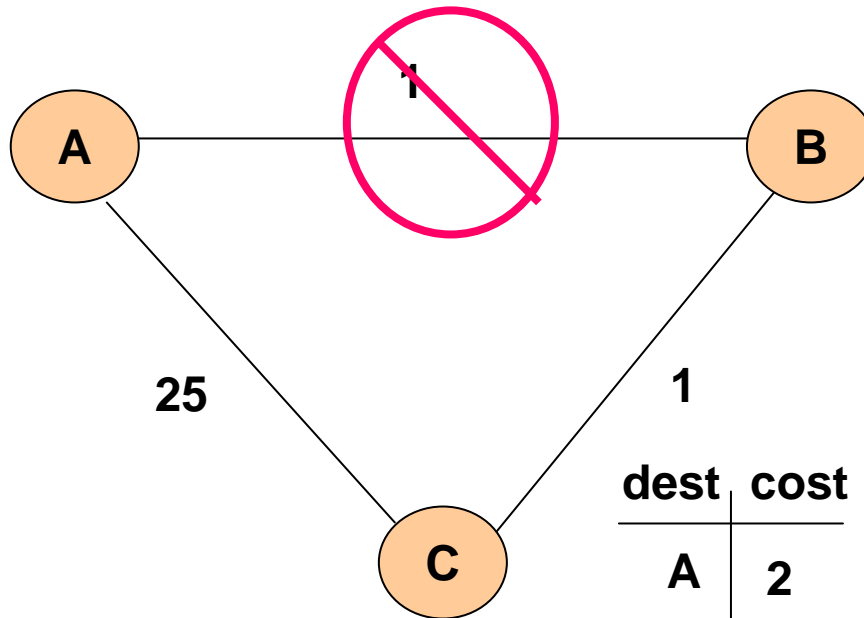


E's routing table

dest	Next hop		
	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Count-to-infinity Problem

dest	cost
B	1
C	2

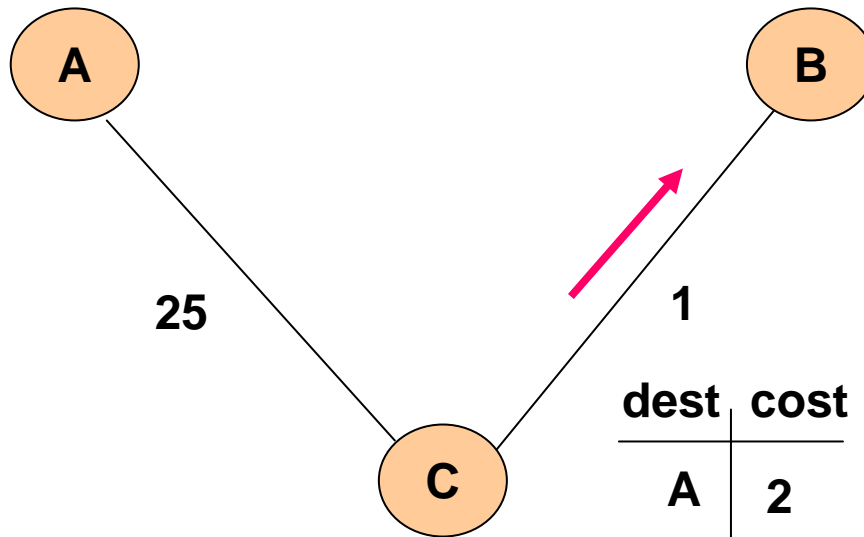


dest	cost
A	1
C	1

dest	cost
A	2
B	1

C Sends Routes to B

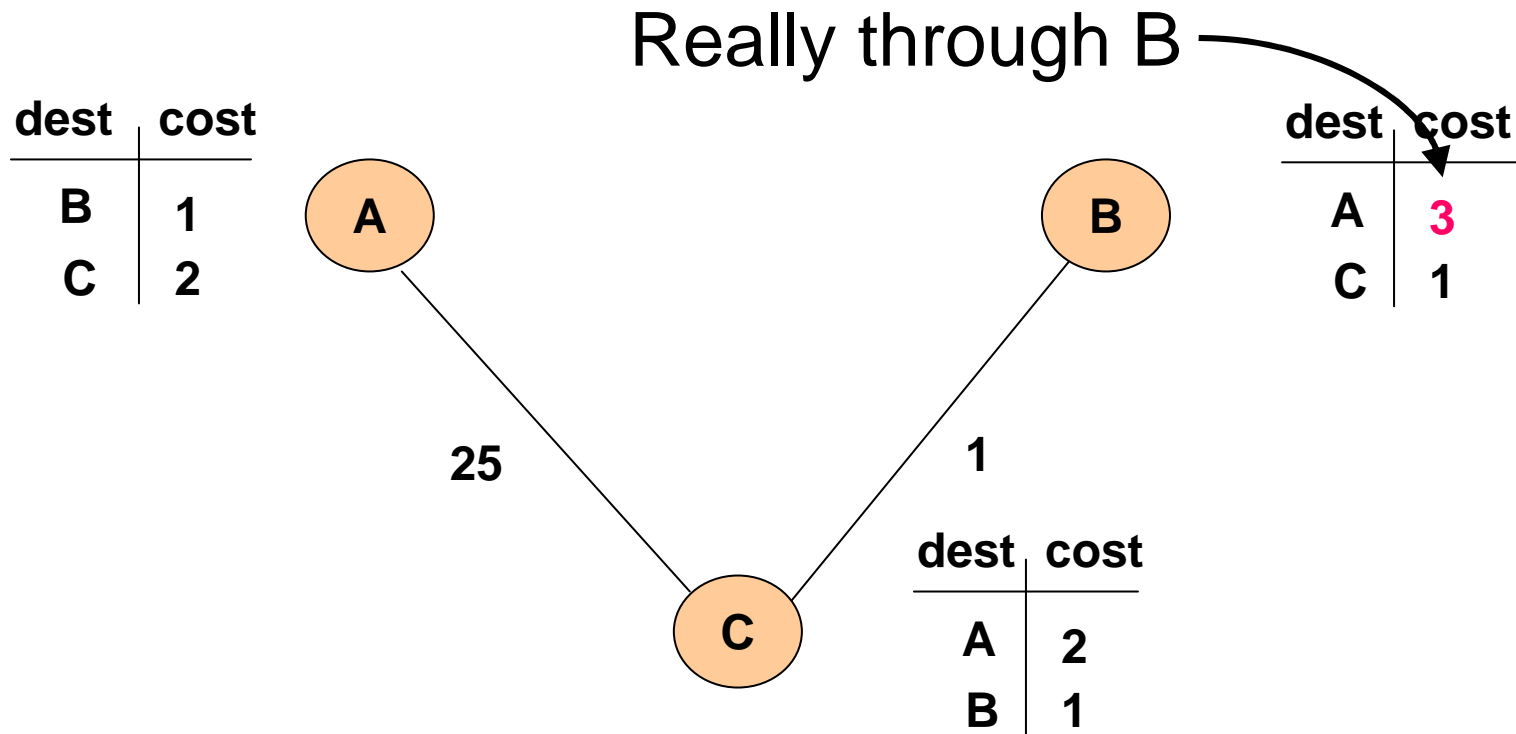
dest	cost
B	1
C	2



dest	cost
A	~
C	1

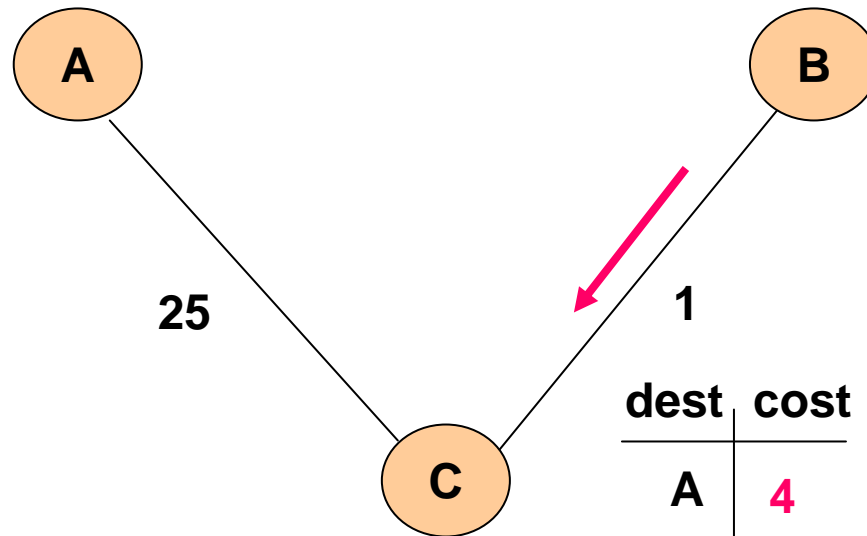
dest	cost
A	2
B	1

B Updates Distance to A



B Sends Routes to C

dest	cost
B	1
C	2

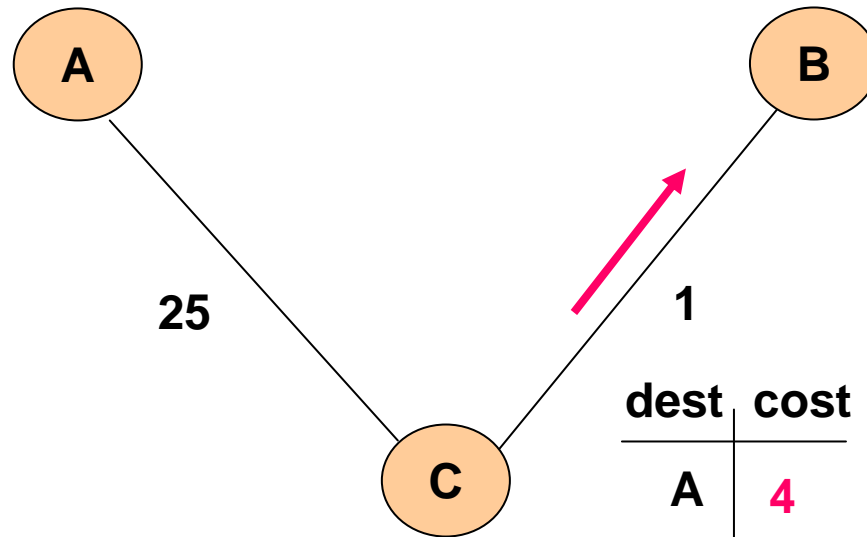


dest	cost
A	3
C	1

dest	cost
A	4
B	1

C Sends Routes to B

dest	cost
B	1
C	2



dest	cost
A	5
C	1

dest	cost
A	4
B	1

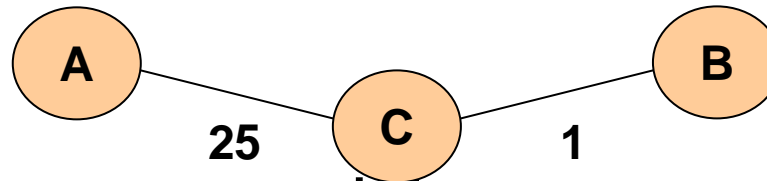
Distance Vector Routing

- Problem
 - Node **X** notices that its link to **Y** is broken
 - Other nodes believe that the route through **X** is still good
 - Mutual deception!

How Are These Loops Caused?

- Observation 1:
 - B's metric **increases**
- Observation 2:
 - C picks B as next hop to A
 - But, the **implicit path** from C to A includes itself!

Solution 1: Holddowns

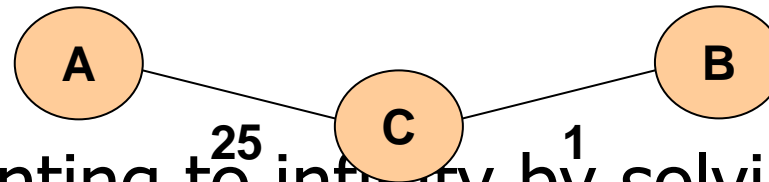


- If metric increases, delay propagating information
 - in our example, B delays advertising route
 - C eventually thinks B's route is gone, picks its own route
 - B then selects C as next hop
- Adversely affects convergence

Heuristics for breaking loops

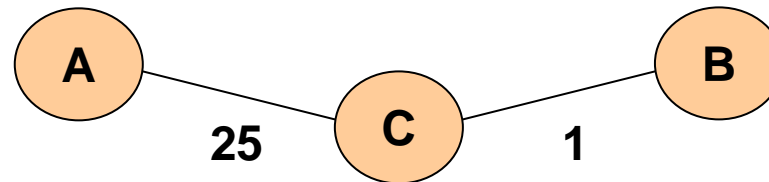
- Set infinity to 16
 - Small limit allows fast completion of “counting to infinity”
 - Limits the size of the network
- Split horizon
 - Avoid counting to infinity by solving “mutual deception” problem
- Split horizon with poisoned reverse
 - “Poison” the routes sent to you by your neighbors
- Sequence numbers on delay estimates

Split Horizon



- Avoid counting to infinity by solving “mutual deception” problem
- Distance Vector with split horizon:
 - when sending an update to node **X**, do not include destinations that you would route through **X**
 - If **X** thinks route is not through you, no effect
 - If **X** thinks route is through you, **X** will timeout route

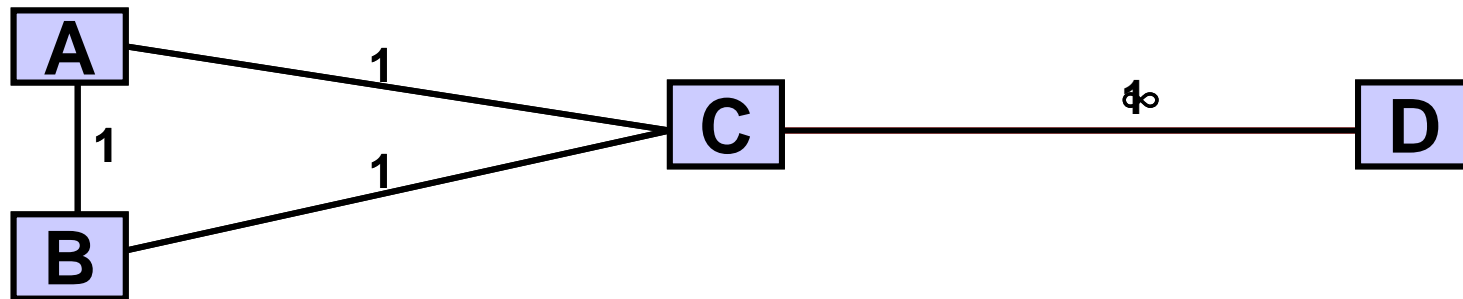
Split Horizon and Poisoned Reverse



- Distance Vector with Split Horizon and Poisoned Reverse:
 - When sending update to node **X**, include destinations that you would route through **X** with distance set to infinity
 - Don't need to wait for **X** to timeout
- Problem:
 - Router on edge of Internet would need to include infinity route for all outside destinations on Internet!

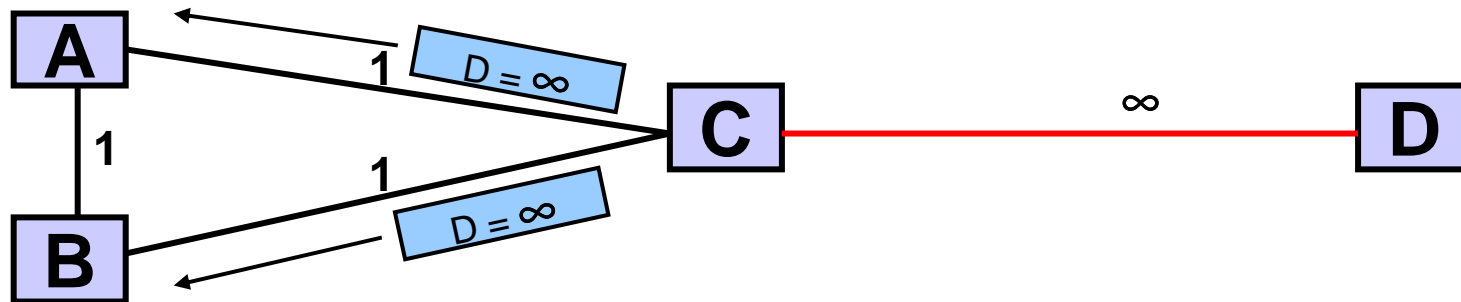
Split Horizon

- Split Horizon (with or without poisoned reverse) may still allow some routing loops and counting to infinity
 - guarantees no 2-node loops
 - can still be fooled by 3-node (or larger) loops
- Consider link failure from **C** to **D**



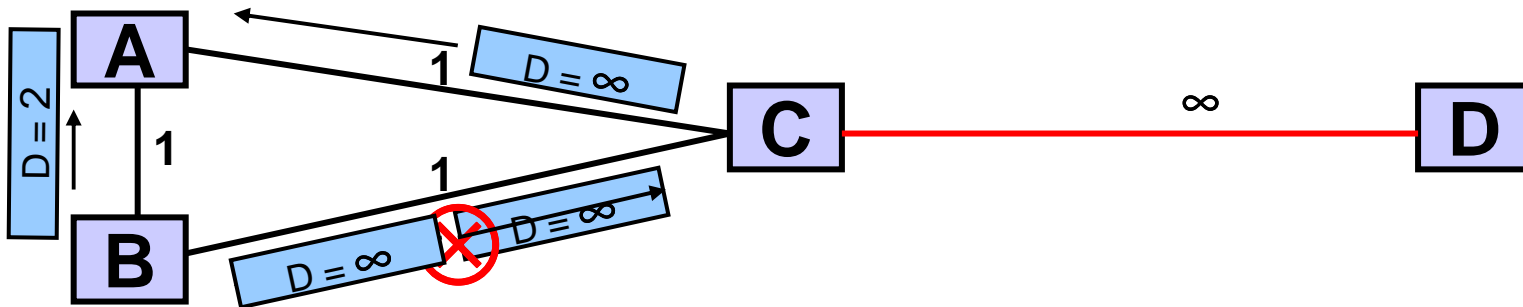
Split Horizon

- Initial routing table entries for route to **D**:
 - A** 2 via **C**
 - B** 2 via **C**
 - C** 1
- **C** notices link failure and changes to infinity
- Now **C** sends updates to **A** and **B**:
 - to **A**: infinity
 - to **B**: infinity



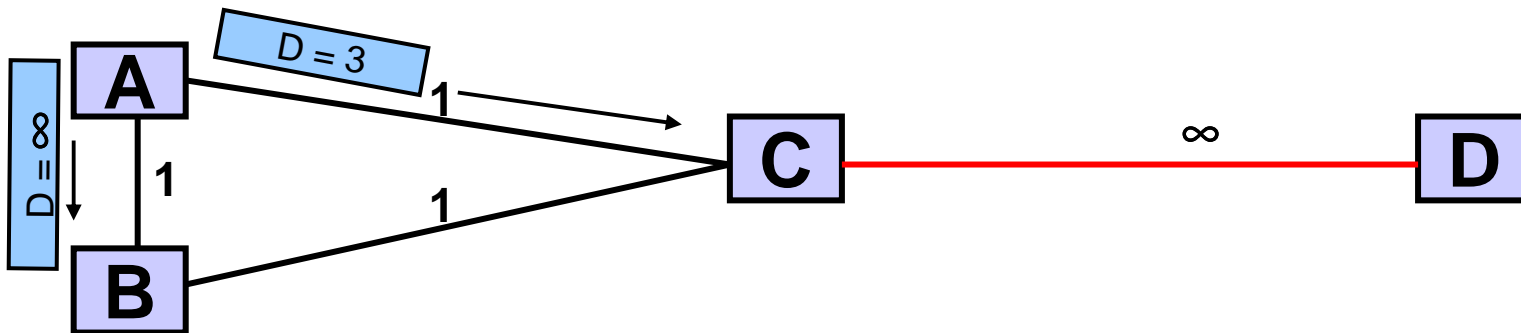
Split Horizon

- Suppose update to **B** is lost
- New tables:
 - A** unreachable
 - B** 2 via **C**
 - C** unreachable
- Now **B** sends its periodic routing update:
 - to **C**: infinity (poisoned reverse)
 - to **A**: 2



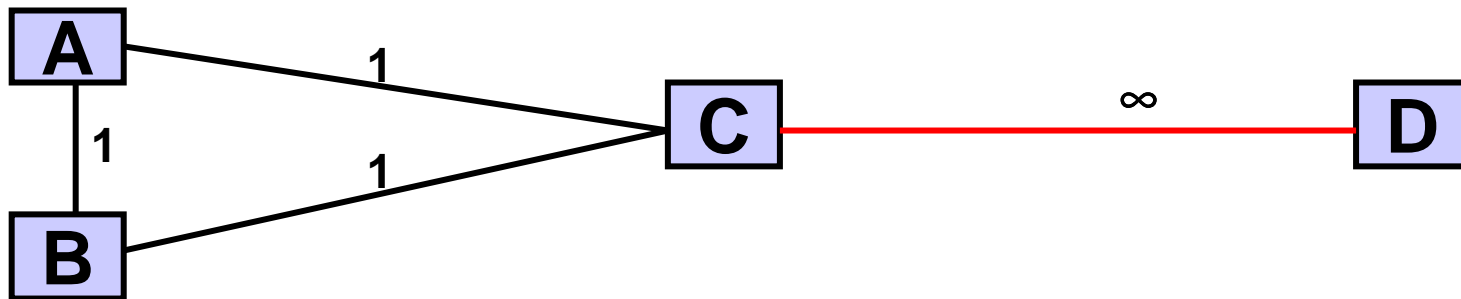
Split Horizon

- New tables for route to **D**:
 - A** 3 via **B**
 - B** 2 via **C**
 - C** unreachable
- Finally A sends its periodic routing update:
 - to **B**: infinity (poisoned reverse)
 - to **C**: 3

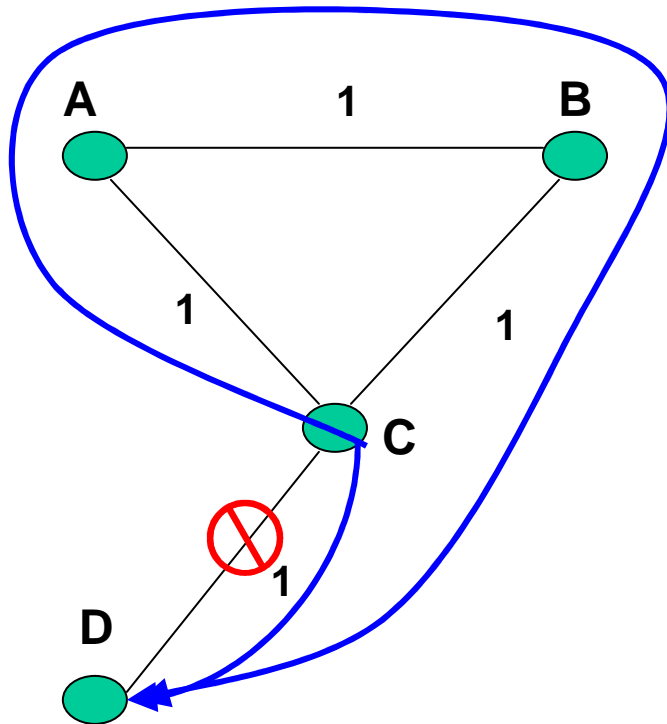


Split Horizon

- New tables for route to **D**:
 - A** 3 via **B**
 - B** 2 via **C**
 - C** 4 via **A**
- A, B and C will still continue to count to infinity



Example Where Split Horizon Fails



- Link breaks
 - C marks D as unreachable and reports that to A and B.
- Suppose A learns it first.
 - A now thinks best path to D is through B.
 - A reports a route of cost=3 to C.
- C thinks D is reachable through A at cost 4 and reports that to B.
- B reports a cost 5 to A who reports new cost to C.
- etc...

Avoiding Counting to Infinity

- Select loop-free **paths**
- One way of doing this:
 - Each route advertisement carries entire path
 - If a router sees itself in path, it rejects the route
- BGP does it this way
- Space proportional to diameter

Loop Freedom at Every Instant

- Does keeping paths avoid all loops?
 - No! **Transient** loops are still possible
 - Why? Because path information may be stale
- Only way to fix this
 - Ensure that you have up-to-date information by explicitly querying

Distance Vector in Practice

- RIP and RIP2
 - uses split-horizon/poison reverse
- BGP/IDRP
 - propagates entire path
 - path also used for affecting policies
- AODV
 - “on-demand” distance vector protocol for wireless networks
 - Only maintain distance vectors along paths to destinations that you need to reach

Distance Vector Routing

- Problem
 - Information propagates slowly
 - One period per hop for new routes
 - Count to infinity to detect lost routes

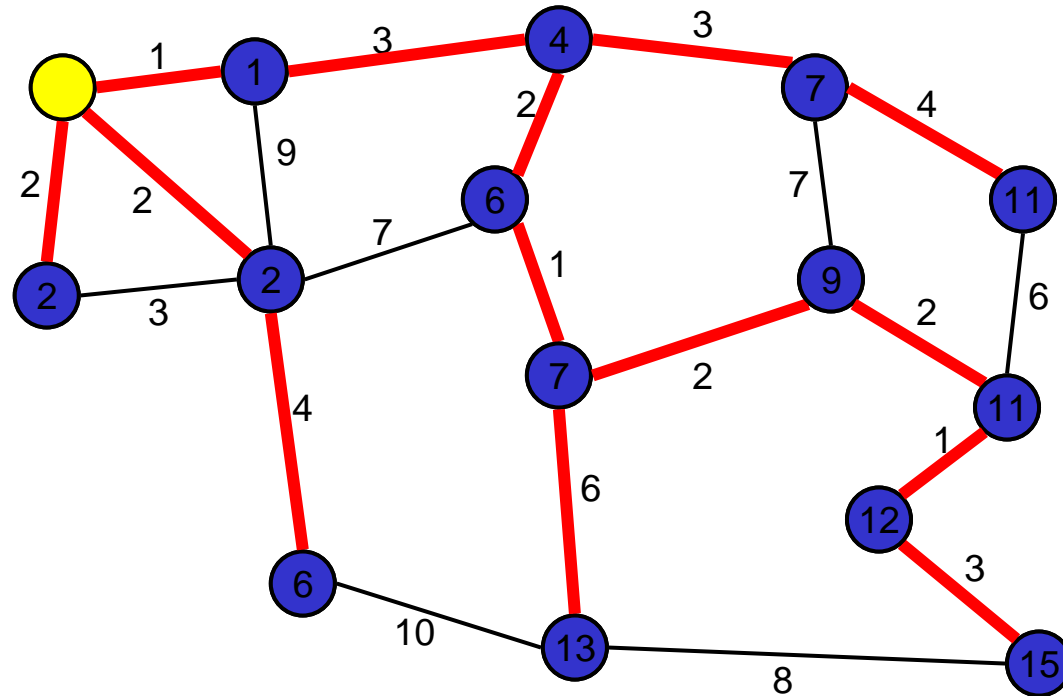
Dijkstra's Algorithm

- Given
 - Directed graph with edge weights (distances)
- Calculate
 - Shortest paths from one node to all others

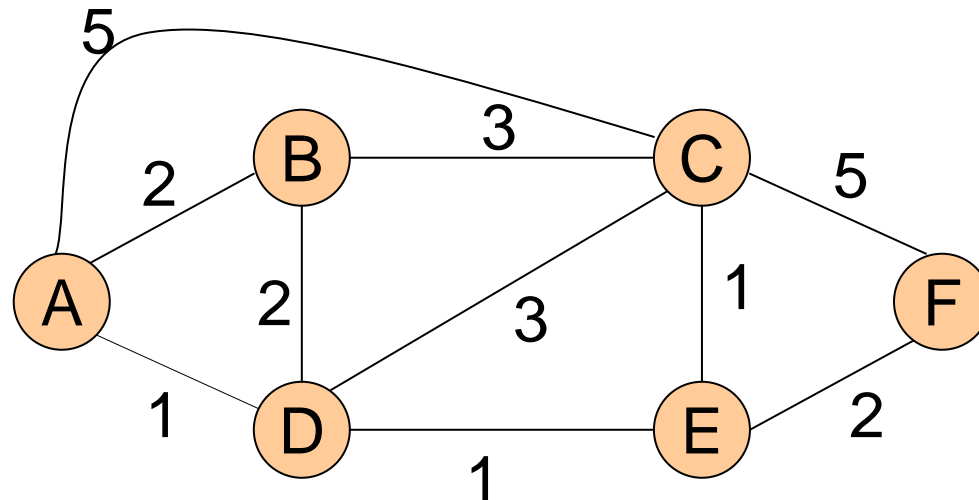
Dijkstra's Algorithm

- Greedily grow set C of confirmed least cost paths
- Initially $C = \{\text{source}\}$
- Loop N-1 times
 - Determine the node M outside C that is closest to the source
 - Add M to C and update costs for each node P outside C
 - Is the path (source \rightarrow ... \rightarrow M \rightarrow P) better than the previously known path for (source \rightarrow P)?
 - If YES
 - Update cost to reach P

Dijkstra's Algorithm

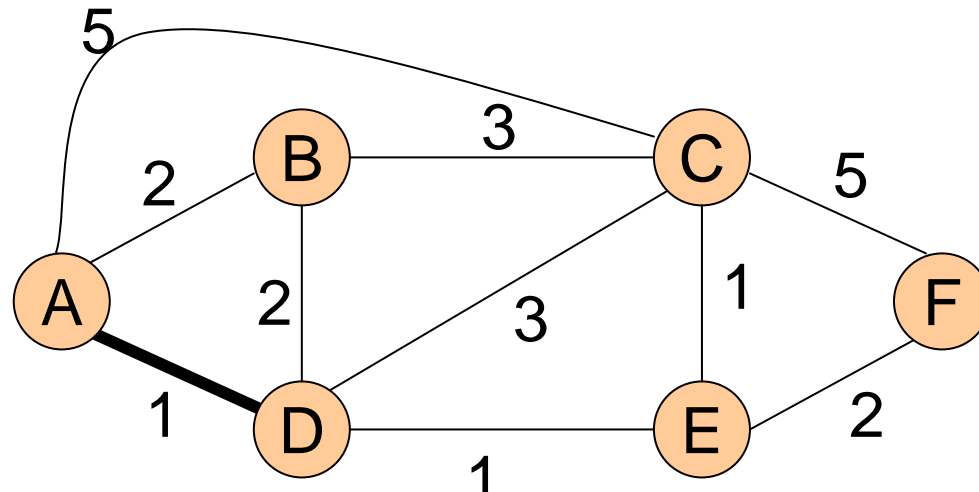


Example



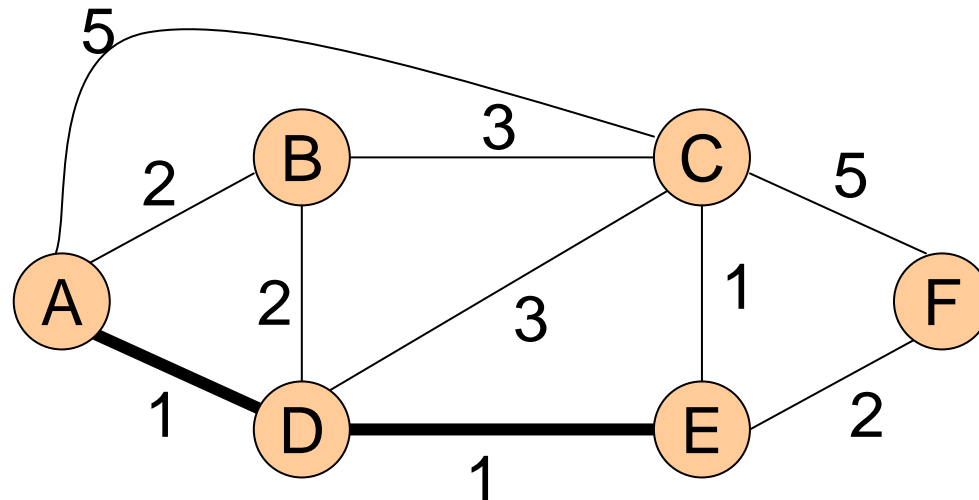
step	SPT	B	C	D	E	F
		D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~

Example



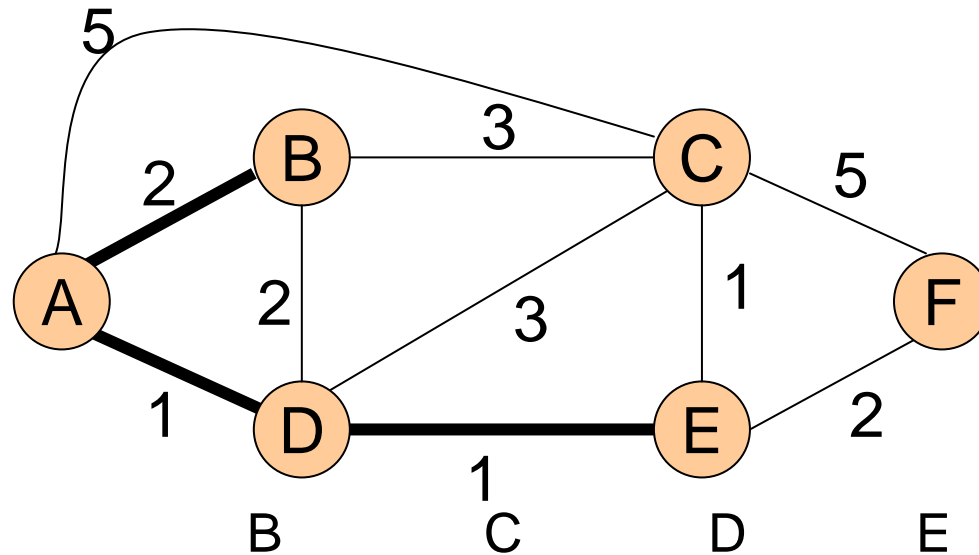
step	SPT	B	C	D	E	F
		D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~

Example



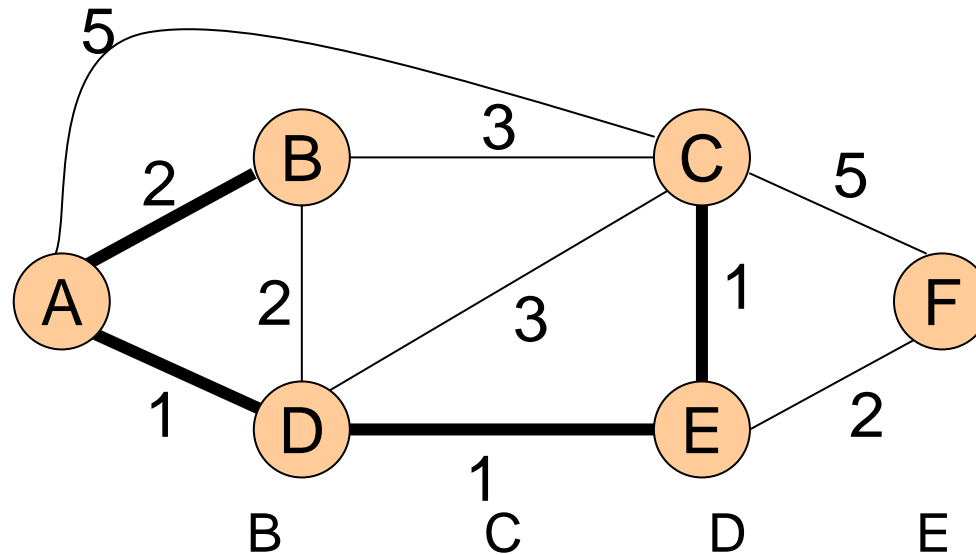
step	SPT	B	C	D	E	F
		D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E

Example



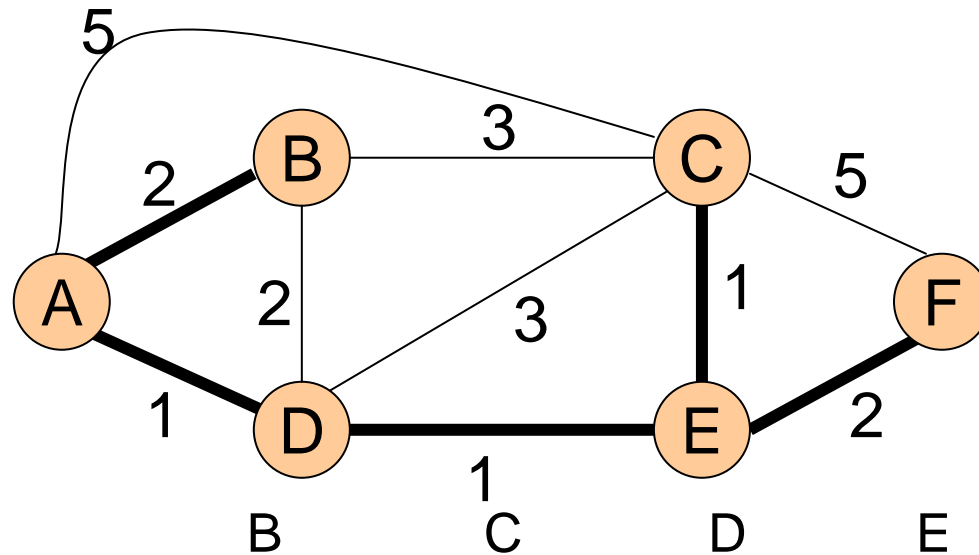
step	SPT	D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E
3	ADEB		3, E			4, E

Example



step	SPT	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	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E
3	ADEB		3, E			4, E
4	ADEBC					4, E

Example



step	SPT	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	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E
3	ADEB		3, E			4, E
4	ADEBC					4, E

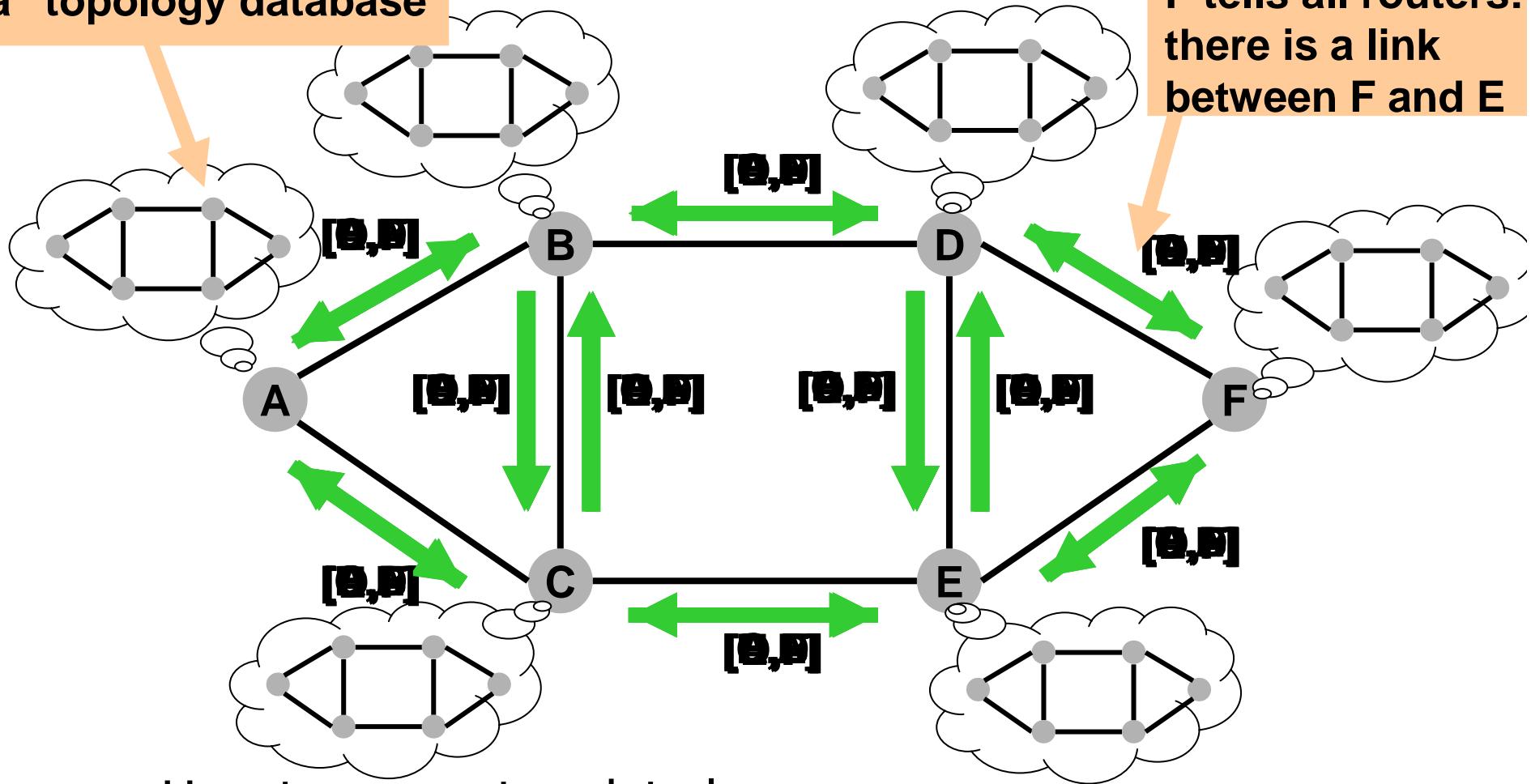
Link State Routing

- Strategy
 - Send all nodes information about directly connected links
 - Status of links is flooded in link state packets (LSPs)
- Each LSP carries
 - ID of node that created the LSP
 - Vector of <neighbor, cost of link to neighbor> pairs for the node that created the LSP
 - Sequence number
 - Time-to-live (TTL)
- Each node maintains a list of (ideally all) LSP's and runs Dijkstra's algorithm on the list

Link state: update propagation

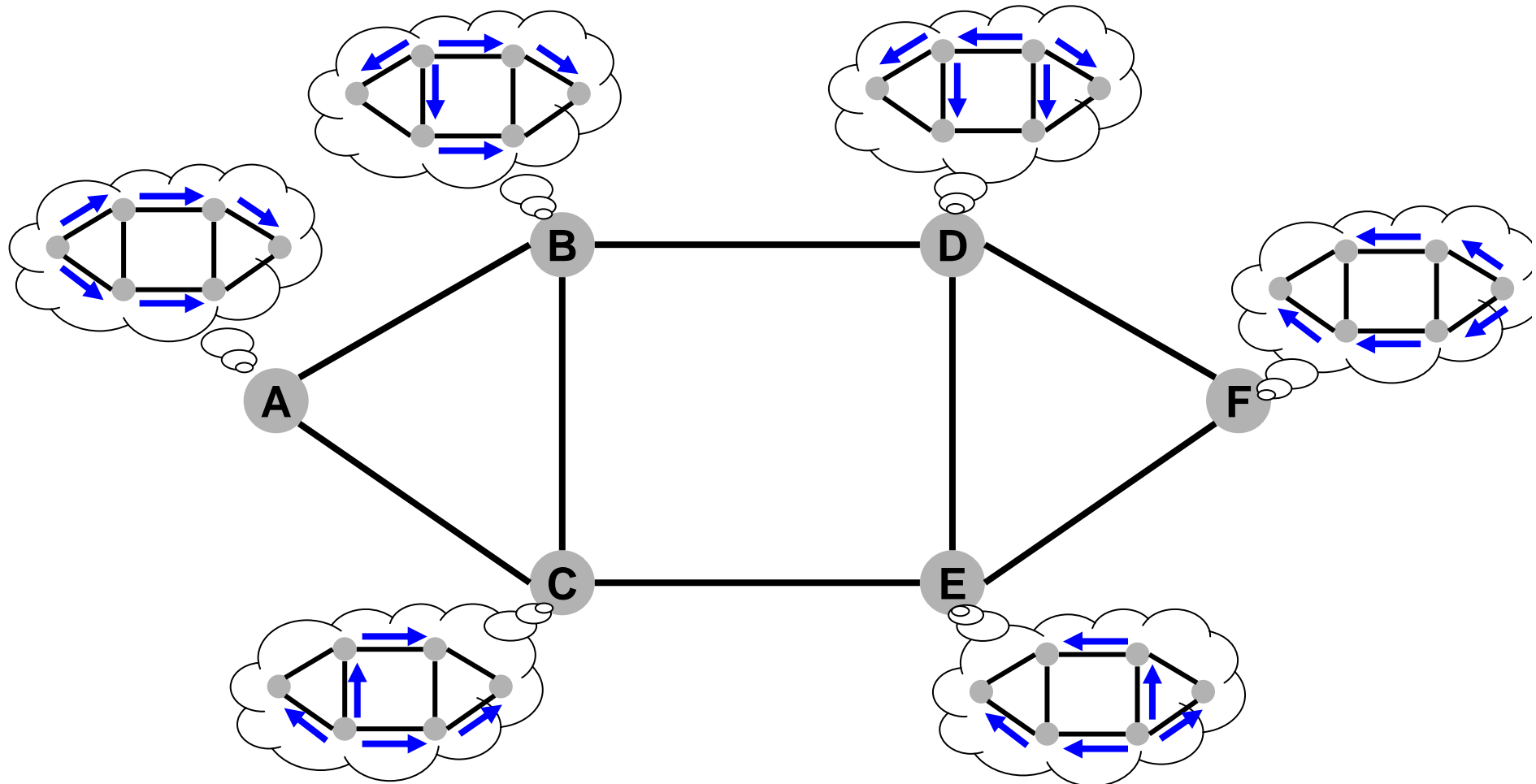
Each node maintains a "topology database"

F tells all routers: there is a link between F and E



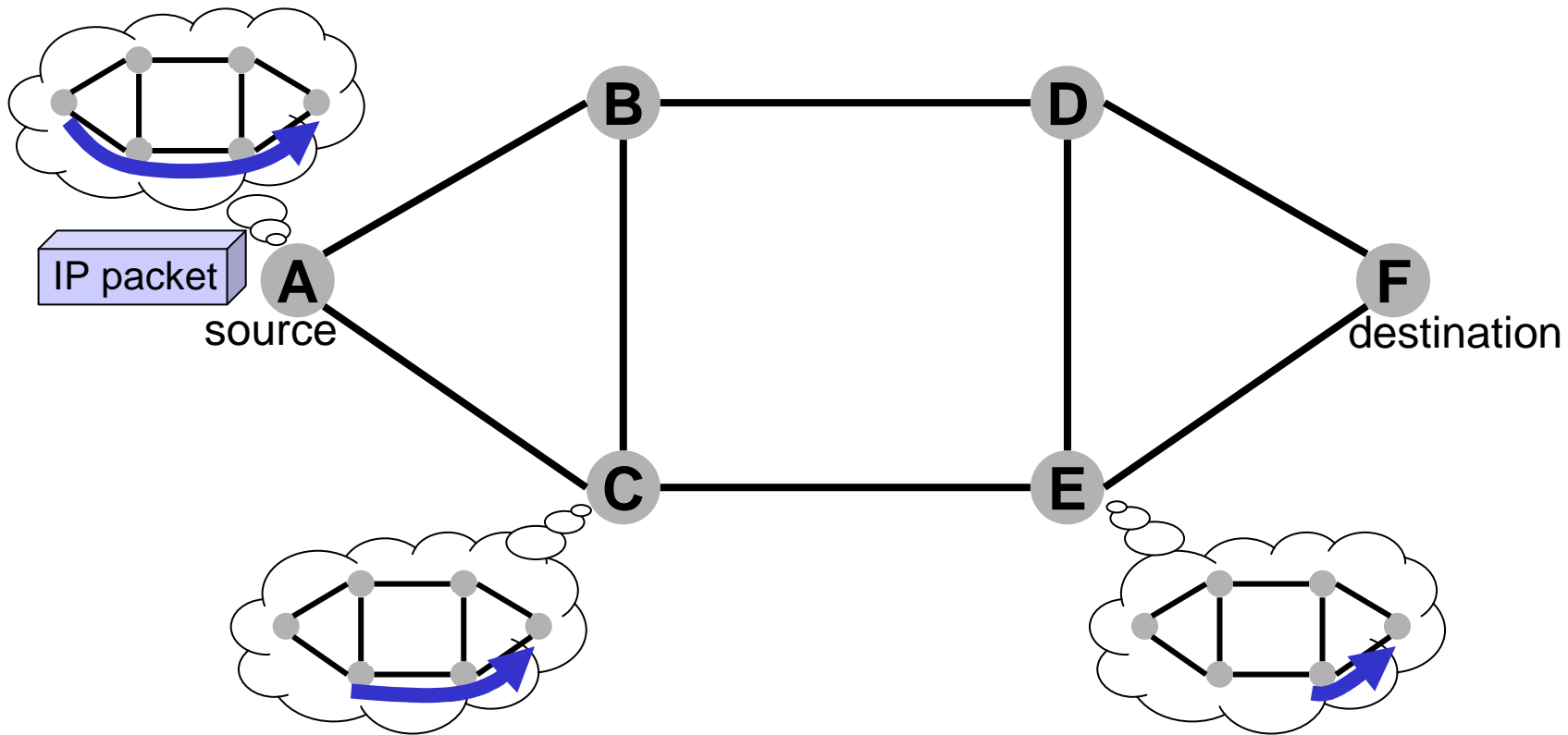
- How to prevent update loops:
- How to bring up new node:

Link state: route computation



- Each router computes shortest path tree, rooted at that router
- Determines next-hop to each dest, publish to forwarding table
- Operators can assign link costs to control path selection

Link-state: packet forwarding



- Downsides of link-state:
 - Lesser control on policy (certain routes can't be filtered), more cpu
 - Increased visibility (bad for privacy, but good for diagnostics)

Link State Routing

- LSP must be delivered to all nodes
- Information acquisition via reliable flooding
 - Create local LSP periodically with increasing sequence number
 - Send local LSP to all immediate neighbors
 - Forward LSP (if it has a new sequence number than previously received) out on all other links
- Why not just use TCP between every pair of routers?

Basic Steps

- Each node assumed to know state of links to its neighbors
- **Step 1:** Each node broadcasts its state to all other nodes
- **Step 2:** Each node locally computes shortest paths to all other nodes from global state

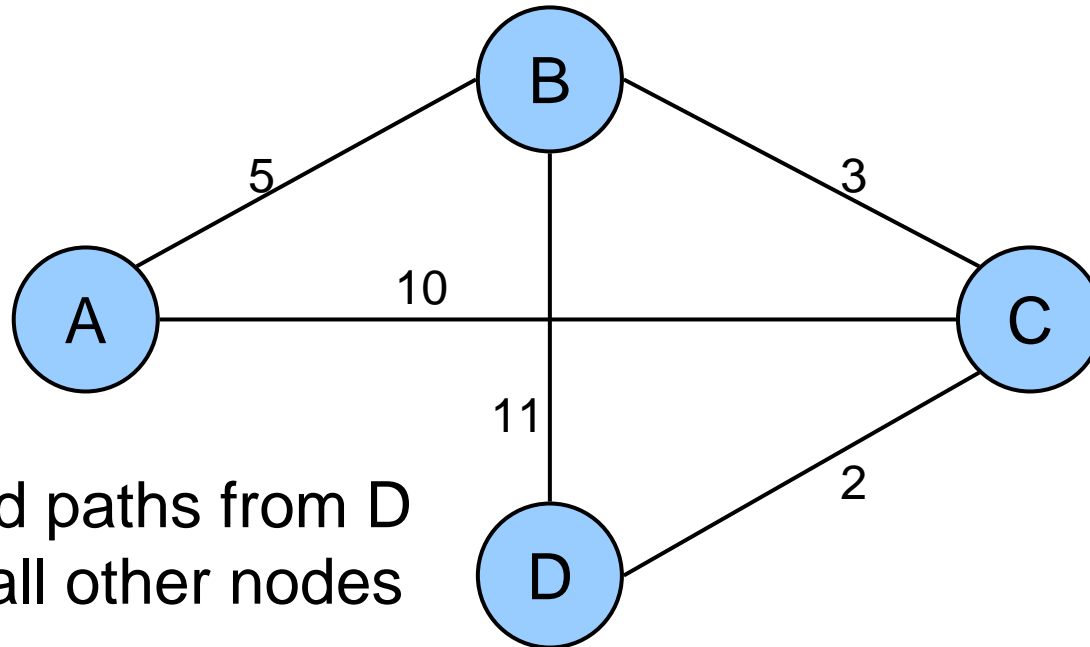
Reliable Flooding

- When i receives LSP from j :
 - If LSP is the most recent LSP from j that i has seen so far
 - i saves it in database and forwards a copy on all links except link LSP was received on
 - Otherwise, discard LSP

Link State Routing

- At each router, perform a forward search algorithm (variation of Dijkstra's)
- Router maintains two lists
 - Confirmed (nodes I know the shortest paths to)
 - Tentative (nodes that are adjacent to Confirmed nodes)
- Each list contains triplets
 - <destination, cost, nexthop>

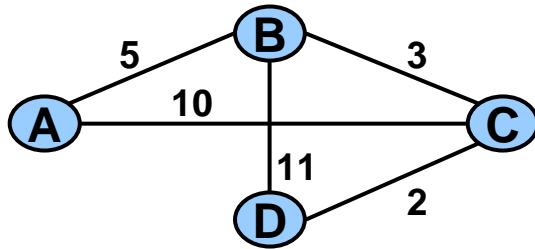
Link State Routing



Find paths from D
to all other nodes

Link State Routing

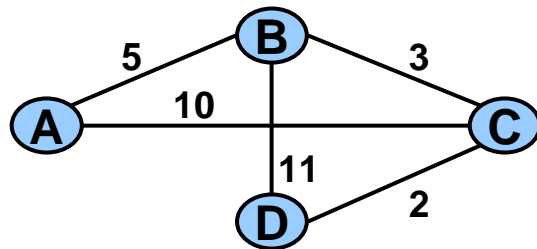
Step	Confirmed	Tentative
1.		
2.		
3.		
4.		



Step	Confirmed	Tentative
5		
6		
7		

Link State Routing

Step	Confirmed	Tentative
1.	(D,0,-)	
2.	(D,0,-)	(B,11,B) (C,2,C)
3.	(D,0,-) (C,2,C)	(B,11,B)
4.	(D,0,-) (C,2,C)	(B,5,C) (A,12,C)

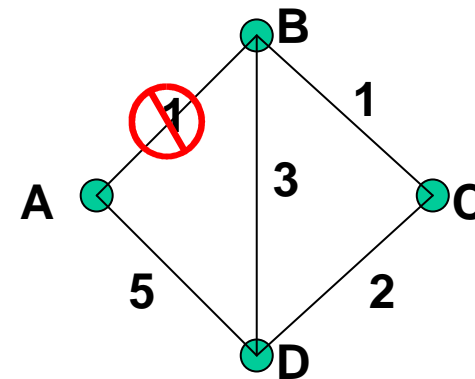


Step	Confirmed	Tentative
5	(D,0,-) (C,2,C) (B,5,C)	(A,12,C)
6	(D,0,-) (C,2,C) (B,5,C)	(A,10,C)
7	(D,0,-) (C,2,C) (B,5,C) (A,10,C)	

Link State Characteristics

- With consistent LSDBs, all nodes compute consistent loop-free paths
- Limited by Dijkstra computation overhead, space requirements
- Can still have transient loops

Packet from C->A
may loop around BDC

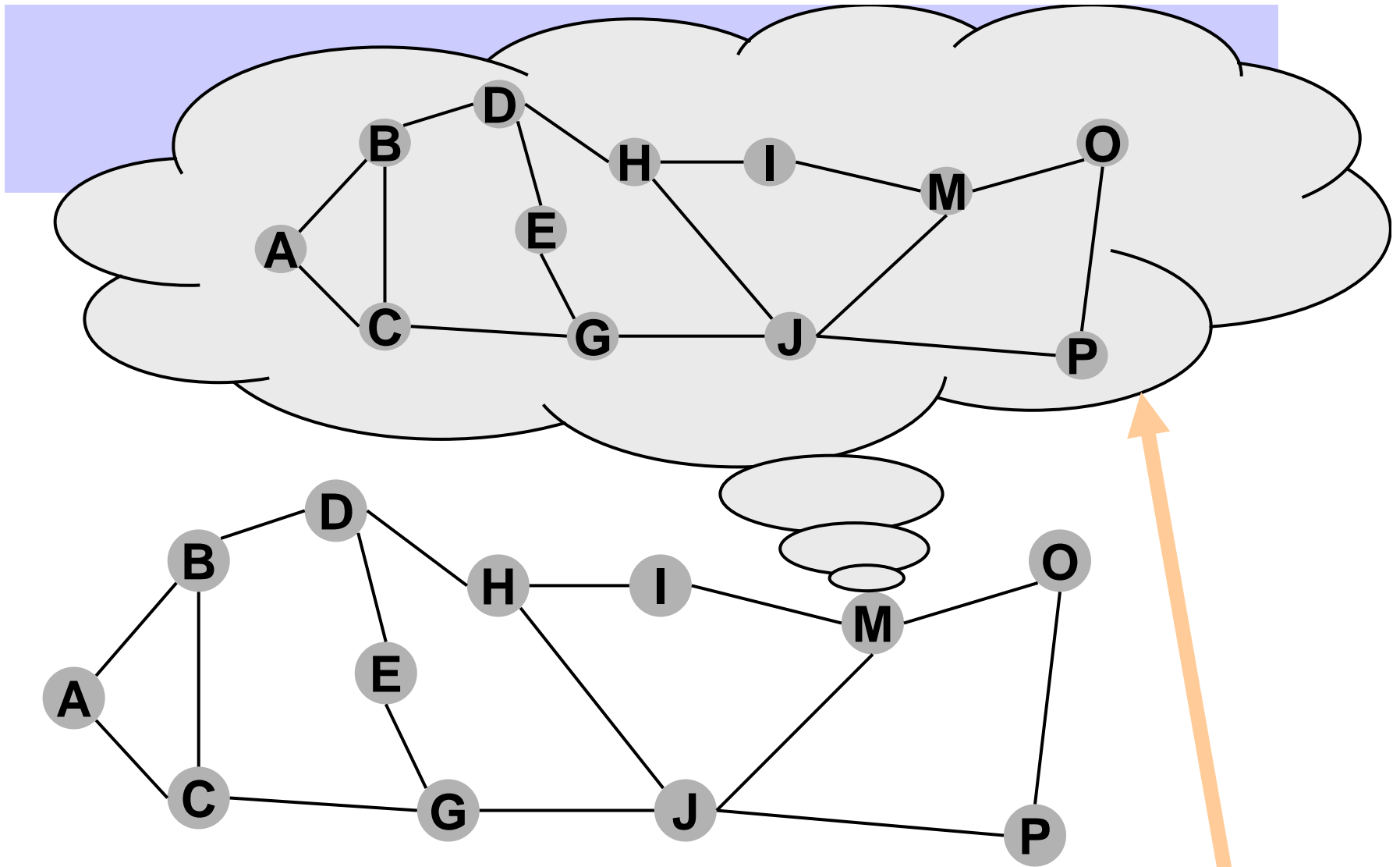


Link State Routing

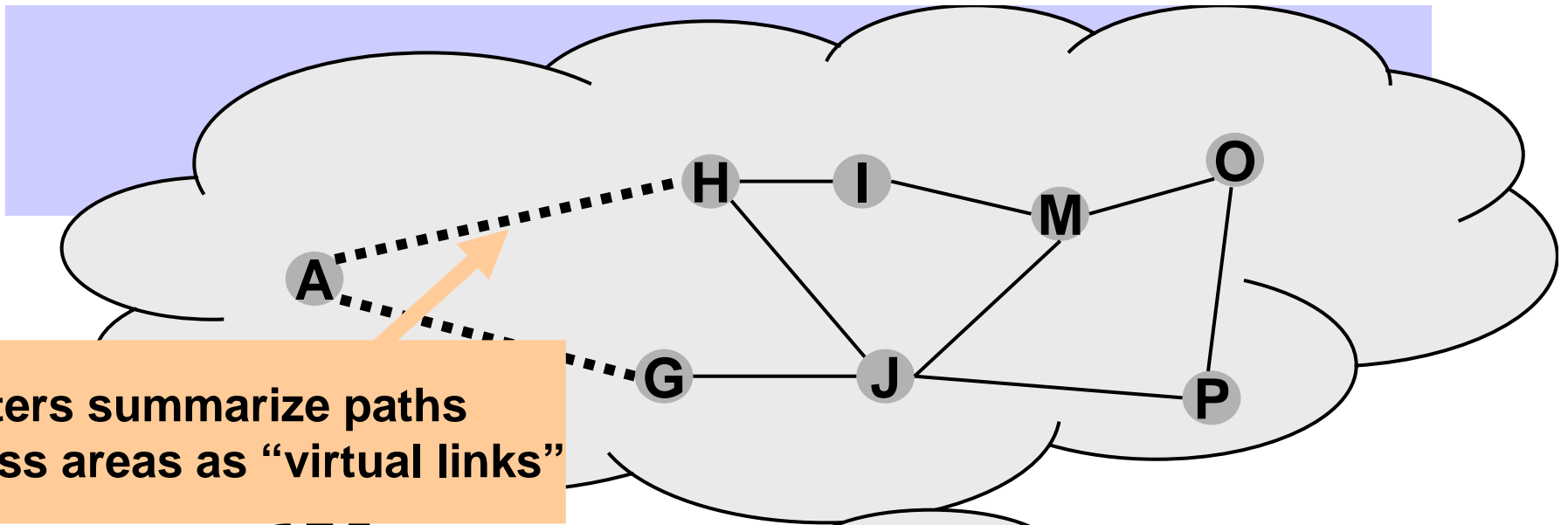
- Advertise routes to “IP prefixes” (blocks of IP addresses)
- Intermediate System-Intermediate System (IS-IS)
 - Designed for DECnet
 - Adopted by ISO for connectionless network layer protocol (CNLP)
 - Used in NSFNET backbone
 - Used in some ISPs, some digital cellular systems
- Open shortest path first (OSPF)
 - Defined in RFC 5340
 - Used in some ISPs

OSPF

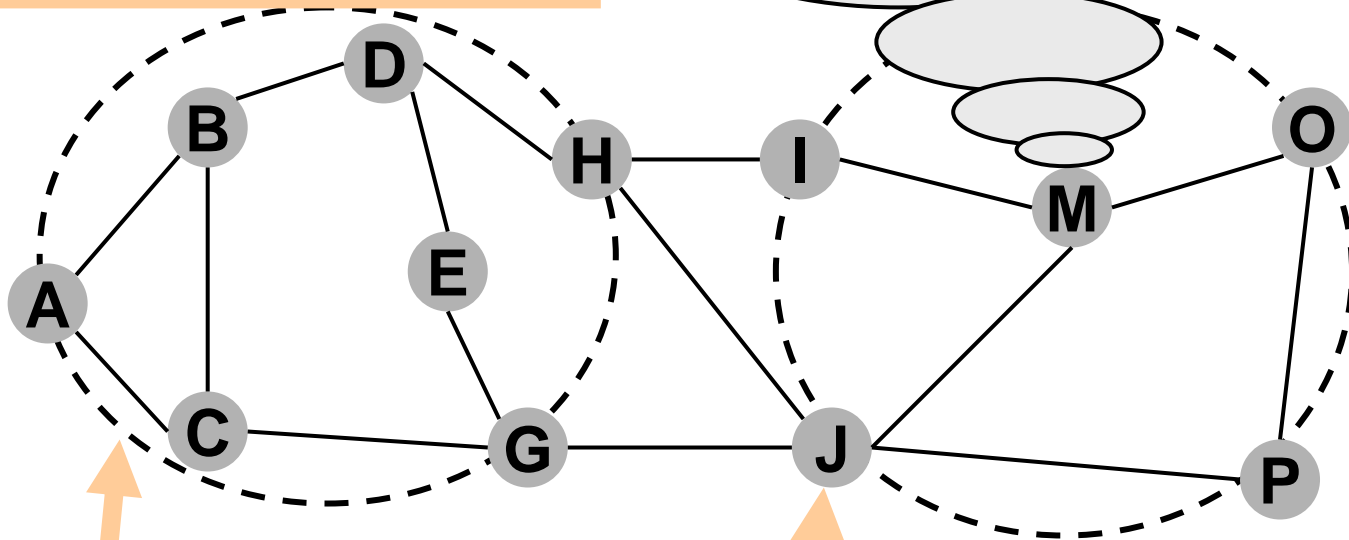
- Authentication of routing messages
 - Encrypted communication between routers
- Additional hierarchy
 - Domains are split into areas
 - Routers only need to know how to reach every node in a domain
 - Routers need to know how to get to the right area
 - Load balancing
 - Allows traffic to be distributed over multiple routes



In regular link-state, routers maintain map of entire topology



Routers summarize paths across areas as “virtual links”



Aggregate groups of routers into “areas”

“Border routers” generate “summary LSPs” to reach other border routers

Tradeoffs of hierarchical routing

- Advantages: scalability
 - Reduce size of link-state database
 - Isolate rest of network from changes/faults
- Disadvantages
 - Complexity
 - Extra configuration effort
 - Requires tight coupling with address assignment
 - Inefficiency
 - One link change may affect multiple path costs
 - Summarization hides shorter paths

LS vs. DV

- Distance Vector (DV)
 - Send everything you know to your neighbors
- Link State (LS)
 - Send info about your adjacent links to everyone
- Which one's better?
- Message exchange
 - LS: $O(nE)$
 - DV: $O(nd)$ for d destinations, worst-case $O(d*n!)$
 - But per-node computation time less in DV

LS vs. DV

- LS typically used *within* ISPs because
 - Faster convergence (usually)
 - Simpler troubleshooting
- DV typically used *between* ISPs because
 - Can support more flexible policies
 - Can avoid exporting routes
 - Can hide private regions of topology

LS vs. DV: Robustness

- LS can broadcast incorrect/corrupted LSP
 - Localized problem
 - But across multiple destinations
- DV can advertise incorrect paths to all destinations
 - Incorrect calculation can spread to entire network
 - But only for that destination
- Soft-state vs. Hard-state approaches
 - Should we periodically refresh? Or rely on routers to locally maintain their state correctly?

Traffic engineering with routing protocols

- Load balancing
 - Some hosts/networks/paths are more popular than others
 - Need to shift traffic to avoid overrunning capacity
 - Why is this a different problem from congestion control?
- Avoiding oscillations
 - What if metrics are a function of offered load?
 - Causes dependencies across paths

Challenge #1: Avoiding oscillations

- Choice of link cost defines traffic load
 - Low cost = high probability link belongs to SPT
 - Will attract traffic, which increases cost
- Main problem: convergence
 - Avoid oscillations
 - Achieve good network utilization

Metrics

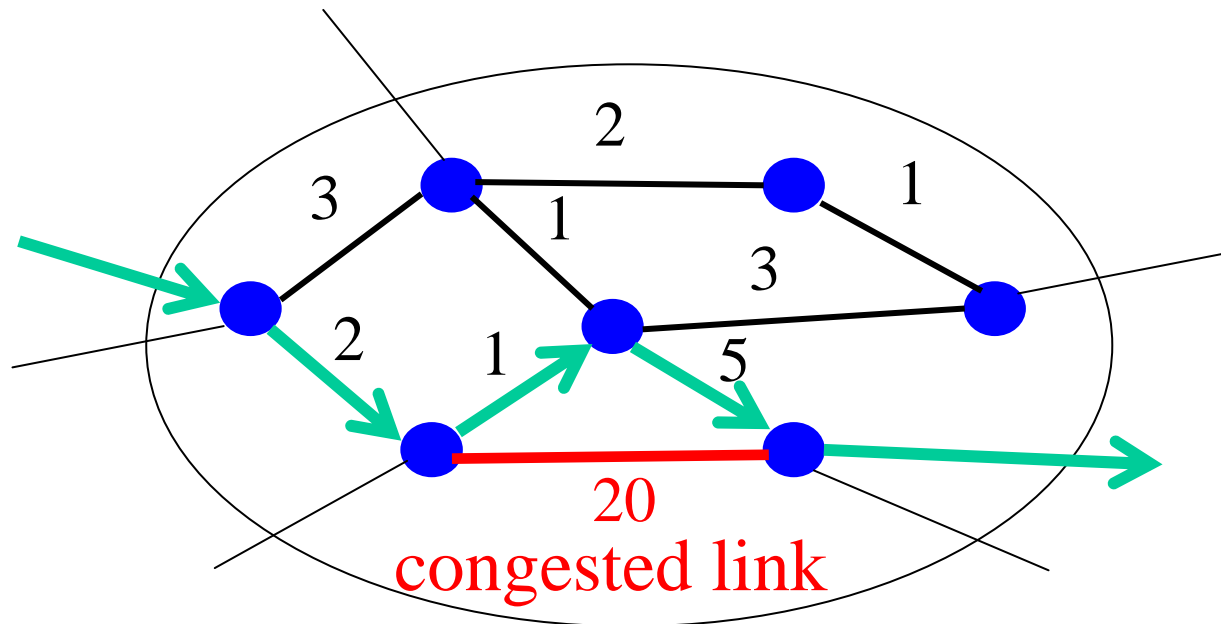
- Capture a general notion of distance
- A heuristic combination of
 - Distance
 - Bandwidth
 - Average traffic
 - Queue length
 - Measured delay

Metric Choices

- Fixed metrics (e.g., hop count)
 - Good only if links are homogeneous
 - Definitely not the case in the Internet
- Static metrics do not take into account
 - Link delay
 - Link capacity
 - Link load (hard to measure)
- But, can improve stability

Original ARPANET Algorithm

- Shortest-path routing based on link metrics
- Instantaneous queue length plus a constant
- Distance vector routing for shortest paths



Original ARPANET Algorithm

- Light load
 - Delay dominated by the constant part (transmission and propagation delay)
- Medium load
 - Queuing delay no longer negligible
 - Moderate traffic shifts to avoid congestion
- Heavy load
 - Very high metrics on congested links
 - Busy links look bad to all of the routers
 - All routers avoid the busy links
 - Routers may send packets on longer paths

Second ARPANET Algorithm (1979)

- Averaging of link metric over time
 - Old: Instantaneous delay fluctuates a lot
 - New: Averaging reduces the fluctuations
- Link-state protocol instead of DV
 - Old: DV led to loops
 - New: Flood metrics and let each router compute shortest paths
- Reduce frequency of updates
 - Old: Sending updates on each change is too much
 - New: Send updates if change passes a threshold

Problem #2: Load balancing

- Conventional static metrics:
 - Proportional to physical distance
 - Inversely proportional to link capacity
- Conventional dynamic metrics:
 - Tune weights based on the offered traffic
 - Network-wide optimization of link-weights
 - Directly minimizes metrics like maximum link utilization

Traffic engineering in IP networks

- Question: given traffic loads arriving at the network, how can we assign costs to links, to achieve desired balance of traffic across routers?
- Formulated as an optimization problem
 - Input parameters: network topology, input traffic matrix
 - Input constraints: minimize delay, maintain 70% average spare capacity on links
 - Compute: assignment of weights to links

Application to AT&T's backbone network

- Performance of the optimized weights
 - Search finds a good (approximate) solution within a few minutes
 - Much better than link capacity or physical distance
- How AT&T changes the link weights
 - Maintenance from Midnight to 6am ET
 - Predict effects of removing links from network
 - Reoptimize links to avoid congestion
 - Configure new weights before disabling equipment (costing-out)